云数据库 GaussDB
24.2.0
用户指南

文档版本 01

发布日期 2024-02-29

华为云计算技术有限公司

商标声明


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

注意

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

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

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 查看实例总览

操作场景

可展示已创建数据库实例相关信息，如实例状态统计、告警统计。

说明

该功能仅针对特定用户开放，如需配置白名单权限，您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交开通白名单的申请。

操作步骤

步骤1 **登录管理控制台**。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在左侧导航栏中单击“总览”，显示“总览”页面。

说明

没有数据库实例时，总览页面没有数据库实例相关信息，只展示“开始创建数据库实例”功能。

新用户可依据界面提示，创建实例。已有实例可展示实例相关信息。

表 1-1 实例总览参数解释

分类	参数名称	参数解释
实例状态统计	实例总数	显示当前云数据库GaussDB管理平台中所有创建的实例数。
	异常运行	显示“实例管理”中“运行状态”为“异常”的实例个数。 磁盘空间满的状态也属于“异常”。
	创建失败	显示“实例管理”中“运行状态”为“创建失败”的实例个数。

分类	参数名称	参数解释
	冻结	显示“实例管理”中“运行状态”为“冻结”的实例个数。
	创建中	显示“实例管理”中“运行状态”为“创建中”的实例个数。
	运行中	显示“实例管理”中“运行状态”为“正常”的实例个数。
告警统计 **说明** 支持选择查看近1个小时、近3个小时、近12个小时、近1天、近3天、近1周时间段内的告警统计。	级别统计	显示选定时间内所有未清除的告警统计。
	告警总数TOP实例	显示选定时间内未清除告警数量TOP5实例的告警统计。
紧急告警	告警名称/ID	显示该告警的名称与ID。
	告警实例名称/ID	显示该告警实例的名称与ID。
	告警级别	显示该告警的级别。 **须知** 紧急级别的故障影响到系统提供的服务，需要立即采取相应动作。如某设备或资源完全不可用，需立即采取措施，进行恢复。
	告警类型	显示该告警的类型，告警类型有“指标”或“事件”。
	首次告警时间	显示该告警首次上报的时间。
	告警规则更新时间	显示该告警的规则最近一次更新的时间。

----结束

2 使用规范建议

2.1 概述

简介

本规范以产品生命周期为主线，详细描述产品设计和开发流程过程中与数据库相关的设计和开发规范。

规范以提高可读性、代码质量为原则，强调实用性、可操作性，对GaussDB数据库开发容易产生问题的地方做出了明确的规定。主要包括下列内容：

设计规范包括：数据库设计、性能设计。

编程规范包括：排版、命名、注释、语法、脚本、数据库编程。

同时，在必要时，对部分规范给出细则及具体的示例。

术语约定

本规范采用以下的术语描述：

- **规格**：数据库规格，编程、设计必须遵守的原则，否则数据库将报错。
- **规则**：编程、设计必须考虑遵守的原则。
- **建议**：编程、设计推荐加以考虑的原则。
- **说明**：对规则或建议进行必要的解释。
- **示例**：对规则或建议进行或正或反方面的举例。
- **分片**：将一个表中的数据按照指定的策略拆分并存储至多个DN上的同名表中，这些表中存储的数据互不重叠，可以理解为“表的水平分库”，举例：t_user表按主键userId Hash拆分到GaussDB的4个不同DN中就是4个分片，每个分片内都有一张同名的t_user表。

适用范围

本规范适用于GaussDB 1.x及以上版本。

本规范适合人群包括：设计人员、开发人员、开发DBA、运维DBA、运维人员。

2.2 数据库设计规范

2.2.1 基本规范

数据库特性规范

表 2-1 数据库价值特性推荐

特性分类	特性列表	说明
表类型	HASH分布表	自动分片的表，建议数据量大的表使用（如交易记录）。
	REPLICATE分布表	不分片的普通表，建议数据少的表使用（如国家名称表）。
事务	分布式事务（弱一致）	GTM Free模式，在sharding场景下可保证强一致，不保证跨DN分片读一致性。建议完美sharding业务使用。
	分布式事务（强一致）	GTM Lite模式，保证跨DN读写一致性，建议非完美sharding业务使用。
扩容	在线平滑扩容	在线业务的最大阻塞小于5s，主要是为了追增切换期间写入日志，扩容速度100G/小时。
部署	3AZ3副本	建议默认采用3AZ3副本部署。
数据类型	整数类型	TINYINT, SMALLINT, INTEGER, BIGINT
	任意精度类型	NUMERIC/DEMICAL
	浮点类型	REAL/FLOAT4,DOUBLE PRECISION/ FLOAT8,FLOAT
	布尔类型	BOOLEAN
	定长字符	CHAR(n)
	变长字符	VARCHAR(n),NVARCHAR2(n), TEXT
	时间类型	DATE, TIME, TIMETZ, TIMESTAMP, TIMESTAMPTZ, SMALLDATETIME, INTERVAL, REALTIME
	二进制类型	BYTEA（变长二进制类型）
	位串类型	BIT(n), VARBIT(n)
函数	字符处理函数	字符类数据类型处理函数
	二进制字符串函数	二进制字符类型处理函数
	数字操作函数	数值类型处理函数

特性分类	特性列表	说明
	时间和日期处理函数	时间和日期类型处理函数
索引	主键/唯一索引	单列或多列主键/唯一索引
	BTREE索引	索引类型

说明

未包含在价值列表中的特性（包括但不限于自定义，UUID等数据类型，触发器等特性），如需使用建议联系GaussDB 数据库技术人员进行评估。

数据库指标规范

表 2-2 数据库指标规范列表

指标	推荐值
集群最佳分片数（主DN数）	<256
集群最佳长连接数	<10w
单DN数据量最大值	2TB
单表最佳字段个数	<50
单表最佳索引个数	<5
单表最佳复合索引个数	<3
单复合索引包含最佳列数	<5
单行最佳行宽	<2k
单个字段建议最大值	10MB
SQL语句最佳长度	<5k
磁盘可用空间不足提示（考虑扩容极端情况）	45%
磁盘可用空间不足告警（考虑扩容极端情况）	55%

2.2.2 部署规范

资源评估规范

基于表2-3模板估算台账，资源利用率尽量控制在40%~70%间，低40%建议缩容，高于70%建议扩容。

表 2-3 GaussDB 上线台账

组件	规格一	规格二	规格三	规格四	规格五	MEM/CPU	文件	磁盘容量	备注
CN	8C64G	16C128G	32C256G	ARM：60C480G X86：64C512G	/	8	/var/chroot/var/lib/log	64G	日志盘
							/var/chroot/usr/local	256G	数据盘
DN	8C64G	16C128G	32C256G	ARM：60C480G X86：64C512G	/	8	/var/chroot/var/lib/log	64G	日志盘
							/var/chroot/var/lib/engine/data	3788G	数据盘
							/var/chroot/var/lib/log/backup	20G	备份会把元数据信息放一份到这个盘上
GTM	4C32G	8C64G	16C128G	32C256G	ARM：60C480G X86：64C512G	8	/var/chroot/var/lib/log	32G	日志盘
	1-9分片	9-16分片	16-32分片	32-64分片	64分片以上		/var/chroot/usr/local	32G	数据盘
CMS+ETCD	4C32G	8C64G	16C128G	32C256G	ARM：60C480G X86：64C512G	8	/var/chroot/var/lib/log	32G	日志盘

组件	规格一	规格二	规格三	规格四	规格五	MEM/CPU	文件	磁盘容量	备注
	1-9分片	9-16分片	16-32分片	32-64分片	64分片以上		/var/chroot/usr/local	64G	数据盘

数据库参数规范

安装数据库后，DBA应根据环境特性，合理配置数据库GUC参数，当前GaussDB已提供默认参数模板，如果业务模型较为特殊的，需要联系技术支持进行调参。

2.2.3 数据库对象命名规范

- 数据库对象命名需要满足约束：长度不超过63个字符, 以字母或下划线开头， 中间字符可以是字母、数字、下划线、$、#。
- 不使用保留或非保留关键字命名数据库对象。

  说明

  也可以使用如下指令查看关键字：

  SELECT * **FROM** pg_get_keywords();

- 避免使用双引号括起来的字符串来定义数据库对象名称，除非必须限制数据库对象名称的大小写。

  说明

  GaussDB默认不区分SQL中对象名称的大小写，假如同一数据库中同时存在“t_Table”“t_table”两张不同的表，这种情况下应使用“”双引号，如果不存在该情况，禁止避免使用“”双引号。数据库对象名称大小写敏感会使定位问题难度增加。

- 数据库对象命名风格务必保持统一，建议使用小写。

  增量开发的业务系统或进行业务迁移的系统，建议遵守历史的命名风格。

  建议使用多个单词组成，以下划线分割。

  数据库对象名称建议能够望文知意，尽量避免使用自定义缩写（可以使用通用的术语缩写进行命名）。例如，在命名中可以使用具有实际业务含义的英文词汇或汉语拼音，但规则应该在集群范围内保持一致。

  变量名的关键是要具有描述性，即变量名称要有一定的意义，变量名要有前缀标明该变量的类型。

  表 2-4 数据库对象命名规则

对象类型	前缀	范例	长度约束(单位:字节)	备注
数据库名	db__	db_businessname	<=63	/

对象类型	前缀	范例	长度约束(单位:字节)	备注
普通表	t_	t_tablename	<=63	/
临时表	tmp_	tmp_tablename	<=63	例如：运营人员临时用作备份或临时进行数据采集用的中间表。 建议使用命名规则：tmp_表名缩写_创建人账号缩写_创建日期，例如：tmp_user_ytw_160505
主键	pk_	pk_tablename	<=63	如果表名过长，则用表名的缩写表示，尽量使用通用缩写或去元音的缩写方式。
唯一性约束	uk_	uk_tablename_columnname	<=63	唯一索引，用uk_表示。 如果表名或字段名过长，则用表名和字段名的缩写表示，尽量使用通用缩写或去元音的缩写方式。
函数	f_	f_functionname	<=63	/
表字段	/	/	<=63	字段命名建议使用实际含义的英文单词或简写。 例如表示bool类型的字段，命名规则：“is_”+描述。如member表上表示为enabled的会员的列命名为is_enabled；

2.2.4 数据库设计规范

- 使用JDBC客户端连接数据库时必须指明数据库名，具体格式为：
  ```
  jdbc:postgresql://host:port/database?param1=value1&param2=value2
  ```
- JDBC实例一旦创建，无法进行数据库切换。
- 数据库目前不支持不区分大小写的排序方式。
- 目前仅支持对数据库定义字符集，不支持对表、字段等其他对象定义字符集。
- 业务使用前必须先创建业务数据库。

  说明

  不应使用数据库安装后默认创建的postgres数据库存储业务数据。
- 创建数据库时必须指定字符集为UTF8，创建数据库时必须选择与客户端统一的编码字符集。

为了使用全球化需求，数据库编码应能够存储与标识绝大多数的字符，因此推荐使用UTF8。GaussDB中的UTF8字符集与MySQL的UTF8MB4等价，能够支持emoji表情字符。

如果客户端的编码方式与数据库的编码方式不统一，会带来转码性能，同时，针对同编码的内核优化无法触发，影响查询效率。

客户端的编码字符集需通过以下方式修改：

- 设置客户端连接参数，例如JDBC连接参数可通过在URL中追加characterEncoding和allowEncodingChanges参数。
  ```
  jdbc:postgresql://ip:port/database_name?characterEncoding=utf8&allowEncodingChanges=true
  ```
- 修改数据库GUC参数。
  ```
  SET client_encoding = 'UTF8';
  ```
- 数据库的编码在CREATE DATABASE时进行设置。
  ```
  CREATE DATABASE tester WITH ENCODING = 'UTF8';
  ```

- 数据库一旦创建无法更改字符集。
- 从便捷性和资源共享效率上考虑，建议使用SCHEMA进行业务隔离。

  说明

  GaussDB可以使用DATABASE和SCHEMA两种方式实现业务的隔离。

  区别在于DATABASE的隔离更加彻底，各个DATABASE之间共享资源极少，可实现连接隔离、权限隔离等。

  但DATABASE之间无法互相访问，JDBC建连时必须指明DATABASE，连接后无法切换DATABASE。

  SCHEMA隔离的方式共用资源较多，可以通过GRANT与REVOKE语法便捷地控制不同用户对各SCHEMA及其下属对象的权限。

- 创建数据库时建议指定LC_COLLATE和LC_CTYPE和存放的数据内容语言（中文\英文\等等）一致，该参数将影响数据的排序顺序。默认会用系统当前环境变量的默认设置。

  示例：

  ```
  CREATE DATABASE tester WITH ENCODING = 'UTF8' LC_COLLATE = 'en_US.UTF-8' LC_CTYPE = 'en_US.UTF-8';
  ```

  LC_COLLATE：用于明确字符排序规则。

  ```
  LC_COLLATE=C
  1
  2
  3
  A
  B
  C
  a  --注：小写在大写后面，按ASCII码排序
  b
  c

  en_US.UTF-8
  1
  2
  3
  a  --注：按字符排序
  A
  b
  B
  c
  C

  zh_CN.UTF-8
  1
  2
  ```

```
3
a
A
b
B
c
C
```

LC_CTYPE：用于判断哪些是字符is_alpha，是大写is_upper还是小写is_lower。

2.2.5 权限设计规范

- 业务使用前必须由root用户为业务创建DATABASE、SCHEMA和USER，然后再赋予相关用户对应对象的权限。

  说明

  如果该用户不是该schema的owner，要访问schema下的对象，需要同时给用户赋予schema的usage权限和对象的相应权限。

- DATABASE、SCHEMA和USER名使用小写。

  说明

  由于数据库默认会把对象名称转为小写，连接串里面如果出现大写的对象名无法连接到数据库。如create user MyUser;创建的用户，无法使用user=MyUser连接到数据库，需要用user=myuser才能连接到数据库。为了避免混淆，一律使用小写。

- 合理对角色和用户赋权，应使用最小化权限原则。

表 2-5 数据库对象权限及说明

对象	权限	说明
数据库 DATABASE	CONNECT	允许用户连接到指定的数据库。
	TEMP	-
	CREATE	允许在数据库里创建新的模式。
模式 SCHEMA	CREATE	允许在模式中创建新的对象。
	USAGE	允许访问包含在指定模式中的对象，若没有该权限，则只能看到这些对象的名字。
函数 FUNCTION	EXECUTE	允许使用指定的函数，以及利用这些函数实现的操作符。
表空间 TABLESPACE	CREATE	允许在表空间中创建表，允许在创建数据库和模式的时候把该表空间指定为缺省表空间。
表 TABLE	INSERT，DELETE UPDATE，SELECT	允许用户对指定表进行增删改查操作。
	TRUNCATE	允许执行TRUNCATE语句删除指定表中的所有记录。

对象	权限	说明
	REFERENCES	创建一个外键约束，必须拥有参考表和被参考表的REFERENCES权限。

- 通过角色而不是用户来管理权限。

  使用角色管理权限，即在角色中配置权限，再将角色赋予用户。

  通过角色管理权限，更便于多用户、用户变更等场景下的权限管理。例如：

  - 角色和用户为多对多关系，一个角色可以赋予多个用户，修改角色中的权限，被赋予角色的用户权限就可以同时更新。
  - 删除用户时，不会影响到角色。
  - 新建用户后可以通过赋予角色快速获取所需权限。
- 在删除指定数据库时，应回收用户对该数据库的CONNECT权限，避免删除时仍然存在活跃的数据库连接而失败。

2.2.6 表设计规范

- 必须指定表分布（DISTRIBUTE BY），表分布策略选择的原则如下：

  目前提供REPLICATION和HASH两种表分布策略。REPLICATION分布会在每个节点保留一份相同的完整的数据表。HASH分布会根据所提供的分布键值将表数据分布到多个节点中。

  - 对于系统配置表、数据字典表等数据规模小于2000w且插入更新十分低频的表，建议采用REPLICATION分布。

    须知

    慎用REPLICATION分布，该分布表会造成空间膨胀、DML性能下降等负面影响。

  - 对于数据量较大，更新频率较高的表，必须进行数据分片，要求采用HASH分布策略，分布键建议是主键中的一个或多个字段。
- 合理设计分布键，既要考虑查询开发的便利性，又要考虑数据的均匀存储，避免数据倾斜和读热点。

  Hash表的分布键选取至关重要，如果分布键选择不当，可能会导致数据倾斜，从而导致查询时，I/O负载集中在部分DN上，影响整体查询性能。因此，在确定Hash表的分布策略之后，需要对表数据进行倾斜性检查，以确保数据的均匀分布。

  a. 应使用取值较为离散的字段作为分布键，以便数据能够均匀分布到各个DN中。

  b. 在满足条件1情况下，存在常量过滤的字段不建议成为分布键，否则会使得所有的查询任务都会分发到唯一固定的DN上。

  c. 在满足条件1和2原则下，尽量选择查询中的关联条件作为分布键，这样可保证JOIN任务的相关数据分布在相同的DN上，减少DN间数据的流动代价。

说明

尽量避免数据shuffle。shuffle，是指在物理上，数据从一个节点，传输到另一个节点。shuffle占用了大量宝贵的网络资源，减小不必要的数据shuffle，可以减少网络压力，使数据的处理本地化，提高集群的性能和可支持的并发度。通过对关联条件和分组条件的仔细设计，能够尽可能的减少不必要的数据shuffle。

d. 由于数据库规格要求HASH分布表的主键必须包含其分布列，因此在选择分布列时，也可以考虑选择表的主键作为分布键。

表 2-6 常见的分布键及效果

分布键值	分布键分布均匀性
用户 ID，应用程序中有许多用户。	好
状态代码，只有几个可用的状态代码。	差
项目创建日期，四舍五入至最近的时间段（例如，天、小时或分钟）。	差
设备 ID，每个设备以相对类似的间隔访问数据.	好

- 分布键使用的列长度不易超过128，过长会带来较高的计算开销。
- 分布键值一旦插入不允许更新（UPDATE），如需更新需删除后插入。
- 视图不允许嵌套。

  一方面，如果视图编写时使用了通配符，当被调用的视图新增或删除列时，视图将发生错误。

  另一方面，视图嵌套可能因无法使用索引而执行效率低下，尽量使用带有索引的基表而不是视图上做关联操作。
- 分布键不建议超过3列，列数过多将带来较高的计算开销。
- 视图定义中尽量避免排序操作。

  ORDER BY子句在顶层视图上无效，如果必须对输出数据排序，请考虑在调用视图中使用ORDER BY。

2.2.7 字段设计规范

- 字段设计应使用推荐类型。

  字段设计需使用推荐字段，如果需要使用禁用、不推荐的字段类型，建议联系技术支持进行评估。

  这些数据类型不推荐或禁止的原因是业务使用场景较少，未大规模商用。

  对于业务上有迫切字段类型要求的，联系技术支持，提交需求。

表 2-7 数据库数据类型最佳实践

数据类型	说明	是否推荐
UUID	不同集群可能产生相同UUID	禁止，建议业务直接采用中间件平台提供的分布式ID

数据类型	说明	是否推荐
序列整型	即自增列，包括SMALLSERIAL,SERIAL,BIGSERIAL	禁止
整数类型	TINYINT, SMALLINT, INTEGER, BIGINT	推荐
任意精度类型	NUMERIC/DEMICAL	推荐
浮点类型	REAL/FLOAT4,DOUBLE PRECISION/FLOAT8,FLOAT	推荐
布尔类型	BOOLEAN	推荐
定长字符	CHAR(n)	推荐
变长字符	VARCHAR(n),NVARCHAR2(n) VARCHAR/TEXT	推荐
时间类型	DATE, TIME, TIMESTAMP, SMALLDATETIME, INTERVAL, REALTIME	推荐
	TIMETZ，TIMESTAMPTZ	不推荐
二进制类型	BYTEA（变长二进制类型）	推荐
	CLOB（字符大对象）,BLOB（二进制大对象），RAW（变长十六进制）	禁止
位串类型	BIT(n), VARBIT(n)	推荐
特殊字符类型	NAME,"CHAR"，通常供数据库系统内部使用	禁止
JSON类型	JSON类型目前不支持操作符	禁止
自定义类型	可用于定义枚举EMU等类型	禁止
HLL数据类型	建议直接使用HLL相关函数，减少性能影响	禁止
货币类型	MONEY 存储带有固定小数精度的货币金额	禁止
几何类型	POINT, LSEG, BOX, PATH, POLYGON, CIRCLE	禁止
网络地址类型	存储IPV4 IPV6 MAC地址数据类型	禁止
文本搜索类型	用于支持全文检索	禁止

- 合理选用字符串数据类型。优先使用变长字符类VARCHAR。只有该字段输入确定为固定字符则使用定长字符类型，或需要自动补充空格，才使用CHAR(n)。

说明

典型的定长字段类型，例如“sex”字段，仅允许输入“f”或“m”一个字节长度的字符。这类字段建议使用定长数据类型（如CHAR(n)）。

如果不存在此特点，或者后续可能扩展需要输入更长的字符，请优先使用变长字符类型（如VARCHAR, TEXT），且**不建议指定变长类型的长度**。

原因如下：

- 定长字段会对不够长度的输入数据补充空格，然后存入数据库中，产生不必要的存储空间浪费。
- 如果定义为定长字符类型，后续扩展长度，需要对全表进行扫描重写，性能开销大，影响在线业务。

对于指定固定长度的变长字段，每次插入时会检查是否长度越界，带来性能开销。

- 字符类型字段不应存储数字类型的数据。

  如果对存储在字符类型字段中的数据进行数值计算，或者与数值进行比较操作（如置于过滤条件中），会带来不必要的数据类型转换的开销，同时该字段上的索引可能失效，影响查询性能。

- 字符类型字段不应存储时间或日期类数据。

  如果对存储在字符类型字段中的数据与日期类数据进行计算或比较操作（如置于过滤条件中），会带来不必要的数据类型转换的开销，同时该字段上的索引可能失效，影响查询性能。

- 对于明确不存在NULL值的字段加上NOT NULL约束。

  对于NOT NULL字段，优化器在某些场景下会进行特殊优化，可较大提升查询性能。

- 相关联字段的数据类型应保持一致。

  在进行关联操作时，如果字段类型不一致，会带来数据类型转换开销。

- 大字段（例如varchar（1000）、varchar(4000)）不建议超过8个。
- 字段定义时建议同时创建COMMENT注释信息，以便于未来维护。

  不同类型字段说明、取值范围及使用方法请参考**数据类型**章节。

- 用于WHERE条件过滤和关联的字段都应设置NOT NULL约束。

  对于NOT NULL字段，优化器在某些场景下会进行特殊优化，可较大提升查询性能。

- 不建议对表预留字段。大部分场景下可支持快速新增、删除表字段，或者修改字段的DEFAULT值。

  说明

  新增列必须符合以下要求，否则会带来全表更新开销，影响在线业务。

  - 数据类型为以下类型中的一种：BOOL, BYTEA, SMALLINT, BIGINT, SMALLINT, INTEGER, NUMERIC, FLOAT, DOUBLE PRECISION, CHAR, VARCHAR, TEXT, TIMESTAMPTZ, TIMESTAMP, DATE, TIME, TIMETZ, INTERVAL；
  - 新增列的DEFAULT值长度不超过128个字节；
  - 新增列DEFAULT值不包含volatile函数；
  - 新增列设置有DEFAULT值，且DEFAULT值不为NULL。

  如果不确定是否满足条件，请联系数据库技术人员进行评估。

- 尽量使用高效的数值类数据类型。在满足业务精度的情况下，选择的优先级从高到低依次为整数、浮点数、NUMERIC。

- 合理设置数值字段的数据类型，根据取值范围选择合适的数值类型，尽量少用NUMERIC/DECIMAL类型。

  NUMERIC和DECIMAL等价，NUMERIC(或DECIMAL）数据类型操作对CPU消耗较高。

表 2-8 数值类数据类型存储空间及取值范围

类型	存储空间/Byte	最小值	最大值
TINYINT	1	0	255
SMALLINT	2	-32768	32767
INTEGER	4	-2,147,483,648	2,147,483,647
BIGINT	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
REAL/FLOAT4	4	6位十进制数字精度	
DOUBLE PRECISION/FLOAT8	8	15位十进制数字精度	

2.2.8 索引设计规范

- 使用数据库索引实践推荐的索引类型。

  索引设计建议使用推荐类型，如果需要使用禁用、不推荐、限制使用的索引类型，建议联系GaussDB数据库专家进行评估。

表 2-9 数据库索引实践推荐

索引类型	说明	是否推荐
主键/唯一索引	单列或多列主键/唯一索引	推荐
表达式索引	索引列为表的一列或多列计算而来的一个函数或者标量表达式	限制使用

- 对于HASH分布表，主键和唯一索引必须包含分布键。
- 当设计组合索引时，优化器会通过计算代价来选择合适的执行计划。例如：在组合索引(a,b,c)下，当查询时如果只使用过滤条件b，优化器经过计算索引成本较低时，会选择索引。
- 不建议单表创建多个唯一索引。

  同时维护多个唯一索引的开销远大于维护一个多列唯一索引，如果业务逻辑上多个唯一索引，与一个多列唯一索引等价，应使用多列唯一索引。
- 组合索引字段个数不超过5个。
- 禁止组合索引组合字符串的总长度超过200。
- 索引（包括单列索引和复合索引）字段应为NOT NULL字段。

- 同字段上创建索引的维护效率不同。数值类型字段优于字符类型及其他数据类型，因此对于考虑创建索引的ID、时间等字段，建议使用数值类型进行存储。
- 建议在关联列上创建索引。

  GaussDB支持HASH JOIN，但是当内表较小等RESCAN代价较低的情况下，仍然可能选择NESTLOOP JOIN来完成关联。如果通过EXPLAIN可以查看到NESTLOOP JOIN计划，则可以通过在关联列上创建索引，提高NESTLOOP JOIN效率。

2.2.9 函数/存储过程设计规范

- 避免使用存储过程、触发器等实现业务逻辑，应该将这些逻辑都放到业务服务器上处理，避免对数据库产生逻辑依赖。
- 业务的数据库升级脚本中，禁止使用存储过程实现升级逻辑。
- 仅创建对固定入参有固定返回值的函数，函数必须设为IMMUTABLE和SHIPPABLE类型。

  目前数据库支持三种类型的函数，分别是IMMUTABLE, STABLE, VOLATILE。

  对于IMMUTABLE函数且设置为SHIPPABLE的函数，会允许函数在DN上执行。在大部分场景下，该函数的执行效率较高。

  但是此类函数要求对于固定的入参得到固定的返回值，来保证函数在DN上执行的正确性。如果函数的结果依赖对数据表的扫描结果（例如获取某个表中列的max值）或依赖时间（如获取当前时间），那么函数应设置为STABLE或者VOLATILE，且NOT SHIPPABLE，以保证函数执行的正确性。在此种场景下，所有DN上的数据将发送至某一个CN上进行计算，导致查询执行效率低下。

2.3 数据库编程规范

2.3.1 GUC 参数编程规范

客户端（如JDBC）应使用默认（全局）参数执行查询，禁用会话级别的GUC参数。

通过ODBC或JDBC修改GUC参数时，需注意GUC参数仅会在当前connection中生效，特别是在连接池场景下，容易发生问题，且导致问题定位困难。

如果在连接中必须进行GUC参数设置，那么在将连接归还给连接池之前，必须使用

SET SESSION AUTHORIZATION DEFAULT;

RESET ALL;

将连接的状态清空。

2.3.2 对象访问编程规范

访问对象（表，函数等）时建议带上SCHEMA名称，即使用schemaname.tablename进行访问。

如果不追加SCHEMA名称前缀，会根据当前search_path中表空间列表，依次搜索所有表空间直到找到匹配的表作为目标表，带来不必要的性能开销。

2.3.3 WHERE

- 表查询时，WHERE条件中应包含所有分布键字段等值查询条件，否则将在多个节点上进行查询，影响系统并发度和性能。

- 禁止在WHERE条件相同表字段进行相互比较。

  例如如下语句应考虑合理性：

  SELECT * **FROM** t1 **WHERE** col1 = col1;

  应考虑修改为：

  SELECT * **FROM** t1 **WHERE** col1 IS NOT NULL;

- 禁止WHERE条件涉及隐式数据类型转换。

  数据库中进行隐式转换后可能导致无法使用所创建的索引，导致潜在的性能问题。

  强烈建议在开发过程中开启GUC参数check_implicit_conversions，并关闭enable_fast_query_shipping，以便检查询语句中是否存在可能带来不良性能影响的隐式数据类型。

  SET enable_fast_query_shipping = off;

  SET check_implicit_conversions = true;

  由于隐式数据类型转换检测存在额外的开销，一旦查询语句开发完成后，请关闭check_implicit_conversions参数，并重置enable_fast_query_shipping。

  示例：

  如下代码不符合规范：

  t_tablename表的phonenumber字段为VARCHAR类型（而不是数值类型），以下语句利用phonenumber进行条件过滤时，优化器会将phonenumber隐式转化为bigint类型。

  SELECT column1

  INTO i_l_variable1

  FROM t_tablename

  WHERE phonenumber = 13512345678;

  导致两个后果：

  a. 不能进行DN裁剪，计划下发到所有的DN上执行。

  b. 计划中不能使用index scan方式扫描数据。

  建议修改t_tablename表的phonenumber字段为VARCHAR类型（而不是数值类型）。

  SELECT column1

  INTO i_l_variable1

  FROM t_tablename

  WHERE phonenumber = '13512345678';

- 禁止WHERE 条件字段使用表达式或是函数。

  对条件字段使用表达式或函数时，索引会失效，同时会对每一行数据进行计算，产生不必要的性能消耗。主要因为非常量的表达式在预处理阶段不能转化为Const值，因此不能用来剪枝，导致查询语句扫描所有的数据。

  示例：

  如下代码不符合规范：

  SELECT income **FROM** table **WHERE** abs(income) > ?;

  SELECT income **FROM** table **WHERE** income * 10 > ?;

  SELECT create_time

  FROM table

WHERE date_format(create_time, '%Y%m%d %H:%i:%s') = '20090101 00:00:0';

应修改为：

SELECT income **FROM** table **WHERE** income > ? **OR** income < (-1) * ?;

SELECT income **FROM** table **WHERE** income > ?/10;

SELECT create_time

FROM table

WHERE create_time = str_to_date('20090101 00:00:0', '%Y%m%d %H:%i:%s');

- 查询条件中与NULL做比较时，禁止使用“！=”比较符，应使用IS NULL或IS NOT NULL。

  不能写expression=NULL或expression ！ = NULL，因为NULL代表一个未知的值，不能通过表达式判断两个未知值是否相等。
- 查询条件中禁止对索引字段使用“!= ”比较符，避免索引失效。
- where子句中的过滤条件，尽量符合单边规则。即把字段名放在比较条件的一边，优化器在某些场景下会自动进行剪枝优化。形如col op expression，其中col为表的一个列， op为‘ =’、‘ >’的等比较操作符， expression为不含列名的表达式。

  示例：

  如下代码不推荐使用，根据time列进行筛选

  SELECT id, from_image_id, from_person_id, from_video_id FROM face_data WHERE current_timestamp(6) - time < '1 days'::interval;

  建议修改为：

  SELECT id, from_image_id, from_person_id, from_video_id FROM face_data WHERE time > current_timestamp(6) - '1 days'::interval;
- 查询条件的索引字段上避免与NULL（IS NULL和IS NOT NULL）进行比较。
- 查询条件的索引字段上避免使用NOT。
- 查询条件的索引字段上避免使用NOT IN。
- 模糊查询LIKE语句，非必要情况下，%不应放在首字符位置。

  如果%放在首字符位置，将无法使用索引，会导致全表扫描。
- WHERE条件中IN的候选子集不易过大，建议不超过500。

  说明

  查询时，会对IN中每一条数据进行等值比较，开销较大。

  如果包含的值为较为固定的值，应考虑创建REPLICATION表，并将候选数据写入表中，然后通过INNER JOIN来实现包含查询。
- WHERE条件中IN的候选子集不为常量，而是表中的列时，建议改写为子查询。

说明

在这种情况下，实际上是一个不等值的JOIN，会通过nestloop计划执行。在表过大时执行效率低下，建议修改为等值JOIN的子查询。

示例

如下代码不推荐使用：

```
SELECT col1, COALESCE(max(col2 - 1), 0)
FROM t1, t2
WHERE t1.col1 = ANY(VALUES(id1), (id2))
GROUP BY col1;
```

建议修改为：

```
SELECT col1, COALESCE(max(tmp), 0) FROM
(
(
SELECT col1, (col2-1) AS tmp
FROM t1, t2
WHERE t1.col1 = t2.id1 AND t1.col1 != t2.id2
) UNION ALL (
SELECT col1, (col2-1) AS tmp
FROM t1, t2
WHERE t1.col1 = t2.id2
)
) GROUP BY col1;
```

- 多使用等值操作，少使用非等值操作。

  WHERE条件中的非等值条件（IN、BETWEEN、<、<=、>、>=）会导致后面的条件使用不了索引，因为不能同时用到两个范围条件。

2.3.4 SELECT

- SELECT语句中慎用通配符字段“*”。

  使用通配符字段查询表时，如果因业务或数据库升级导致表结构发生变化，可能出现与业务语句不兼容的情况。

  因此业务应指明所需查询的表字段名称，避免使用通配符。

- 带有LIMIT的查询语句中必须带有ORDER BY保证有序。

  说明

  GaussDB是一种分布式数据库，表数据将分布在多个DN上。

  如果SQL语句中只带有LIMIT，而不带有ORDER BY子句，数据库将会把网络传输较快的DN所发送的（符合查询要求的）结果作为最终结果输出到客户端。

  由于网络传输效率不同时刻可能发生改变，因此导致多次执行该SQL语句时，返回结果表现出不一致的情况。

- 避免对大字段（如VARCHAR(2000)）执行ORDER BY、DISTINCT、GROUP BY、UNION等会引起排序的操作。

  此类操作将消耗大量的CPU和内存资源，执行效率低下。

- 禁止使用LOCK TABLE语句加锁，仅允许使用 SELECT .. FOR UPDATE语句。

  LOCK TABLE提供多种锁级别，但如果对数据库原理和业务理解不足，误用表锁可能触发死锁，导致集群不可用。

- 避免在SELECT目标列中使用子查询，可能导致计划无法下推到DN执行，影响执行性能。
- 考虑使用UNION ALL，少使用UNION，注意考虑去重。

  UNION ALL不去重，少了排序操作，速度相对UNION更快。

  如果没有去重的需求，优先使用UNION ALL。
- 需要统计表中所有记录数时，不要使用count(col)来替代count(*)。count(*)会统计NULL值（真实行数），而count(col)不会统计。
- 在执行count(col)时，将“值为NULL”的记录行计数为0。在执行sum(col)时，当所有记录都为NULL时，最终将返回NULL；当不全为NULL时，“值为NULL”的记录行将被计数为0。
- count(多个字段)时，多个字段名必须用圆括号括起来。例如，count((col1,col2,col3))。注意：通过多字段统计行数时，即使所选字段都为NULL，该行也被计数，效果与count(*)一致。
- count(distinct col)用来计算该列不重复的非NULL的数量， NULL将不被计数。
- count(distinct (col1,col2,...))用来统计多列的唯一值数量，当所有统计字段都为NULL时，也会被计数，同时这些记录被认为是相同的。
- 使用连接操作符“ ||”替换concat函数进行字符串连接。因为concat函数生成的执行计划不能下推，导致查询性能严重劣化。
- 当in(val1, val2, val3…)表达式中字段较多时，建议使用in (values(va11), (val2), (val3)…)语句进行替换。优化器会自动把in约束转换为非关联子查询，从而提升查询性能。
- 避免频繁使用下使用count()获取大表行数，该操作资源消耗较大，影响并行作业执行效率。

  如果不需要实时的行数统计信息，可以尝试使用如下语句来获取表行数。

  SELECT reltuples **FROM** pg_class **WHERE** relname = 'tablename';

须知

pg_class中所记录的表行数信息只会在对该表执行ANALYZE以后才会更新。

目前ANALYZE有两种触发条件:

- 业务主动发送ANALYZE语句，例如：

  --分析连接库中所有表

  ANALYZE;

  --分析指定表

  ANALYZE tablename;
- 借助AUTO VACUUM机制，在每间隔一定时间或表的增删达到一定行数时触发。间隔时间和增删比例可通过GUC参数设置。

2.3.5 INSERT

- INSERT ON DUPLICATE KEY UPDATE不支持对主键或唯一约束的列上执行UPDATE。

  INSERT ON DUPLICATE KEY UPDATE的语义是对唯一约束冲突的行进行更新，这个过程中不应对约束的值进行更新。

- INSERT ON DUPLICATE KEY UPDATE如果插入多条数据，这些数据之间不允许存在主键/唯一约束冲突。

  与MySQL行为存在一定差异，MYSQL允许此行为，例如如下语句：

  INSERT INTO t1 VALUES(1, 1), (1, 2) **ON DUPLICATE KEY UPDATE** col2 = VALUES(col2);

  不符合SQL标准，SQL标准不允许在同一条SQL COMMAND中，对插入行同时进行修改，使得结果无可预期。例如上例中，假设col1为主键，那么此时会在同一个COMMAND中，既插入一条主键为1的，同时对已插入进行修改。由于事务串行化无法确认哪一个操作先执行，因此可能导致结果不稳定。

  如果插入的数据本身存在冲突，在分布式场景下更新应以哪一条记录为准难以确定。

- 禁止对存在多个唯一约束的表执行INSERT ON DUPLICATE KEY UPDATE。

  说明

  表中存在多个唯一约束包括存在多个唯一索引，或既存在主键（PRIMARY KEY），又存在唯一索引(UNQUE INDEX)两种情况。

  当存在多个唯一约束时，会默认检查所有的唯一约束条件，只要任何一个约束存在冲突，就会对冲突行进行更新，即可能更新多条记录，与业务预期不相符。业务应给予更加明确的插入更新条件。

- 对于批量插入的情况，建议使用INSERT INTO TABLE1 VALUES (),(),()，执行效率将高于执行多条INSERT INTO VALUES()。

  说明

  由于目前无法识别多个值是否属于一个shard，所以要使用/*+ multinode */ 来允许此语句执行。

  无论单条和多条插入，关键字均需是VALUES。

  不支持MySQL的INSERT INTO mytable VALUE()的用法。

2.3.6 UPDATE

- 不支持UPDATE语句中直接使用LIMIT，应使用WHERE条件明确需要更新的目标行。
- 在GTM-FREE模式下，不允许跨节点事务，因此更新HASH分布中数据表时WHERE条件中必须指定分布列等值过滤条件。
- 不支持多表更新。

  多表更新即在单条SQL语句中，对多个表进行更新。
- UPDATE语句中必须有WHERE子句，避免全表扫描。
- 不允许在UPDATE子句同时更新多个列时，被更新列同样是更新源。

  同时更新多列，且更新源相同，在不同的数据库下行为不同，为了避免带来兼容性问题，业务层应避免上述操作。

  示例：

  UPDATE table **SET** col1 = col2, col3 = col1 **WHERE** col1 = 1;

  该语句在GaussDB中，col3的值为原col1的值；而MySQL中，col3的值为col2的值（因为col2的值被赋予给了col1）。
- UPDATE语句中禁止使用ORDER BY、GROUP BY子句，避免不必要的排序。
- 有主键/索引的表，更新时WHERE条件应结合主键/索引。

2.3.7 DELETE

- 不支持DELETE语句中使用LIMIT。应使用WHERE条件明确需要更新的目标行。
- 在GMT-FREE模式下，不允许跨节点事务，因此删除HASH分布表中数据时，必须在WHERE条件中指定分布列等值过滤条件。
- 不支持多表删除。

  多表删除即在单条SQL语句中，对多个表进行删除。
- DELETE语句中必须有WHERE子句，避免全表扫描。
- DELETE语句中禁止不应使用ORDER BY、GROUP BY子句，避免不必要的排序。
- 如果需要清空一张表，建议使用TRUNCATE，而不是DELETE。

  TRUNCATE会创建新的物理文件，并在事务结束时将原文件物理删除，清空磁盘空间。而DELETE会将表中数据进行标记，直到VACUUM FULL阶段才会真正清理磁盘空间。
- DELETE有主键或索引的表，WHERE条件应结合主键或索引，提高执行效率。

2.3.8 关联查询

- 多表关联嵌套深度必须小于8。

  关联嵌套过深，容易产生慢sql，应从业务层考虑优化。
- 表关联查询必须明确指定各表的连接条件（ ON ），以避免产生笛卡尔积。

  例如在MySQL中，JOIN与CROSS JOIN和INNER JOIN等价，但是在SQL标准中，JOIN仅与INNER JOIN等价，必须配合使用ON连接条件。

  反例：如下2个示例会造成笛卡尔积查询

  SELECT * FROM TABLE1 A , TABLE2 B;

  SELECT * FROM TABLE1 A JOIN TABLE2 B ON 1=1;
- 关联时，应该根据SQL标准指明连接方式，避免直接使用JOIN关键词，而是使用CROSS JOIN, INNER JOIN, LEFT JOIN, RIGHT JOIN等。
- 多表关联查询时, 必须应对表添加使用别名，保证语句逻辑清晰，便于维护。
- 不同字段的比较开销不同，关联字段应尽量使用比较效率高的字段类型。

  数值类型的比较效率远高于字符串类型。

  在数值类型中，整型效率高于NUMERIC和浮点类型。
- 关联字段应为相同数据类型，避免存在隐式类型转换影响执行效率。
- 少用嵌套子查询，尽量使用表关联，因为子查询会产生临时表，对SQL性能影响较大。
- 对于关联列上存在大量NULL值的情况，建议在WHERE条件中增加关联列IS NOT NULL的过滤条件，能够提升执行效率。

2.3.9 子查询

- 禁止一条SQL语句中，出现重复子查询语句。
- 少用标量子查询。

  标量子查询指结果为1个值，并且条件表达式为等值的子查询。

  示例：不符合规范的语句

  SELECT * FROM t1 WHERE id = (SELECT id FROM t2 LIMIT 1);

上述语句建议业务拆分为两条SQL语句，先执行子查询。

- 避免在SELECT目标列中使用子查询，可能导致计划无法下推影响执行性能。
- 子查询嵌套深度建议不超过2层。

  由于子查询会带来临时表开销，过于复杂的查询应考虑从业务逻辑上进行优化。

2.3.10 事务

- 在GTM-FREE模式下，不允许执行跨节点事务。

  在GTM-FREE模式下，如果所执行的SQL语句包含跨节点事务，会报错处理。

  a. 如果语句拆分多条会报错：

  ```
  INSERT/UPDATE/DELETE/MERGE contains multiple remote queries under GTM-free modeUnsupport DML two phase commit under gtm free mode. modify your SQL to generate light-proxy or fast-query-shipping plan。
  ```

  此时需要修改语句，来单节点执行。

  b. 如果语句涉及多节点会报错：

  ```
  Your SQL needs more than one datanode to be involved in.
  ```

  建议对语句进行修改，使得能够单节点执行。如果需要此种语句多节点执行，需要添加一个hint来允许，例如：**insert /*+ multinode */ into t values(3,3),(1,1);**

  建议开发阶段在jdbc连接串内设置application_type=perfect_sharding_type，这样所有跨节点读写操作的SQL都会报错，用来提示开发人员尽早优化语句。

- 大对象操作不支持事务。

  说明

  大对象操作包括：创建删除DATABASE, ANALYZE, VACUUM。

- 通过JDBC接入数据库时，避免拼接多条SQL为一条语句发送执行。

  当多条语句拼接为一条语句，且其中包含对象操作时，如果中间对象操作失败，会重新开启新事务执行后续语句。

  示例：不符合规则语句

  ```
  Connection conn = ....
  try {
     Statement stmt = null;
     try {
        stmt = conn.createStatement();
        stmt.executeUpdate("CREATE TABLE t1 (a int); DROP TABLE t1");
     } finally {
        stmt.close();
     }
     conn.commit();
  } catch(Exception e) {
    conn.rollback();
  } finally {
    conn.close();
  }
  ```

  上述执行语句，如果“**CREATE TABLE t1;**”失败，会重新开启新事务执行“**DROP TABLE t1;**”导致执行失败。应拆分成两条语句分别发送:

  ```
  Connection conn = ....
  try {
     Statement stmt = null;
     try {
        stmt = conn.createStatement();
        stmt.executeUpdate("CREATE TABLE t1 (a int)");
        stmt.executeUpdate("DROP TABLE t1");
  ```

```
    } finally {
        stmt.close();
    }
    conn.commit();
} catch(Exception e) {
    conn.rollback();
} finally {
    conn.close();
}
```

2.4 客户端编程规范

2.4.1 JDBC

- JDBC实例必须指定数据库，一旦实例创建，无法切换数据库。
- 单条SQL语句的长度不允许超过2G字节，业务应考虑通信成本，建议单条SQL语句不超过5K行。
- 不支持对DDL使用Prepare Execute执行方式。
- fetchsize必须要在autocommit关闭情况下使用，否则fetchsize配置无效。
- 使用默认GUC参数，避免通过JDBC发送SET请求修改GUC参数。

  说明

  更多说明请参考**GUC参数编程规范**。
- 必须使用Prepare Execute方式执行查询语句，提高执行效率。
- JDBC客户端所在主机时区、数据库集群所在主机时区和集群配置过程中的时区，三者应保持一致。
- 如果在连接中创建了临时表，那么在将连接归还给连接池之前，必须将临时表删除，避免业务出错。
- 合理设置prepareThreshold，如果query语句十分固定，建议设置为1。
- 建议设置连接参数autobalance=true，开启CN负载均衡功能，并中设置多个CN连接地址（使用逗号分隔）。

  一旦开启autobalance，JDBC DRIVER会尝试将JDBC connection分配到不同的CN节点上。

  设置多个CN连接地址的目的是避免JDBC DRIVER在首次获取集群CN列表时，因所设置的CN节点故障而失败。

  一旦首次成功获取，便不会再依赖连接参数中指定的CN列表，而是根据实时获取的集群CN列表，每隔一段时间，选取连接其中有效的CN获取最新的CN列表。
- 应根据业务上层请求超时时间合理设置JDBC连接超时时间，避免作业完成或常超作业持续占用数据库资源

说明

超时参数包括loginTimeout、connectTimeout、socketTimeout等。

- loginTimeout：Integer类型。指建立数据库连接的等待时间。超时时间单位为秒。
- connectTimeout：Integer类型。用于连接CN操作的超时值。如果连接到CN花费的时间超过此值，则连接断开。超时时间单位为秒，默认值为0，表示已禁用，timeout不发生。
- socketTimeout：Integer类型。用于socket读取操作的超时值。如果从CN读取所花费的时间超过此值，则连接关闭。超时时间单位为秒，默认值为0，表示已禁用，timeout不发生。
- cancelSignalTimeout：Integer类型。发送取消消息本身可能会阻塞，此属性控制用于取消命令的“connect超时”和“socket超时”。超时时间单位为秒，默认值为10秒。
- tcpKeepAlive：Boolean类型。启用或禁用TCP保活探测功能。默认为false。

以上参数可以在JDBC连接串或者property连接属性中配置，例如：

1. 在连接串中配置：
   ```
   jdbc:postgresql://host:port/postgres?tcpKeepAlive=true
   ```
2. 在property中配置：
   ```
   Properties info = new Properties();
   Info.setProperty("tcpKeepAlive", true);
   ```

2.5 参数配置规范

2.5.1 GaussDB 参数配置标准

数据库提供了许多运行参数，配置这些参数可以影响数据库系统的行为。在修改这些参数时请确保用户理解了这些参数对数据库的影响，否则可能会导致无法预料的结果。

可修改的参数和参数说明请在实例信息页面的“参数修改”页签查看，参数修改方式请参考**修改实例参数**。

3 性能调优

3.1 总体调优思路

GaussDB的总体性能调优思路为性能瓶颈点分析、关键参数调整以及SQL调优。在调优过程中，通过系统资源、吞吐量、负载等因素来帮助定位和分析性能问题，使系统性能达到可接受的范围。

GaussDB性能调优过程需要综合考虑多方面因素，因此，调优人员应对系统软件架构、软硬件配置、数据库配置参数、并发控制、查询处理和数据库应用有广泛而深刻的理解。

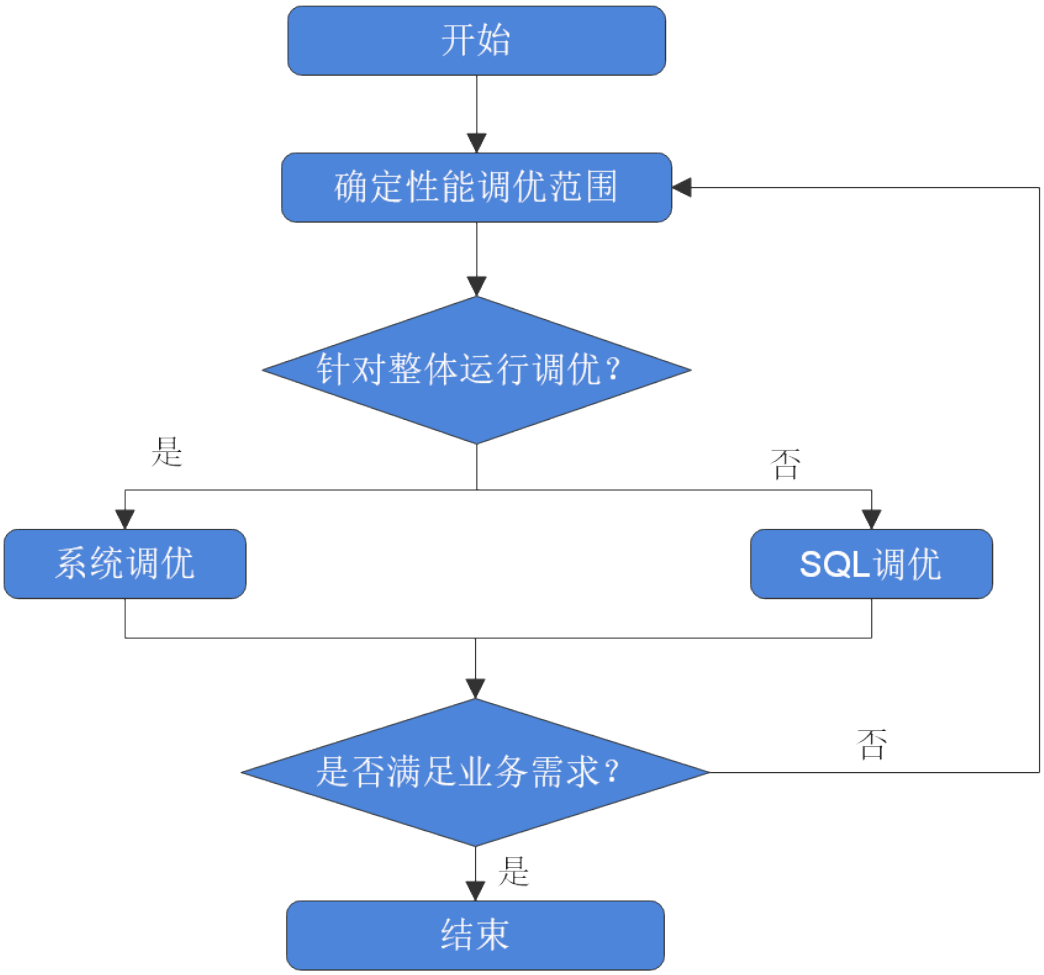
须知

性能调优过程有时候需要重启集群，可能会中断当前业务。因此，业务上线后，当性能调优操作需要重启集群时，操作窗口时间需向管理部门提出申请，经批准后方可执行。

调优流程

调优流程如图3-1所示。

图 3-1 GaussDB 性能调优流程



3.2 确定性能调优范围

数据库性能调优通常发生在用户对业务的执行效率不满意，期望通过调优加快业务执行的情况下。正如“**确定性能调优范围**”小节所述，数据库性能受影响因素多，从而性能调优是一项复杂的工程，有些时候无法系统性地说明和解释，而是依赖于DBA的经验判断。尽管如此，此处还是期望能尽量系统性的对性能调优方法加以说明，方便应用开发人员和刚接触GaussDB的DBA参考。

性能因素

多个性能因素会影响数据库性能，了解这些因素可以帮助定位和分析性能问题。

- 系统资源

  数据库性能在很大程度上依赖于磁盘的I/O和内存使用情况。为了准确设置性能指标，用户需要了解集群部署硬件的基本性能。CPU，硬盘，磁盘控制器，内存和网络接口等这些硬件性能将显著影响数据库的运行速度。

- 负载

  负载等于数据库系统的需求总量，它会随着时间变化。总体负载包含用户查询，应用程序，并行作业，事务以及数据库随时传递的系统命令。比如：多用户在执

行多个查询时会提高负载。负载会显著地影响数据库的性能。了解工作负载高峰期可以帮助用户更合理地利用系统资源，更有效地完成系统任务。

- 吞吐量

  使用系统的吞吐量来定义处理数据的整体能力。数据库的吞吐量以每秒的查询次数、每秒的处理事务数量或平均响应时间来测量。数据库的处理能力与底层系统（磁盘I/O，CPU速度，存储器带宽等）有密切的关系，所以当设置数据库吞吐量目标时，需要提前了解硬件的性能。

- 竞争

  竞争是指两组或多组负载组件尝试使用冲突的方式使用系统的情况。比如，多条查询视图同一时间更新相同的数据，或者多个大量的负载争夺系统资源。随着竞争的增加，吞吐量下降。

- 优化

  数据库优化可以影响到整个系统的性能。在执行SQL制定、数据库配置参数、表设计、数据分布等操作时，启用数据库查询优化器打造最有效的执行计划。

调优范围确定

性能调优主要通过查看集群各节点的CPU、内存、I/O和网络这些硬件资源的使用情况，确认这些资源是否已被充分利用，是否存在瓶颈点，然后针对性调优。

- 如果某个资源已达瓶颈，则通过查询最耗时的SQL语句、跑不出来的SQL语句，找出耗资源的SQL，进行**SQL调优指南**。
- 如果所有资源均未达瓶颈，则表明性能仍有提升潜力。可以查询最耗时的SQL语句，或者跑不出来的SQL语句，进行针对性的**SQL调优指南**。

3.2.1 查询最耗性能的 SQL

系统中有些SQL语句运行了很长时间还没有结束，这些语句会消耗很多的系统性能，请根据本章内容查询长时间运行的SQL语句。

操作步骤

步骤1 使用DAS或者gsql连接实例。

步骤2 查询系统中长时间运行的查询语句。

```
SELECT current_timestamp - query_start AS runtime, datname, usename, query FROM pg_stat_activity
where state != 'idle' ORDER BY 1 desc;
```

查询后会按执行时间从长到短顺序返回查询语句列表，第一条结果就是当前系统中执行时间最长的查询语句。返回结果中包含了系统调用的SQL语句和用户执行SQL语句，请根据实际找到用户执行时间长的语句。

若当前系统较为繁忙，可以通过限制current_timestamp - query_start大于某一阈值来查看执行时间超过此阈值的查询语句。

```
SELECT query FROM pg_stat_activity WHERE current_timestamp - query_start > interval '1 days';
```

步骤3 设置参数track_activities为on。

```
SET track_activities = on;
```

当此参数为on时，数据库系统才会收集当前活动查询的运行信息。

步骤4 查看正在运行的查询语句。

以查看视图pg_stat_activity为例：


```
SELECT datname, usename, state FROM pg_stat_activity;
 datname  | usename | state  |
----------+---------+--------+
 postgres |   omm   | idle   |
 postgres |   omm   | active |
(2 rows)
```

如果state字段显示为idle，则表明此连接处于空闲，等待用户输入命令。

如果仅需要查看非空闲的查询语句，则使用如下命令查看：

```
SELECT datname, usename, state FROM pg_stat_activity WHERE state != 'idle';
```

步骤5 分析长时间运行的查询语句状态。

- 若查询语句处于正常状态，则等待其执行完毕。
- 若查询语句阻塞，则通过如下命令查看当前处于阻塞状态的查询语句：
  ```
  SELECT datname, usename, state, query FROM pg_stat_activity WHERE waiting = true;
  ```
  查询结果中包含了当前被阻塞的查询语句，该查询语句所请求的锁资源可能被其他会话持有，正在等待持有会话释放锁资源。

只有当查询阻塞在系统内部锁资源时，waiting字段才显示为true。尽管等待锁资源是数据库系统最常见的阻塞行为，但是在某些场景下查询也会阻塞在等待其他系统资源上，例如写文件、定时器等。但是这种情况的查询阻塞，不会在视图pg_stat_activity中体现。

----结束

3.2.2 分析作业是否被阻塞

数据库系统运行时，在某些业务场景下查询语句会被阻塞，导致语句运行时间过长，可以强制结束有问题的会话。

操作步骤

步骤1 使用DAS或者gsql连接实例。

步骤2 查看阻塞的查询语句及阻塞查询的表、模式信息。

```
SELECT w.query as waiting_query,
w.pid as w_pid,
w.usename as w_user,
l.query as locking_query,
l.pid as l_pid,
l.usename as l_user,
t.schemaname || '.' || t.relname as tablename
from pg_stat_activity w join pg_locks l1 on w.pid = l1.pid
and not l1.granted join pg_locks l2 on l1.relation = l2.relation
and l2.granted join pg_stat_activity l on l2.pid = l.pid join pg_stat_user_tables t on l1.relation = t.relid
where w.waiting;
```

该查询返回线程ID、用户信息、查询状态，以及导致阻塞的表、模式信息。

步骤3 使用如下命令结束相应的会话。

```
SELECT PG_TERMINATE_BACKEND(139834762094352);
```

其中，139834762094352为线程ID。

显示类似如下信息，表示结束会话成功。

```
 PG_TERMINATE_BACKEND
----------------------
 t
(1 row)
```


显示类似如下信息，表示用户正在尝试结束当前会话。

```
FATAL:  terminating connection due to administrator command
FATAL:  terminating connection due to administrator command
```

说明

- gsql客户端使用PG_TERMINATE_BACKEND函数结束当前正在执行会话的后台线程时，如果当前的用户是初始用户，客户端不会退出而是自动重连，即还会返回“The connection to the server was lost. Attempting reset: Succeeded.”；否则客户端会重连失败，即返回“The connection to the server was lost. Attempting reset: Failed.”。这是因为只有初始用户可以免密登录，普遍用户不能免密登录，从而重连失败。
- 对于使用PG_TERMINATE_BACKEND函数结束非活跃的后台线程时。如果打开了线程池，此时空闲的会话没有线程ID，无法结束会话。非线程池模式下，结束的会话不会自动重连。

----结束

3.2.3 参数调优建议

数据库参数是数据库系统运行的关键配置信息，设置不合适的参数值可能会影响业务。本文列举了一些重要参数说明，更多参数详细说明，请参考**导出参数**，将参数导出后查看。

通过控制台界面修改参数值，请参见**修改实例参数**。

查询

- **track_stmt_session_slot**

  作用：设置一个session缓存的最大的全量/慢SQL的数量。

  影响：缓存的SQL定期会被写入到系统表，如果业务量很大，超过这个数量语句执行将不会被跟踪，直到落盘线程将缓存语句落盘，留出空闲的空间，但不影响SQL的执行。

- **effective_cache_size**

  作用：设置节点优化器在一次单一的查询中可用的磁盘缓冲区的有效大小。设置这个参数，还要考虑的共享缓冲区以及内核的磁盘缓冲区。另外，还要考虑预计的在不同表之间的并发查询数目，因为它们将共享可用的空间。这个参数对分配的共享内存大小没有影响，它也不会使用内核磁盘缓冲，它只用于估算。数值是用磁盘页来计算的，通常每个页面是8192字节。

  取值范围：整型，1～INT_MAX，单位为8KB。

  影响：比默认值高的数值可能会导致使用索引扫描，更低的数值可能会导致选择顺序扫描。

- **enable_stream_operator**

  控制优化器对stream的使用。当该参数关闭时，可能会有大量关于计划不能下推的日志记录到日志文件中。

- **log_min_duration_statement**

  作用：当某条语句的持续时间大于或者等于特定的毫秒数时，记录每条完成语句的持续时间。设置log_min_duration_statement可以很方便地跟踪需要优化的查询语句。对于使用扩展查询协议的客户端，语法分析、绑定、执行每一步所花时间被独立记录。

  影响：设置过低的阈值可能影响负载吞吐，-1表示关闭此功能。

审计参数

- **audit_system_object**

  作用：该参数决定是否对数据库对象的CREATE、DROP、ALTER操作进行审计。数据库对象包括DATABASE、USER、SCHEMA、TABLE等。通过修改该配置参数的值，可以只审计需要的数据库对象的操作，在主备强制选主场景建议audit_system_object取最大值，所有DDL对象全部审计。

  影响：不当修改该参数会导致丢失DDL审计日志，请在客服人员指导下进行修改。

锁管理

- **update_lockwait_timeout**

  设置并发更新同一行数据时单个锁的最长等待时间，当申请的锁等待时间超过设定值时系统会报错。0表示不会超时，默认值为2min。

连接与认证

- **session_timeout**

  表明与服务器建立连接后，不进行任何操作一定时间后超时的限制，0表示关闭超时设置。

- **failed_login_attempts**

  设置密码错误次数上限，输入密码错误的次数达到该参数所设置的值时，账户将会被自动锁定，配置为0时表示不限制密码输入错误的次数。

- **password_effect_time**

  设置账户密码的有效时间，0表示不开启有效期限制功能。

- **password_lock_time**

  设置账户被锁定后的自动解锁时间，单位为天。

3.3 SQL 调优指南

SQL调优的唯一目的是“资源利用最大化”，即CPU、内存、磁盘IO、网络IO四种资源利用最大化。所有调优手段都是围绕资源使用开展的。所谓资源利用最大化是指SQL语句尽量高效，节省资源开销，以最小的代价实现最大的效益。比如做典型点查询的时候，可以用seqscan+filter(即读取每一条元组和点查询条件进行匹配)实现，也可以通过indexscan实现，显然indexscan可以以更小的代价实现相同的效果。根据硬件资源和客户的业务特征确定合理的集群部署方案和表定义是数据库在多数情况下满足性能要求的基础。

3.3.1 Query 执行流程

SQL引擎从接受SQL语句到执行SQL语句需要经历的步骤如图3-2和表3-1所示。其中，红色字体部分为DBA可以介入实施调优的环节。

图 3-2 SQL 引擎执行查询类 SQL 语句的流程

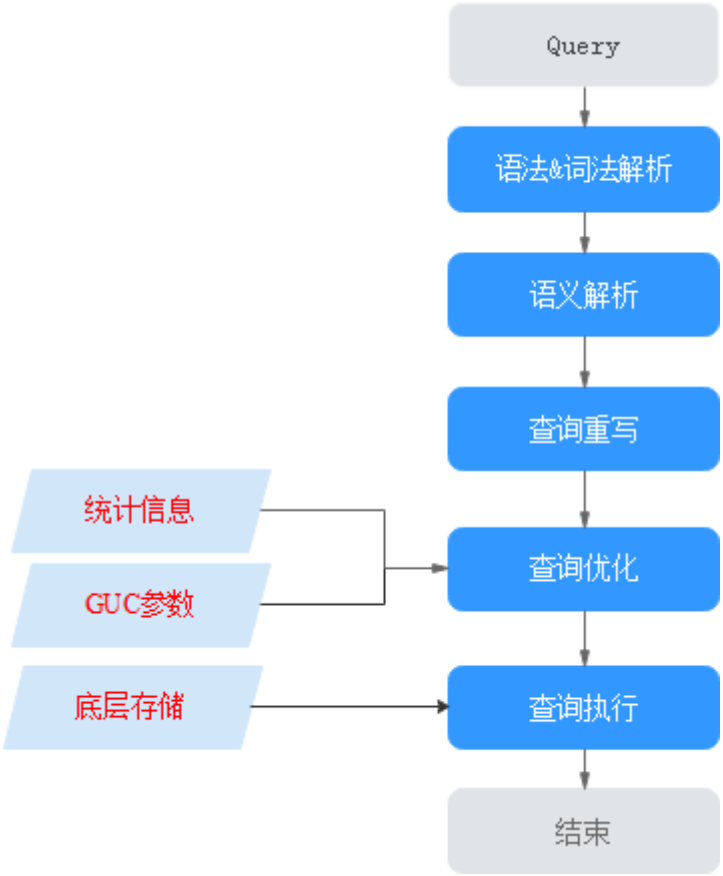


表 3-1 SQL 引擎执行查询类 SQL 语句的步骤说明

步骤	说明
语法&词法解析	按照约定的SQL语句规则，把输入的SQL语句从字符串转化为格式化结构(Stmt)。
语义解析	将“语法&词法解析”输出的格式化结构转化为数据库可以识别的对象。
查询重写	根据规则把“语义解析”的输出等价转化为执行上更为优化的结构。
查询优化	根据“查询重写”的输出和数据库内部的统计信息规划SQL语句具体的执行方式，也就是执行计划。统计信息和GUC参数对查询优化（执行计划）的影响，请参见**调优手段之统计信息**和**调优手段之GUC参数**。
查询执行	根据“查询优化”规划的执行路径执行SQL查询语句。底层存储方式的选择合理性，将影响查询执行效率。详见**调优手段之底层存储**。

调优手段之统计信息

GaussDB优化器是典型的基于代价的优化 (Cost-Based Optimization，简称CBO)。在这种优化器模型下，数据库根据表的元组数、字段宽度、NULL记录比率、distinct

值、MCV值、HB值等表的特征值，以及一定的代价计算模型，计算出每一个执行步骤的不同执行方式的输出元组数和执行代价(cost)，进而选出整体执行代价最小/首元组返回代价最小的执行方式进行执行。这些特征值就是统计信息。从上面描述可以看出统计信息是查询优化的核心输入，准确的统计信息将帮助规划器选择最合适的查询规划，一般来说通过analyze语法收集整个表或者表的若干个字段的统计信息，周期性地运行ANALYZE，或者在对表的大部分内容做了更改之后马上运行它是个好习惯。

调优手段之 GUC 参数

查询优化的主要目的是为查询语句选择高效的执行方式。

如下SQL语句:

```
select count(1)
from customer inner join store_sales on (ss_customer_sk = c_customer_sk);
```

在执行customer inner join store_sales的时候，GaussDB支持Nested Loop、Merge Join和Hash Join三种不同的Join方式。优化器会根据表customer和表store_sales的统计信息估算结果集的大小以及每种join方式的执行代价，然后对比选出执行代价最小的执行计划。

正如前面所说，执行代价计算都是基于一定的模型和统计信息进行估算，当因为某些原因代价估算不能反映真实的cost的时候，就需要通过GUC参数设置的方式让执行计划倾向更优规划。

调优手段之底层存储

GaussDB的表支持行存表、列存表，底层存储方式的选择严格依赖于客户的具体业务场景。一般来说计算型业务查询场景(以关联、聚合操作为主)建议使用列存表；点查询、大批量UPDATE/DELETE业务场景适合行存表。

对于每种存储方式还有对应的存储层优化手段，这部分会在后续的调优章节深入介绍。

调优手段之 SQL 重写

除了上述干预SQL引擎所生成执行计划的执行性能外，根据数据库的SQL执行机制以及大量的实践发现，有些场景下，在保证客户业务SQL逻辑的前提下，通过一定规则由DBA重写SQL语句，可以大幅度的提升SQL语句的性能。

这种调优场景对DBA的要求比较高，需要对客户业务有足够的了解，同时也需要扎实的SQL语句基本功，后续会介绍几个常见的SQL改写场景。

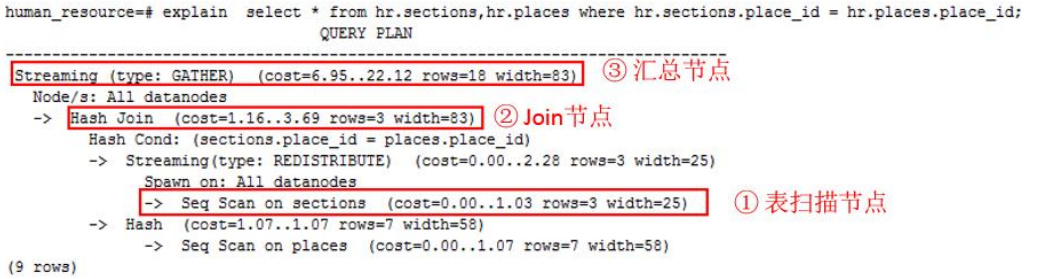
3.3.2 SQL 执行计划介绍

3.3.2.1 SQL 执行计划概述

SQL执行计划是一个节点树，显示GaussDB执行一条SQL语句时执行的详细步骤。每一个步骤为一个数据库运算符。

使用EXPLAIN命令可以查看优化器为每个查询生成的具体执行计划。EXPLAIN给每个执行节点都输出一行，显示基本的节点类型和优化器为执行这个节点预计的开销值。如**图3-3**所示。

图 3-3 SQL 执行计划示例



- 最底层节点是表扫描节点，它扫描表并返回原始数据行。不同的表访问模式有不同的扫描节点类型：顺序扫描、索引扫描等。最底层节点的扫描对象也可能是非表行数据（不是直接从表中读取的数据），如VALUES子句和返回行集的函数，它们有自己的扫描节点类型。
- 如果查询需要连接、聚集、排序、或者对原始行做其它操作，那么就会在扫描节点之上添加其它节点。 并且这些操作通常都有多种方法，因此在这些位置也有可能出现不同的执行节点类型。
- 第一行(最上层节点)是执行计划总执行开销的预计。这个数值就是优化器试图最小化的数值。

执行计划显示格式

GaussDB对执行计划提供了normal、pretty、summary、run四种显示格式：

- normal：代表使用默认的打印格式。图3-3中即为此显示格式。
- pretty：代表使用GaussDB改进后的新显示格式。新的格式层次清晰，计划包含了plan node id，性能分析简单直接。

```
gaussdb=#  explain select * from t1,t2 where t1.c1=t2.c2;
 id |              operation               | E-rows | E-width | E-costs
----+--------------------------------------+--------+---------+---------
  1 | ->  Streaming (type: GATHER)         |     20 |     180 | 29.07
  2 |    ->  Hash Join (3,5)               |     20 |     180 | 27.75
  3 |      ->  Streaming(type: REDISTRIBUTE) |     20 |     104 | 14.31
  4 |        ->  Seq Scan on t2            |     20 |     104 | 13.13
  5 |      ->  Hash                        |     21 |      76 | 13.13
  6 |        ->  Seq Scan on t1            |     20 |      76 | 13.13
(6 rows)
```

- summary：是在pretty的基础上增加了对打印信息的分析。
- run：在summary的基础上，将统计的信息输出到csv格式的文件中，以便于进一步分析。

通过设置GUC参数explain_perf_mode，可以显示不同格式的执行计划。下文的用例默认显示pretty格式。

执行计划显示信息

除了设置不同的执行计划显示格式外，还可以通过不同的EXPLAIN用法，显示不同详细程度的执行计划信息。常见有如下几种，关于更多用法请参见**EXPLAIN**语法说明。

- EXPLAIN *statement*： 只生成执行计划，不实际执行。其中statement代表SQL语句。
- EXPLAIN ANALYZE *statement*：生成执行计划，进行执行，并显示执行的概要信息。显示中加入了实际的运行时间统计，包括在每个规划节点内部花掉的总时间(以毫秒计)和它实际返回的行数。

- EXPLAIN PERFORMANCE *statement*：生成执行计划，进行执行，并显示执行期间的全部信息。

EXPLAIN为了测量运行时在执行计划中每个节点的开销，EXPLAIN ANALYZE或EXPLAIN PERFORMANCE会在当前查询执行上增加性能分析的开销。在一个查询上运行EXPLAIN ANALYZE或EXPLAIN PERFORMANCE有时会比普通查询明显的花费更多的时间。超支的数量依赖于查询的本质和使用的平台。详情参见当前版本的《云数据库 GaussDB 开发指南（1.x版本）》的“SQL参考 > SQL语法 > EXPLAIN”章节。

因此，当定位SQL运行慢问题时，如果SQL长时间运行未结束，建议通过EXPLAIN命令查看执行计划，进行初步定位。如果SQL可以运行出来，则推荐使用EXPLAIN ANALYZE或EXPLAIN PERFORMANCE查看执行计划及其实际的运行信息，以便更准确地定位问题原因。

EXPLAIN PERFORMANCE轻量化执行方式与EXPLAIN PERFORMANCE保持一致，在原来的基础上减少了性能分析的时间，执行时间与SQL执行时间的差异显著减少。

3.3.2.2 详解

如**SQL执行计划概述**节中所说，EXPLAIN会显示执行计划，但并不会实际执行SQL语句。EXPLAIN ANALYZE和EXPLAIN PERFORMANCE两者都会实际执行SQL语句并返回执行信息。在这一节将详细解释执行计划及执行信息。

执行计划

以如下SQL语句为例：

```
select d, avg(a::numeric(7, 2)) from t_distinct group by d;
```

执行EXPLAIN的输出为：

```
gaussdb=#  explain select d, avg(a::numeric(7, 2)) from t_distinct group by d;
 id |                    operation                    | E-rows | E-width | E-costs
----+-------------------------------------------------+--------+---------+---------
  1 | ->  Row Adapter                                 |     20 |      40 | 14.52
  2 |    ->  Vector Streaming (type: GATHER)          |     20 |      40 | 14.52
  3 |       ->  Vector Hash Aggregate                 |     20 |      40 | 13.59
  4 |          ->  Vector Streaming(type: REDISTRIBUTE) |     20 |       8 | 13.33
  5 |             ->  CStore Scan on t_distinct       |     20 |       8 | 13.01
(5 rows)
```

执行计划字段解读（横向）：

- id：执行算子节点编号。
- operation：具体的执行节点算子名称。

  Vector前缀的算子是指向量化执行引擎算子，一般出现含有列存表的Query中。

  Streaming是一个特殊的算子，它实现了分布式架构的核心数据shuffle功能，Streaming共有三种形态，分别对应了分布式结构下不同的数据shuffle功能：
  - Streaming (type: GATHER)：作用是coordinator从DN收集数据。
  - Streaming(type: REDISTRIBUTE)：作用是DN根据选定的列把数据重分布到所有的DN。
  - Streaming(type: BROADCAST)：作用是把当前DN的数据广播给其他所有的DN。
- E-rows：每个算子估算的输出行数。

- E-memory：DN上每个算子估算的内存使用量，只有DN上执行的算子会显示。某些场景会在估算的内存使用量后使用括号显示该算子在内存资源充足下可以自动扩展的内存上限。
- E-width：每个算子输出元组的估算宽度。
- E-costs：每个算子估算的执行代价。
  - E-costs是优化器根据成本参数定义的单位来衡量的，习惯上以磁盘页面抓取为1个单位， 其它开销参数将参照它来设置。
  - 每个节点的开销（ E-costs值 ）包括它的所有子节点的开销。
  - 开销只反映了优化器关心的东西，并没有把结果行传递给客户端的时间考虑进去。虽然这个时间可能在实际的总时间里占据相当重要的分量，但是被优化器忽略了，因为它无法通过修改规划来改变。

执行计划层级解读（纵向）：

1. 第一层：CStore Scan on t1

   表扫描算子，用CStore Scan的方式扫描表t1。这一层的作用是把表t1的数据从buffer或者磁盘上读上来输送给上层节点参与计算。
2. 第二层：Vector Hash Aggregate

   聚合算子，作用是把下层计算输送上来的算子做聚合操作(group by)。
3. 第三层：Vector Streaming (type: GATHER)

   Shuffle算子，此处GATHER类型的Shuffle算子作用是把数据从DN汇聚到CN。
4. 第四层：Row Adapter

   存储格式转化算子，主要作用是把内存中列式格式数据转为行式数据，以便客户端展示。

需要注意的是最顶层算子为Data Node Scan时，需要设置enable_fast_query_shipping为off才能看到具体的执行计划，如下面这个计划：

```
gaussdb=#  explain select c1,count(1) from t1 group by c1;
                  QUERY PLAN
-----------------------------------------------------
 Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
   Node/s: All datanodes
(2 rows)
```

设置enable_fast_query_shipping参数之后，执行计划显示如下：

```
gaussdb=#  set enable_fast_query_shipping=off;
SET
gaussdb=#  explain select c1,count(1) from t1 group by c1;
 id |          operation           | E-rows | E-width | E-costs
----+------------------------------+--------+---------+---------
  1 | ->  Streaming (type: GATHER) |     20 |      12 | 14.23
  2 |    ->  HashAggregate         |     20 |      12 | 13.30
  3 |       ->  Seq Scan on t1     |     20 |       4 | 13.13
(3 rows)
```

执行计划中的关键字说明：

1. 表访问方式
   - Seq Scan

     全表顺序扫描。
   - Index Scan

     优化器决定使用两步的规划：最底层的规划节点访问一个索引，找出匹配索引条件的行的位置，然后上层规划节点真实地从表中抓取出那些行。独立地

抓取数据行比顺序地读取它们的开销高很多，但是因为并非所有表的页面都被访问了，这么做实际上仍然比一次顺序扫描开销要少。使用两层规划的原因是，上层规划节点在读取索引标识出来的行位置之前，会先将它们按照物理位置排序，这样可以最小化独立抓取的开销。

如果在WHERE里面使用的好几个字段上都有索引，那么优化器可能会使用索引的AND或OR的组合。但是这么做要求访问两个索引，因此与只使用一个索引，而把另外一个条件只当作过滤器相比，这个方法未必是更优。

索引扫描可以分为以下几类，不同类型之间的差异在于索引的排序机制。

- Bitmap Index Scan

  使用位图索引抓取数据页。

- Index Scan using index_name

  使用简单索引搜索，该方式表的数据行是以索引顺序抓取的，这样就令读取它们的开销更大，但是这里的行少得可怜，因此对行位置的额外排序并不值得。最常见的就是看到这种规划类型只抓取一行，以及那些要求ORDER BY条件匹配索引顺序的查询。因为那时候没有多余的排序步骤是必要的以满足ORDER BY。

2. 表连接方式
   - Nested Loop

     嵌套循环，适用于被连接的数据子集较小的查询。在嵌套循环中，外表驱动内表，外表返回的每一行都要在内表中检索找到它匹配的行，因此整个查询返回的结果集不能太大（不能大于10000），要把返回子集较小的表作为外表，而且在内表的连接字段上建议要有索引。
   - (Sonic) Hash Join

     哈希连接，适用于数据量大的表的连接方式。优化器使用两个表中较小的表，利用连接键在内存中建立hash表，然后扫描较大的表并探测散列，找到与散列匹配的行。Sonic和非Sonic的Hash Join的区别在于所使用hash表结构不同，不影响执行的结果集。
   - Merge Join

     归并连接，通常情况下执行性能差于哈希连接。如果源数据已经被排序过，在执行融合连接时，并不需要再排序，此时融合连接的性能优于哈希连接。
3. 运算符
   - sort

     对结果集进行排序。
   - filter

     EXPLAIN输出显示WHERE子句当作一个"filter"条件附属于顺序扫描计划节点。这意味着规划节点为它扫描的每一行检查该条件，并且只输出符合条件的行。预计的输出行数降低了，因为有WHERE子句。不过，扫描仍将必须访问所有 10000 行，因此开销没有降低；实际上它还增加了一些（确切的说，通过10000 * cpu_operator_cost）以反映检查WHERE条件的额外CPU时间。
   - LIMIT

     LIMIT限定了执行结果的输出记录数。如果增加了LIMIT，那么不是所有的行都会被检索到。

执行信息

在SQL调优过程中经常需要执行EXPLAIN ANALYZE或EXPLAIN PERFORMANCE查看SQL语句实际执行信息，通过对比实际执行与优化器的估算之间的差别来为优化提供

依据。EXPLAIN PERFORMANCE相对于EXPLAIN ANALYZE增加了每个DN上的执行信息。

以如下SQL语句为例：

select count(1) from t1;

执行EXPLAIN PERFORMANCE输出为：

```
gaussdb=# explain performance select count(1) from t1;
 id |            operation             |     A-time      | A-rows | E-rows | E-distinct | Peak Memory  | E-memory | A-
width | E-width | E-costs
----+----------------------------------+-----------------+--------+--------+------------+--------------+----------
+---------+---------+---------
  1 | ->  Aggregate                    | 9.326           |      1 |      1 |            | 14KB         |          |        |       8 |
209.10
  2 |    ->  Streaming (type: GATHER)  | 9.281           |      2 |      2 |            | 80KB         |          |        |       8
| 209.10
  3 |       ->  Aggregate              | [5.981,6.491] |      2 |      2 |            | [13KB, 13KB] | 1MB      |        |
8 | 209.01
  4 |          ->  Seq Scan on public.t1 | [2.553,2.909] |  20000 |  20000 |            | [15KB, 15KB] | 1MB
|         |       0 | 184.00
(4 rows)

         Memory Information (identified by plan id)
-------------------------------------------------------------------
 Coordinator Query Peak Memory:
        Query Peak Memory: 0MB
 DataNode Query Peak Memory
        datanode1 Query Peak Memory: 2MB
        datanode2 Query Peak Memory: 0MB
   1 --Aggregate
        Peak Memory: 14KB, Estimate Memory: 64MB
   2 --Streaming (type: GATHER)
        Peak Memory: 80KB, Estimate Memory: 64MB
   3 --Aggregate
        datanode1 Peak Memory: 13KB, Estimate Memory: 1024KB
        datanode2 Peak Memory: 13KB, Estimate Memory: 1024KB
   4 --Seq Scan on public.t1
        datanode1 Peak Memory: 15KB, Estimate Memory: 1024KB
        datanode2 Peak Memory: 15KB, Estimate Memory: 1024KB
(15 rows)

 Targetlist Information (identified by plan id)
-------------------------------------------------
   1 --Aggregate
        Output: count((count(1)))
   2 --Streaming (type: GATHER)
        Output: (count(1))
        Node/s: All datanodes
   3 --Aggregate
        Output: count(1)
   4 --Seq Scan on public.t1
        Output: c1, c2, c3, c4, c5
        Distribute Key: c1
(10 rows)

                                  Datanode Information (identified by plan id)
-----------------------------------------------------------------------------------------------------------------------------------
   1 --Aggregate
        (actual time=9.326..9.326 rows=1 loops=1)
        (Buffers: 0)
        (CPU: ex c/r=-17813058098842432, ex row=2, ex cyc=-35626116197684864, inc
cyc=71252232399791904)
   2 --Streaming (type: GATHER)
        (actual time=8.628..9.281 rows=2 loops=1)
        (Buffers: 0)
        (CPU: ex c/r=53439174298738384, ex row=2, ex cyc=106878348597476768, inc
cyc=106878348597476768)
```

```
  3 --Aggregate
        datanode1 (actual time=5.980..5.981 rows=1 loops=1)
        datanode2 (actual time=6.491..6.491 rows=1 loops=1)
        datanode1 (Buffers: shared hit=85)
        datanode2 (Buffers: shared hit=84)
        datanode1 (CPU: ex c/r=-35622581151734248, ex row=10078, ex cyc=-359004372847177760768, inc
cyc=71252232395610160)
        datanode2 (CPU: ex c/r=-35622525572390744, ex row=9922, ex cyc=-353446698729260974080, inc
cyc=71252232398542704)
  4 --Seq Scan on public.t1
        datanode1 (actual time=0.018..2.553 rows=10078 loops=1)
        datanode2 (actual time=0.017..2.909 rows=9922 loops=1)
        datanode1 (Buffers: shared hit=85)
        datanode2 (Buffers: shared hit=84)
        datanode1 (CPU: ex c/r=35629651228376004, ex row=10078, ex cyc=359075625079573381120, inc
cyc=359075625079573381120)
        datanode2 (CPU: ex c/r=35629706809278324, ex row=9922, ex cyc=353517950961659543552, inc
cyc=353517950961659543552)
(22 rows)

                        User Define Profiling
---------------------------------------------------------------------------
 Plan Node id: 2  Track name: coordinator get datanode connection
       coordinator1: (time=0.019 total_calls=1 loops=1)
 Plan Node id: 2  Track name: Coordinator serialize plan
       coordinator1: (time=1.059 total_calls=1 loops=1)
 Plan Node id: 2  Track name: Coordinator send begin command
       coordinator1: (time=0.003 total_calls=1 loops=1)
 Plan Node id: 2  Track name: Coordinator start transaction and send query
       coordinator1: (time=0.045 total_calls=1 loops=1)
(8 rows)

                     ====== Query Summary =====
---------------------------------------------------------------------------
 Datanode executor start time [datanode1, datanode2]: [0.421 ms,0.450 ms]
 Datanode executor run time [datanode1, datanode2]: [6.002 ms,6.528 ms]
 Datanode executor end time [datanode2, datanode1]: [0.027 ms,0.028 ms]
 Remote query poll time: 0.000 ms, Deserialze time: 0.000 ms
 System available mem: 8222310KB
 Query Max mem: 8310784KB
 Query estimated mem: 2048KB
 Coordinator executor start time: 0.181 ms
 Coordinator executor run time: 9.340 ms
 Coordinator executor end time: 0.052 ms
 Planner runtime: 0.421 ms
 Plan size: 3122 byte
 Query Id: 72339069014648468
 Total runtime: 9.657 ms
(14 rows)
```

图中显示执行信息分为以下7个部分

1. 以表格的形式将计划显示出来，包含有11个字段，分别是：id、operation、A-time、A-rows、E-rows、E-distinct、Peak Memory、E-memory、A-width、E-width和E-costs。其中计划类字段（id、operation以及E开头字段）的含义与执行EXPLAIN时的含义一致，详见**执行计划**小节中的说明。A-time、A-rows、E-distinct、Peak Memory、A-width的含义说明如下：
   - A-time：当前算子执行完成时间，一般DN上执行的算子的A-time是由[]括起来的两个值，分别表示此算子在所有DN上完成的最短时间和最长时间。
   - A-rows：表示当前算子的实际输出元组数。
   - E-distinct：表示hashjoin算子的distinct估计值。
   - Peak Memory：此算子在每个DN上执行时使用的内存峰值。
   - A-width：表示当前算子每行元组的实际宽度，仅对于重内存使用算子会显示，包括：(Vec)HashJoin、(Vec)HashAgg、(Vec) HashSetOp、

(Vec)Sort、(Vec)Materialize算子等，其中(Vec)HashJoin计算的宽度是其右子树算子的宽度，会显示在其右子树上。

2. Predicate Information (identified by plan id)：

   这一部分主要显示的是静态信息，即在整个计划执行过程中不会变的信息，主要是一些join条件和一些filter信息。

3. Memory Information (identified by plan id)：

   这一部分显示的是整个计划中会将内存的使用情况打印出来的算子的内存使用信息，主要是Hash、Sort算子，包括算子峰值内存（peak memory），控制内存（control memory），估算内存使用（operator memory），执行时实际宽度（width），内存使用自动扩展次数（auto spread num），是否提前下盘（early spilled），以及下盘信息，包括重复下盘次数（spill Time(s)），内外表下盘分区数（inner/outer partition spill num），下盘文件数（temp file num），下盘数据量及最小和最大分区的下盘数据量（written disk IO [min, max] ）。

4. Targetlist Information (identified by plan id)：

   这一部分显示的是每一个算子输出的目标列。

5. DataNode Information (identified by plan id)：

   这一部分会将各个算子的执行时间、CPU、buffer的使用情况全部打印出来。

6. User Define Profiling：

   这一部分显示的是CN和DN、DN和DN建连的时间，以及存储层的一些执行信息。

7. ====== Query Summary =====：

   这一部分主要打印总的执行时间和网络流量，包括了各个DN上初始化和结束阶段的最大最小执行时间、CN上的初始化、执行、结束阶段的时间，以及当前语句执行时系统可用内存、语句估算内存等信息。

须知

- A-rows和E-rows的差异体现了优化器估算和实际执行的偏差度。一般来说，偏差越大优化器生成的计划越不可信，人工干预调优的必要性越大。
- A-time中的两个值偏差越大，表明此算子的计算偏斜(在不同DN上执行时间差异)越大，人工干预调优的必要性越大。
- Max Query Peak Memory经常用来估算SQL语句耗费内存，也被用来作为SQL语句调优时运行态内存参数设置的重要依据。一般会以EXPLAIN ANALYZE或EXPLAIN PERFORMANCE的输出作为进一步调优的输入。

3.3.3 调优流程

对慢SQL语句进行分析，通常包括以下步骤：

操作步骤

步骤1 收集SQL中涉及到的所有表的统计信息。在数据库中，统计信息是优化器生成计划的源数据。没有收集统计信息或者统计信息陈旧往往会造成执行计划严重劣化，从而导致性能问题。从经验数据来看，10%左右性能问题是因为没有收集统计信息。具体请参见**更新统计信息**。

步骤2 通过查看执行计划来查找原因。如果SQL长时间运行未结束，通过EXPLAIN命令查看执行计划，进行初步定位。如果SQL可以运行出来，则推荐使用EXPLAIN ANALYZE或

EXPLAIN PERFORMANCE查看执行计划及实际运行情况，以便更准确地定位问题原因。有关执行计划的详细介绍请参见**SQL执行计划介绍**。

步骤3 **审视和修改表定义**。

步骤4 针对EXPLAIN或EXPLAIN PERFORMANCE信息，定位SQL慢的具体原因以及改进措施，具体参见**典型SQL调优点**。

步骤5 通常情况下，有些SQL语句可以通过查询重写转换成等价的，或特定场景下等价的语句。重写后的语句比原语句更简单，且可以简化某些执行步骤达到提升性能的目的。查询重写方法在各个数据库中基本是通用的。**经验总结：SQL语句改写规则**介绍了几种常用的通过改写SQL进行调优的方法。

----结束

3.3.4 更新统计信息

在数据库中，统计信息是优化器生成计划的源数据。没有收集统计信息或者统计信息陈旧往往会造成执行计划严重劣化，从而导致性能问题。

背景信息

ANALYZE语句可收集与数据库中表内容相关的统计信息，统计结果存储在系统表PG_STATISTIC中。查询优化器会使用这些统计数据，以生成最有效的执行计划。

建议在执行了大批量插入/删除操作后，例行对表或全库执行ANALYZE语句更新统计信息。目前默认收集统计信息的采样比例是30000行（即：GUC参数default_statistics_target默认设置为100），如果表的总行数超过一定行数（大于1600000），建议设置GUC参数default_statistics_target为-2，即按2%收集样本估算统计信息。

对于在批处理脚本或者存储过程中生成的中间表，也需要在完成数据生成之后显式的调用ANALYZE。

对于表中多个列有相关性且查询中有同时基于这些列的条件或分组操作的情况，可尝试收集多列统计信息，以便查询优化器可以更准确地估算行数，并生成更有效的执行计划。

操作步骤

使用以下命令更新某个表或者整个database的统计信息。

```
ANALYZE tablename;                    --更新单个表的统计信息
ANALYZE;                              --更新全库的统计信息
```

使用以下命令进行多列统计信息相关操作。

```
ANALYZE tablename ((column_1, column_2));                    --收集tablename表的column_1、column_2列的多列统计信息

ALTER TABLE tablename ADD STATISTICS ((column_1, column_2));    --添加tablename表的column_1、column_2列的多列统计信息声明
ANALYZE tablename;                                           --收集单列统计信息，并收集已声明的多列统计信息

ALTER TABLE tablename DELETE STATISTICS ((column_1, column_2)); --删除tablename表的column_1、column_2列的多列统计信息或其声明
```


须知

在使用ALTER TABLE tablename ADD STATISTICS语句添加了多列统计信息声明后，系统并不会立刻收集多列统计信息，而是在下次对该表或全库进行ANALYZE时，进行多列统计信息的收集。

说明

如果想直接收集多列统计信息，请使用ANALYZE命令进行收集。

使用EXPLAIN查看各SQL的执行计划时，如果发现某个表SEQ SCAN的输出中rows=10，rows=10是系统给的默认值，有可能该表没有进行ANALYZE，需要对该表执行ANALYZE。

3.3.5 审视和修改表定义

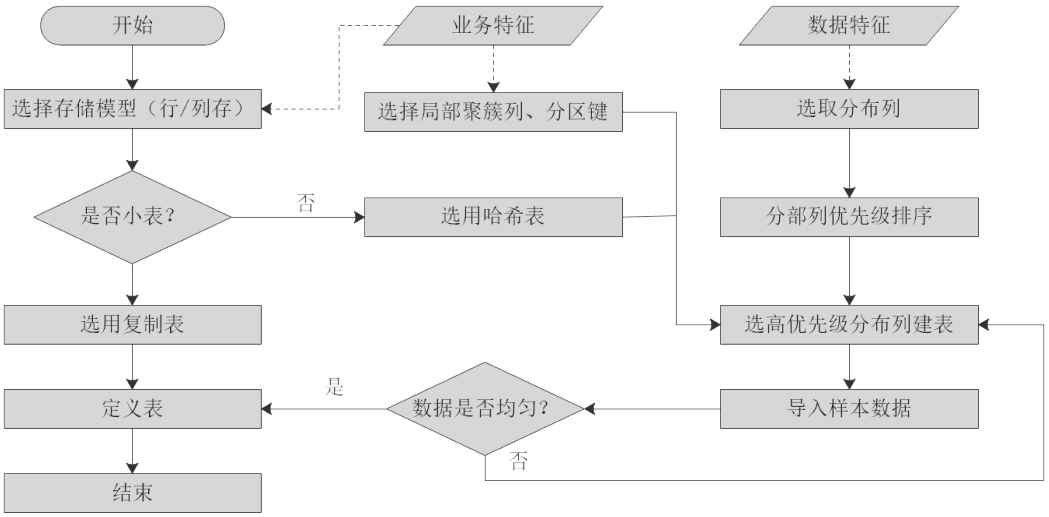
3.3.5.1 审视和修改表定义概述

在分布式框架下，数据分布在各个DN上。一个或者几个DN的数据存在一块物理存储设备上，好的表定义至少需要达到以下几个目标：

1. **表数据均匀分布在各个DN上**，以防止单个DN对应的存储设备空间不足造成集群有效容量下降。选择合适分布列，避免数据分布倾斜可以实现该点。
2. **表Scan压力均匀分散在各个DN上**，以避免单DN的Scan压力过大，形成Scan的单节点瓶颈。分布列不选择基表上等值filter中的列可以实现该点。
3. **减少扫描数据数据量**。通过分区的剪枝机制可以实现该点。
4. **尽量减少随机IO**。通过聚簇/局部聚簇可以实现该点。
5. **尽量避免数据shuffle**，减小网络压力。通过选择join-condition或者group by列为分布列可以更好的实现这点。

从上述描述来看表定义中最重要的一点是分布列的选择。创建表定义一般遵循图3-4所示流程。表定义在数据库设计阶段创建，在SQL调优过程中进行审视和修改。

图 3-4 表定义流程



3.3.5.2 选择存储模型

进行数据库设计时，表设计上的一些关键项将严重影响后续整库的查询性能。表设计对数据存储也有影响：好的表设计能够减少I/O操作及最小化内存使用，进而提升查询性能。

表的存储模型选择是表定义的第一步。客户业务属性是表的存储模型的决定性因素，依据下面表格选择适合当前业务的存储模型。

存储模型	适用场景
行存	点查询(返回记录少，基于索引的简单查询)。 增删改比较多的场景。
列存	统计分析类查询 (group , join多的场景)。

3.3.5.3 选择分布方式

复制表（Replication）方式将表中的全量数据在集群的每一个DN实例上保留一份。主要适用于记录集较小的表。这种存储方式的优点是每个DN上都有该表的全量数据，在join操作中可以避免数据重分布操作，从而减小网络开销，同时减少了plan segment(每个plan segment都会起对应的线程)；缺点是每个DN都保留了表的完整数据，造成数据的冗余。一般情况下只有较小的维度表才会定义为Replication表。

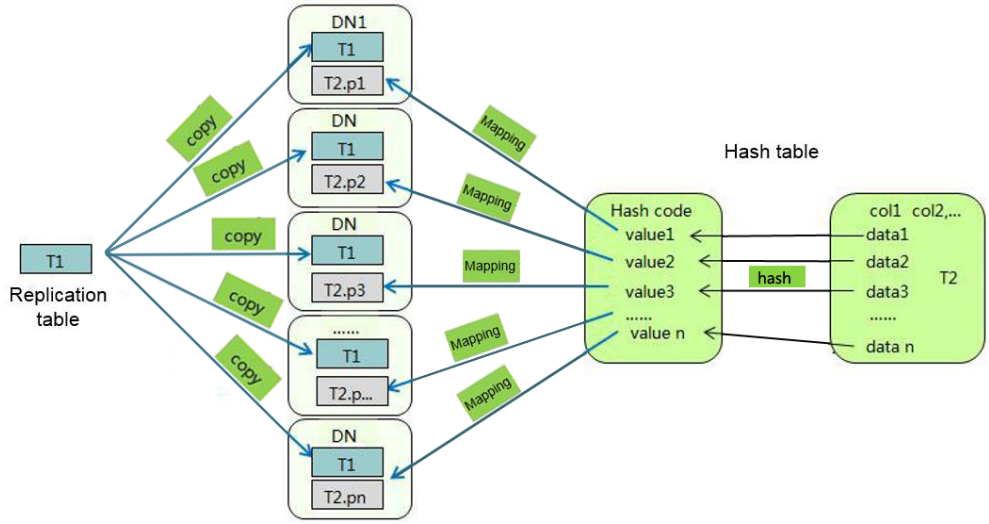
哈希（Hash）表将表中某一个或几个字段进行hash运算后，生成对应的hash值，根据DN实例与哈希值的映射关系获得该元组的目标存储位置。对于Hash分布表，在读/写数据时可以利用各个节点的IO资源，大大提升表的读/写速度。一般情况下大表定义为Hash表。

范围（Range）和列表（List）分布是由用户自定义的分布策略，根据分布列的取值落入满足一定范围或者具体值的对应目标DN，这两种分布方式便于用户灵活地进行数据管理，但对用户本身的数据抽象能力有一定的要求。

策略	描述	适用场景
Hash	表数据通过hash方式散列到集群中的所有DN实例上。	数据量较大的事实表。
Replication	集群中每一个DN实例上都有一份全量表数据。	小表、维度表。
Range	表数据对指定列按照范围进行映射，分布到对应DN。	用户需要自定义分布规则的场景。
List	表数据对指定列按照具体值进行映射，分布到对应DN。	用户需要自定义分布规则的场景。

如图3-5所示，复制表如图中的表T1，哈希表如图中的表T2。

图 3-5 复制表和哈希表



3.3.5.4 选择分布列

Hash分布表的分布列选取至关重要，需要满足以下原则：

1. **列值应比较离散，以便数据能够均匀分布到各个DN**。例如，考虑选择表的主键为分布列，如在人员信息表中选择身份证号码为分布列。
2. **在满足第一条原则的情况下尽量不要选取存在常量filter的列**。例如，表t1相关的部分查询中出现t1的列zqdh存在常量的约束(例如zqdh=’000001’)，那么就应当尽量不用zqdh做分布列。
3. **在满足前两条原则的情况，考虑选择查询中的连接条件为分布列**，以便Join任务能够下推到DN中执行，且减少DN之间的通信数据量。

对于Hash分表策略，如果分布列选择不当，可能导致数据倾斜，查询时出现部分DN的I/O短板，从而影响整体查询性能。因此在采用Hash分表策略之后需对表的数据进行数据倾斜性检查，以确保数据在各个DN上是均匀分布的。可以使用以下SQL检查数据倾斜性

```
select
xc_node_id, count(1)
from tablename
group by xc_node_id
order by xc_node_id desc;
```

其中xc_node_id对应DN，一般来说，**不同DN的数据量相差5%以上即可视为倾斜，如果相差10%以上就必须要调整分布列**。

GaussDB支持多分布列特性，可以更好地满足数据分布的均匀性要求。

3.3.5.5 使用局部聚簇

局部聚簇（Partial Cluster Key）是列存下的一种技术。这种技术可以通过min/max稀疏索引较快的实现基表扫描的filter过滤。Partial Cluster Key可以指定多列，但是一般不建议超过2列。Partial Cluster Key的选取原则：

1. 受基表中的简单表达式约束。这种约束一般形如col op const，其中col为列名，op为操作符 =、>、>=、<=、<，const为常量值。
2. 尽量采用选择度比较高(过滤掉更多数据)的简单表达式中的列。

3. 尽量把选择度比较低的约束col放在Partial Cluster Key中的前面。
4. 尽量把枚举类型的列放在Partial Cluster Key中的前面。

3.3.5.6 使用分区表

分区表是把逻辑上的一张表根据某种方案分成几张物理块进行存储。这张逻辑上的表称之为分区表，物理块称之为分区。分区表是一张逻辑表，不存储数据，数据实际是存储在分区上的。分区表和普通表相比具有以下优点：

1. 改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索效率。
2. 增强可用性：如果分区表的某个分区出现故障，表在其他分区的数据仍然可用。
3. 方便维护：如果分区表的某个分区出现故障，需要修复数据，只修复该分区即可。

GaussDB支持的分区表为范围分区表。

范围分区表：将数据基于范围映射到每一个分区。这个范围是由创建分区表时指定的分区键决定的。分区键经常采用日期，例如将销售数据按照月份进行分区。

3.3.5.7 选择数据类型

高效数据类型，主要包括以下三方面：

1. **尽量使用执行效率比较高的数据类型**

   一般来说整型数据运算(包括=、＞、＜、≧、≦、≠等常规的比较运算，以及group by)的效率比字符串、浮点数要高。比如某客户场景中对列存表进行点查询，filter条件在一个numeric列上，执行时间为10+s；修改numeric为int类型之后，执行时间缩短为1.8s左右。

2. **尽量使用短字段的数据类型**

   长度较短的数据类型不仅可以减小数据文件的大小，提升IO性能；同时也可以减小相关计算时的内存消耗，提升计算性能。比如对于整型数据，如果可以用smallint就尽量不用int，如果可以用int就尽量不用bigint。

3. **使用一致的数据类型**

   表关联列尽量使用相同的数据类型。如果表关联列数据类型不同，数据库必须动态地转化为相同的数据类型进行比较，这种转换会带来一定的性能开销。

3.3.6 典型 SQL 调优点

SQL调优是一个不断分析与尝试的过程：试跑Query，判断性能是否满足要求；如果不满足要求，则通过**SQL执行计划介绍**分析原因并进行针对性优化；然后重新试跑和优化，直到满足性能目标。

3.3.6.1 SQL 自诊断

用户在执行查询或者执行INSERT/DELETE/UPDATE/CREATE TABLE AS语句时，可能会遇到性能问题。这种情况下，通过查询GS_WLM_SESSION_STATISTICS，GS_WLM_SESSION_HISTORY，GS_WLM_SESSION_QUERY_INFO_ALL视图的warning字段可以获得对应查询可能导致性能问题的告警信息，为性能调优提供参考。

告警场景

目前支持对以下7种导致性能问题的场景上报告警。

- 多列/单列统计信息未收集

如果存在单列或者多列统计信息未收集，则上报相关告警。调优方法可以参考**更新统计信息**和**统计信息调优**。

需要特别注意的是，对于基于OBS外表的查询，如果未收集统计信息也会上报统计信息未收集的告警，但是由于OBS外表的analyze的性能比较差，因此，需要用户对这种场景下告警是否通过analyze收集统计信息，以获取更优的性能，和查询本身的复杂度做权衡。

告警信息示例：

整表的统计信息未收集：

```
Statistic Not Collect:
    schema_test.t1
```

单列统计信息未收集：

```
Statistic Not Collect:
    schema_test.t2(c1,c2)
```

多列统计信息未收集:

```
Statistic Not Collect:
    schema_test.t3((c1,c2))
```

单列和多列统计信息未收集：

```
Statistic Not Collect:
    schema_test.t4(c1,c2)    schema_test.t4((c1,c2))
```

- SQL不下推

  对于不下推的SQL，尽可能详细上报导致不下推的原因。调优方法可以参考案例**语句下推调优**。

  - 对于函数导致的不下推，告警导致不下推的函数名信息；
  - 对于不支持下推的语法，会告警对应语法不支持下推，例如：含有With Recursive，Distinct On，row表达式，返回值为record类型的，会告警相应语法不支持下推等等。

告警信息示例：

```
SQL is not plan-shipping, reason : "With Recursive" can not be shipped"
SQL is not plan-shipping, reason : "Function now() can not be shipped"
SQL is not plan-shipping, reason : "Function string_agg() can not be shipped"
```

- HashJoin中大表做内表

如果在表连接过程中使用了HashJoin(可以在GS_WLM_SESSION_HISTORY视图的query_plan字段中查看到)，且连接的内表行数是外表行数的10倍或以上；同时内表在每个DN上的平均行数大于10万行，且发生了下盘，则上报相关告警。调优方法可以参考**使用Plan Hint进行调优**。

告警信息示例：

```
PlanNode[7] Large Table is INNER in HashJoin “Vector Hash Aggregate”
```

- 大表等值连接使用Nestloop

如果在表连接过程中使用了nestloop(可以在GS_WLM_SESSION_HISTORY视图的query_plan字段中查看到)，并且两个表中较大表的行数平均每个DN上的行数大于10万行、表的连接中存在等值连接，则上报相关告警。调优方法可以参考**使用Plan Hint进行调优**。

告警信息示例：

```
PlanNode[5] Large Table with Equal-Condition use Nestloop"Nested Loop"
```

- 大表Broadcast

如果在Broadcast算子中，平均每DN的行数大于10万行，则告警大表broadcast。调优方法可以参考**使用Plan Hint进行调优**。

告警信息示例：

```
PlanNode[5] Large Table in Broadcast "Streaming(type: BROADCAST dop: 1/2)"
```

- 数据倾斜

某表在各DN上的分布，存在某DN上的行数是另一DN上行数的10倍或以上，且有DN中的行数大于10万行，则上报相关告警。调优方法可以参考案例**数据倾斜调优**。

告警信息示例：

```
PlanNode[6] DataSkew:"Seq Scan", min_dn_tuples:0, max_dn_tuples:524288
```

- 估算不准

如果优化器的估算行数和实际行数中的较大值平均每DN行数大于10万行，并且估算行数和实际行数中较大值是较小值的10倍或以上，则上报相关告警。调优方法可以参考**使用Plan Hint进行调优**。

告警信息示例：

```
PlanNode[5] Inaccurate Estimation-Rows: "Hash Join" A-Rows:0, E-Rows:52488
```

规格约束

1. 告警字符串长度上限为2048。如果告警信息超过这个长度（例如存在大量未收集统计信息的超长表名，列名等信息）则不告警，只上报warning：

   WARNING, "Planner issue report is truncated, the rest of planner issues will be skipped"

2. 如果query存在limit节点（即查询语句中包含limit），则不会上报limit节点以下的Operator级别的告警。
3. 对于“数据倾斜”和“估算不准”两种类型告警，在某一个plan树结构下，只上报下层节点的告警，上层节点不再重复告警。这主要是因为这两种类型的告警可能是因为底层触发上层的。例如，如果在scan节点已经存在数据倾斜，那么在上层的hashagg等其他算子很可能也出现数据倾斜。

3.3.6.2 语句下推调优

语句下推介绍

目前，GaussDB优化器在分布式框架下制定语句的执行策略时，有三种执行计划方式：生成下推语句计划、生成分布式执行计划、生成发送语句的分布式执行计划。

- 下推语句计划：指直接将查询语句从CN发送到DN进行执行，然后将执行结果返回给CN。
- 分布式执行计划：指CN对查询语句进行编译和优化，生成计划树，再将计划树发送给DN进行执行，并在执行完毕后返回结果到CN。

- 发送语句的分布式执行计划：上述两种方式都不可行时，将可下推的查询部分组成查询语句（多为基表扫描语句）下推到DN进行执行，获取中间结果到CN，然后在CN执行剩下的部分。

在第3种策略中，要将大量中间结果从DN发送到CN，并且要在CN运行不能下推的部分语句，会导致CN成为性能瓶颈（带宽、存储、计算等）。在进行性能调优的时候，应尽量避免只能选择第3种策略的查询语句。

执行语句不能下推是因为语句中含有**不支持下推的函数**或者**不支持下推的语法**。一般都可以通过等价改写规避执行计划不能下推的问题。

语句下推典型场景

通常而言explain语句后没有显示具体的执行计划算子，仅存在类似关键字“Data Node Scan on”则说明语句已下推给DN去执行。下面从三个维度场景介绍下语句下推以及其支持的范围。

1 单表查询语句下推

在分布式数据库中对于单表查询而言，当前语句是否可以下推需要判断CN是否要进一步参与计算而不是简单收集数据。如果CN要进一步对DN结果进行计算则语句不可下推。通常带有agg, windows function, limit/offset, sort, distinct等关键字都不可下推。

- 可下推：简单查询，无需在CN进一步计算则可以下推。

```
gaussdb=# explain select * from t where c1 > 1;
                              QUERY PLAN
------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_FQS_QUERY__"  (cost=0.00..0.00 rows=0 width=0)
   Node/s: All datanodes
(2 rows)
```

- 不可下推：带有limit子句，对于CN而言不能简单发语句给DN并收集数据，明显与limit语义不符。

```
gaussdb=# explain select * from t limit 1;
                                  QUERY PLAN
-------------------------------------------------------------------------------------
 Limit  (cost=0.00..0.00 rows=1 width=12)
   ->  Data Node Scan on "__REMOTE_LIMIT_QUERY__"  (cost=0.00..0.00 rows=1 width=12)
         Node/s: All datanodes
(3 rows)
```

- 不可下推：带有聚集函数CN不能简单下推语句，而应该对从DN收集结果进一步聚集运算处理。

```
gaussdb=# explain select sum(c1), count(*) from t;
                                  QUERY PLAN
-------------------------------------------------------------------------------------
 Aggregate  (cost=0.10..0.11 rows=1 width=20)
   ->  Data Node Scan on "__REMOTE_GROUP_QUERY__"  (cost=0.00..0.00 rows=20 width=4)
         Node/s: All datanodes
(3 rows)
```

2 多表查询语句下推

多表查询场景下语句能否下推通常与join条件以及分布列有关，即如果join条件与表分布列匹配得上则可下推，否则无法下推。对于复制表来说通常可以下推。

- 创建两个hash分布表。

```
gaussdb=# create table t(c1 int, c2 int, c3 int)distribute by hash(c1);
CREATE TABLE
gaussdb=# create table t1(c1 int, c2 int, c3 int)distribute by hash(c1);
CREATE TABLE
```


- 可下推：join条件满足两个表hash分布列属性。

```
gaussdb=# explain select * from t1 join t on t.c1 = t1.c1;
                              QUERY PLAN
------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_FQS_QUERY__"  (cost=0.00..0.00 rows=0 width=0)
   Node/s: All datanodes
(2 rows)
```

- 不可下推：join条件不满足hash分布列属性，即t1.c2不是t1表的分布列。

```
gaussdb=# explain select * from t1 join t on t.c1 = t1.c2;
                                    QUERY PLAN
-----------------------------------------------------------------------------------------
 Hash Join  (cost=0.25..0.53 rows=20 width=24)
   Hash Cond: (t1.c2 = t.c1)
   ->  Data Node Scan on t1 "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=20 width=12)
         Node/s: All datanodes
   ->  Hash  (cost=0.00..0.00 rows=20 width=12)
         ->  Data Node Scan on t "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=20 width=12)
               Node/s: All datanodes
(7 rows)
```

3 特殊场景

对于有一些特殊场景通常无法下推，例如语句中带有with recursive子句，列存表等不支持下推。

实例分析：自定义函数

对于自定义函数，如果对于确定的输入，有确定的输出，则应将函数定义为immutable类型。

利用TPCDS的销售信息举个例子，比如要写一个函数，获取商品的打折情况，需要一个计算折扣的函数，可以将这个函数定义为：

```
CREATE FUNCTION func_percent_2 (NUMERIC, NUMERIC) RETURNS NUMERIC
AS 'SELECT $1 / $2 WHERE $2 > 0.01'
LANGUAGE SQL
VOLATILE;
```

执行下列语句：

```
SELECT func_percent_2(ss_sales_price, ss_list_price)
FROM store_sales;
```

其执行计划为：

```
 Data Node Scan on store_sales "_REMOTE_TABLE_QUERY_"
   Output: func_percent_2(store_sales.ss_sales_price, store_sales.ss_list_price)
   Remote query: SELECT ss_sales_price, ss_list_price FROM ONLY store_sales WHERE true
(3 rows)
```

可见，func_percent_2并没有被下推，而是将ss_sales_price和ss_list_price收到CN上，再进行计算，消耗大量CN的资源，而且计算缓慢。

由于该自定义函数对确定的输入有确定的输出，如果将该自定义函数改为：

```
CREATE FUNCTION func_percent_1 (NUMERIC, NUMERIC) RETURNS NUMERIC
AS 'SELECT $1 / $2 WHERE $2 > 0.01'
LANGUAGE SQL
IMMUTABLE;
```

执行语句：

```
SELECT func_percent_1(ss_sales_price, ss_list_price)
FROM store_sales;
```

其执行计划为：


```
 Data Node Scan
   Output: (func_percent_1(store_sales.ss_sales_price, store_sales.ss_list_price))
   Remote query: SELECT func_percent_1(ss_sales_price, ss_list_price) AS func_percent_1 FROM store_sales
(3 rows)
```

可见函数func_percent_1被下推到DN执行，提升了执行效率（TPCDS 1000X，3CN18DN，查询效率提升100倍以上）。

不支持下推的函数

首先介绍函数的易变性。在GaussDB中共分三种形态：

- **IMMUTABLE**

  表示该函数在给出同样的参数值时总是返回同样的结果。

- **STABLE**

  表示该函数不能修改数据库，对相同参数值，在同一次表扫描里，该函数的返回值不变，但是返回值可能在不同SQL语句之间变化。

- **VOLATILE**

  表示该函数值可以在一次表扫描内改变，因此不会做任何优化。

函数易变性可以查询pg_proc的provolatile字段获得，i代表IMMUTABLE，s代表STABLE，v代表VOLATILE。另外，在pg_proc中的proshippable字段，取值范围为t/f/NULL，这个字段与provolatile字段一起用于描述函数是否下推。

- 如果函数的provolatile属性为i，则无论proshippable的值是否为t，则函数始终可以下推。
- 如果函数的provolatile属性为s或v，则仅当proshippable的值为t时，函数可以下推。
- random，exec_hadoop_sql，exec_on_extension如果出现CTE中，也不下推。因为这种场景下下推可能出现结果错误。

对于用户自定义函数，可以在创建函数的时候指定provolatile和proshippable属性的值，详细请参考CREATE FUNCTION语法。

对于函数不能下推的场景：

- 如果是系统函数，建议根据业务等价替换这个函数。
- 如果是自定义函数，建议分析客户业务场景，看函数的provolatile和proshippable属性定义是否正确。

不支持下推的语法

以如下三个表定义说明不支持下推的SQL语法。

```
gaussdb=# CREATE TABLE CUSTOMER1
(
    C_CUSTKEY     BIGINT NOT NULL
  , C_NAME        VARCHAR(25) NOT NULL
  , C_ADDRESS     VARCHAR(40) NOT NULL
  , C_NATIONKEY   INT NOT NULL
  , C_PHONE       CHAR(15) NOT NULL
  , C_ACCTBAL     DECIMAL(15,2)   NOT NULL
  , C_MKTSEGMENT  CHAR(10) NOT NULL
  , C_COMMENT     VARCHAR(117) NOT NULL
)
DISTRIBUTE BY hash(C_CUSTKEY);
gaussdb=# CREATE TABLE test_stream(a int,b float); --float不支持重分布
gaussdb=# CREATE TABLE sal_emp ( c1 integer[] ) DISTRIBUTE BY replication;
```

- 不支持returning语句下推

```
gaussdb=# explain update customer1 set C_NAME = 'a' returning c_name;
                              QUERY PLAN
-----------------------------------------------------------------------
 Update on customer1  (cost=0.00..0.00 rows=30 width=187)
   Node/s: All datanodes
   Node expr: c_custkey
   ->  Data Node Scan on customer1 "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=30 width=187)
         Node/s: All datanodes
(5 rows)
```

- 不支持聚集函数中使用order by语句的下推

```
gaussdb=# explain verbose select count ( c_custkey order by c_custkey) from customer1;

                        QUERY PLAN
---------------------------------------------------------------------- Aggregate  (cost=2.50..2.51 rows=1 width=8)
   Output: count(customer1.c_custkey ORDER BY customer1.c_custkey)
   ->  Data Node Scan on customer1 "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=30 width=8)
         Output: customer1.c_custkey
         Node/s: All datanodes
         Remote query: SELECT c_custkey FROM ONLY public.customer1 WHERE true
(6 rows)
```

- count(distinct expr)中的字段不支持重分布，则不支持下推

```
gaussdb=# explain verbose select count(distinct b) from test_stream;
                                  QUERY PLAN
---------------------------------------------------------------------- Aggregate  (cost=2.50..2.51 rows=1 width=8)
   Output: count(DISTINCT test_stream.b)
   ->  Data Node Scan on test_stream "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=30 width=8)
         Output: test_stream.b
         Node/s: All datanodes
         Remote query: SELECT b FROM ONLY public.test_stream WHERE true
(6 rows)
```

- 不支持distinct on用法下推

```
gaussdb=# explain verbose select distinct on (c_custkey) c_custkey from customer1 order by c_custkey;
                                  QUERY PLAN
---------------------------------------------------------------------- Unique  (cost=49.83..54.83 rows=30 width=8)
   Output: customer1.c_custkey
   ->  Sort  (cost=49.83..52.33 rows=30 width=8)
         Output: customer1.c_custkey
         Sort Key: customer1.c_custkey
         ->  Data Node Scan on customer1 "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=30
width=8)
               Output: customer1.c_custkey
               Node/s: All datanodes
               Remote query: SELECT c_custkey FROM ONLY public.customer1 WHERE true
(9 rows)
```

- 不支持数组表达式下推

```
gaussdb=# explain verbose select array[c_custkey,1] from customer1 order by c_custkey;

                        QUERY PLAN
---------------------------------------------------------------------- Sort  (cost=49.83..52.33 rows=30 width=8)
   Output: (ARRAY[customer1.c_custkey, 1::bigint]), customer1.c_custkey
   Sort Key: customer1.c_custkey
   ->  Data Node Scan on "__REMOTE_SORT_QUERY__"  (cost=0.00..0.00 rows=30 width=8)
         Output: (ARRAY[customer1.c_custkey, 1::bigint]), customer1.c_custkey
         Node/s: All datanodes
         Remote query: SELECT ARRAY[c_custkey, 1::bigint], c_custkey FROM ONLY public.customer1
WHERE true ORDER BY 2
(7 rows)
```

- With Recursive当前版本不支持下推的场景和原因如下：

序号	场景	不下推原因
1	包含外表的查询场景	LOG: SQL can't be shipped, reason: RecursiveUnion contains HDFS Table or ForeignScan is not shippable（LOG为CN日志中打印的不下推原因，下同） 外表，当前版本暂不支持下推。
2	多nodegroup场景	LOG: SQL can't be shipped, reason: With-Recursive under multi-nodegroup scenario is not shippable 基表存储nodegroup不相同，或者计算nodegroup与基表不相同，当前版本暂不支持下推。

3	UNION不带ALL，需要去重	LOG: SQL can't be shipped, reason: With-Recursive does not contain "ALL" to bind recursive & none-recursive branches
		例如:
		`WITH recursive t_result AS (`
		`SELECT dm,sj_dm,name,1 as level`
		`FROM test_rec_part`
		`WHERE sj_dm > 10`
		`UNION`
		`SELECT t2.dm,t2.sj_dm,t2.name||' > '||`
		`t1.name,t1.level+1`
		`FROM t_result t1`
		`JOIN test_rec_part t2 ON t2.sj_dm = t1.dm`
		`)`
		`SELECT * FROM t_result t;`

4	基表中有系统表	LOG: SQL can't be shipped, reason: With-Recursive contains system table is not shippable
		例如：
		`WITH RECURSIVE x(id) AS`
		`(`
		`select count(1) from pg_class where oid=1247`
		`UNION ALL`
		`SELECT id+1 FROM x WHERE id < 5`
		`), y(id) AS`
		`(`
		`select count(1) from pg_class where oid=1247`
		`UNION ALL`
		`SELECT id+1 FROM x WHERE id < 10`
		`)`
		`SELECT y.*, x.* FROM y LEFT JOIN x USING (id) ORDER BY 1;`

序号	场景	不下推原因
5	基表扫描只有VALUES子句，仅在CN上即可完成执行。	LOG: SQL can't be shipped, reason: With-Recursive contains only values rte is not shippable 例如： WITH RECURSIVE t(n) AS (VALUES (1) UNION ALL SELECT n+1 FROM t WHERE n < 100) SELECT sum(n) FROM t;
6	相关子查询的关联条件仅在递归部分，非递归部分无关联条件。	LOG: SQL can't be shipped, reason: With-Recursive recursive term correlated only is not shippable 例如： select a.ID,a.Name, (with recursive cte as (select ID, PID, NAME from b where b.ID = 1 union all select parent.ID,parent.PID,parent.NAME from cte as child join b as parent on child.pid=parent.id where child.ID = a.ID) select NAME from cte limit 1) cName from (select id, name, count(*) as cnt from a group by id,name) a order by 1,2;
7	非递归部分带limit为Replicate计划，递归部分为 Hash计划，计划存在冲突。	LOG: SQL can't be shipped, reason: With-Recursive contains conflict distribution in none-recursive(Replicate) recursive(Hash) 例如： WITH recursive t_result AS (select * from(SELECT dm,sj_dm,name,1 as level FROM test_rec_part WHERE sj_dm < 10 order by dm limit 6 offset 2) UNION all SELECT t2.dm,t2.sj_dm,t2.name||' > '|| t1.name,t1.level+1 FROM t_result t1 JOIN test_rec_part t2 ON t2.sj_dm = t1.dm) SELECT * FROM t_result t;

序号	场景	不下推原因
8	多层Recursive嵌套，即recursive的递归部分又嵌套另一个recursive查询。	LOG: SQL can't be shipped, reason: Recursive CTE references recursive CTE "cte" 例如： with recursive cte as (select * from rec_tb4 where id<4 union all select h.id,h.parentID,h.name from (with recursive cte as (select * from rec_tb4 where id<4 union all select h.id,h.parentID,h.name from rec_tb4 h inner join cte c on h.id=c.parentID) SELECT id ,parentID,name from cte order by parentID) h inner join cte c on h.id=c.parentID) SELECT id ,parentID,name from cte order by parentID,1,2,3;

3.3.6.3 子查询调优

子查询背景介绍

应用程序通过SQL语句来操作数据库时会使用大量的子查询，这种写法比直接对两个表做连接操作在结构上和思路上更清晰，尤其是在一些比较复杂的查询语句中，子查询有更完整、更独立的语义，会使SQL对业务逻辑的表达更清晰更容易理解，因此得到了广泛的应用。

GaussDB根据子查询在SQL语句中的位置把子查询分成了子查询、子链接两种形式。

- 子查询SubQuery：对应于查询解析树中的范围表RangeTblEntry，更通俗一些指的是出现在FROM语句后面的独立的SELECT语句。
- 子链接SubLink：对应于查询解析树中的表达式，更通俗一些指的是出现在where/on子句、targetlist里面的语句。

  综上，对于查询解析树而言，SubQuery的本质是范围表、而SubLink的本质是表达式。针对SubLink场景而言，由于SubLink可以出现在约束条件、表达式中，按照GaussDB对sublink的实现，sublink可以分为以下几类：
  - exist_sublink：对应EXIST、NOT EXIST语句
  - any_sublink：对应op ALL(select…)语句，其中OP可以是IN,<,>,=操作符
  - all_sublink：对应op ALL(select…)语句，其中OP可以是IN,<,>,=操作符
  - rowcompare_sublink：对应record op (select …)语句
  - expr_sublink：对应(SELECT with single targetlist item ...)语句
  - array_sublink：对应ARRAY(select…)语句
  - cte_sublink：对应with query(…)语句

GaussDB 对 SubLink 的优化

针对SubLink的优化策略主要是让内层的子查询提升(pullup)，能够和外表直接做关联查询，从而避免生成SubPlan+Broadcast内表的执行计划。判断子查询是否存在性能风险，可以通过explain查询语句查看Sublink的部分是否被转换成SubPlan+Broadcast的执行计划。

例如：

```
select t1.c1,t1.c2
from t1
where t1.c1 in (
    select c2
    from t2
    where t2.c1 = t1.c1
);
```



```
QUERY PLAN
-----------------------------------------------------
Streaming (type: GATHER)
  Node/s: All datanodes
  ->  Seq Scan on t1
        Filter: (SubPlan 1)
        SubPlan 1
          ->  Result
                Filter: (t2.c1 = t1.c1)
                ->  Materialize
                      ->  Streaming(type: BROADCAST)
                            Spawn on: All datanodes
                            ->  Seq Scan on t2
(11 rows)
```

- **目前GaussDB支持的Sublink-Release场景**
  - IN-Sublink无相关条件
    - 不能包含上一层查询的表中的列（可以包含更高层查询表中的列）。
    - 不能包含易变函数。

```
select t1.c1,t1.c2
from t1
where t1.c1 in (
    select c2
    from t2
    where t2.c1 = 1
);
```



```
QUERY PLAN
-----------------------------------------------------
Streaming (type: GATHER)
  Node/s: All datanodes
  ->  Nested Loop Semi Join
        Join Filter: (t1.c1 = t2.c2)
        ->  Seq Scan on t1
        ->  Materialize
              ->  Streaming(type: REDISTRIBUTE)
                    Spawn on: datanode1
                    ->  Seq Scan on t2
                          Filter: (c1 = 1)
(10 rows)
```

  - Exist-Sublink包含相关条件

    Where子句中必须包含上一层查询的表中的列，子查询的其它部分不能含有上层查询的表中的列。其它限制如下。
    - 子查询必须有from子句。
    - 子查询不能含有with子句。
    - 子查询不能含有聚集函数。
    - 子查询里不能包含集合操作、排序、limit、windowagg、having操作。
    - 不能包含易变函数。

```
select t1.c1,t1.c2
from t1
where exists (
    select c2
    from t2
    where t2.c1 = t1.c1
);
```



```
QUERY PLAN
-----------------------------------
Streaming (type: GATHER)
  Node/s: All datanodes
  ->  Hash Semi Join
        Hash Cond: (t1.c1 = t2.c1)
        ->  Seq Scan on t1
        ->  Hash
              ->  Seq Scan on t2
(7 rows)
```

- 包含聚集函数的等值相关子查询的提升

  子查询的where条件中必须含有来自上一层的列，而且此列必须和子查询本层涉及表中的列做相等判断，且这些条件必须用and连接。其它地方不能包含上层的列。其它限制条件如下。

  - 子查询中where条件包含的表达式(列名)必须是表中的列。
  - 子查询的Select关键字后，必须有且仅有一个输出列，此输出列必须是聚集函数(如max)，并且聚集函数的参数(t2.c2)不能是来自外层表(t1)中的列。聚集函数不能是count。

    例如，下列示例可以提升。

    ```
    select * from t1 where c1 >(
        select max(t2.c1) from t2 where t2.c1=t1.c1
    );
    ```

    下列示例不能提升，因为子查询没有聚集函数。

    ```
    select * from t1 where c1 >(
        select  t2.c1 from t2 where t2.c1=t1.c1
    );
    ```

    下列示例不能提升，因为子查询有两个输出列。

    ```
    select * from t1 where (c1,c2) >(
        select  max(t2.c1),min(t2.c2) from t2 where t2.c1=t1.c1
    );
    ```

  - 子查询必须是from子句。
  - 子查询中不能有groupby、having、集合操作。
  - 子查询只能是inner join。

    例如：下列示例不能提升。

    ```
    select * from t1 where c1 >(
        select max(t2.c1) from t2 full join t3 on (t2.c2=t3.c2) where t2.c1=t1.c1
    );
    ```

  - 子查询的targetlist中不能包含返回set的函数。
  - 子查询的where条件中必须含有来自上一层的列，而且此列必须和子查询层涉及表中的列做相等判断，且这些条件必须用and连接。其它地方不能包含上层的上层中的列。例如：下列示例中的最内层子链接可以提升。

    ```
    select * from t3 where t3.c1=(
          select t1.c1
          from t1 where c1 >(
                select max(t2.c1) from t2 where t2.c1=t1.c1
    ));
    ```

    基于上面的示例，再加一个条件，则不能提升，因为最内侧子查询引用了上层中的列。示例如下：

    ```
    select * from t3 where t3.c1=(
          select t1.c1
          from t1 where c1 >(
                select max(t2.c1) from t2 where t2.c1=t1.c1 and t3.c1>t2.c2

    ));
    ```

- 提升OR子句中的SubLink

  当WHERE过滤条件中有OR连接的EXIST相关SubLink，

  例如：


```
select a, c from t1
where t1.a = (select avg(a) from t3 where t1.b = t3.b) or
exists (select * from t4 where t1.c = t4.c);
```

将OR-ed连接的EXIST相关子查询OR字句的提升过程：

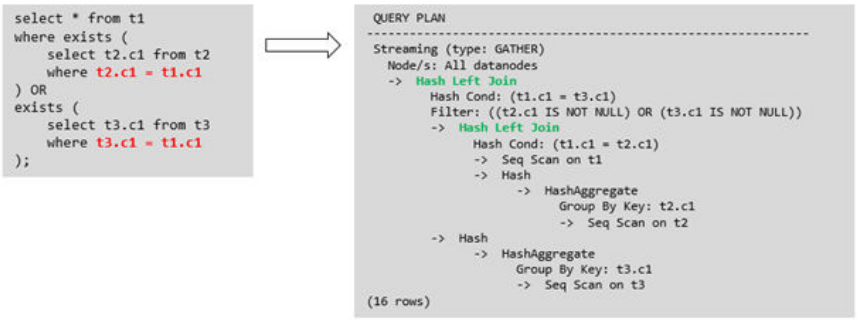
i. 提取where条件中，or子句中的opExpr。为：t1.a = (select avg(a) from t3 where t1.b = t3.b)

ii. 这个op操作中包含subquery，判断是否可以提升，如果可以提升，重写subquery为：select avg(a), t3.b from t3 group by t3.b，生成not null条件t3.b is not null，并将这个opexpr用这个not null条件替换。此时SQL变为：

```
select a, c
from t1 left join (select avg(a) avg, t3.b from t3 group by t3.b)  as t3 on (t1.a = avg and t1.b = t3.b)
where t3.b is not null or exists (select * from t4 where t1.c = t4.c);
```

iii. 再次提取or子句中的exists sublink，exists (select * from t4 where t1.c = t4.c)，判断是否可以提升，如果可以提升，转换subquery为：select t4.c from t4 group by t4.c生成NotNull条件t4.c is not null提升查询，SQL变为：

```
select t1.a, t1.c from t1 left join (select avg(a) avg, t3.b from t3 group by t3.b) as t3 on (t1.a = avg and t1.b = t3.b) left join (select t5.c from t5 group by t5.c) as t5 on (t1.c = t5.c) where t3.b is not null or t5.c is not null;
```



- **目前GaussDB不支持的Sublink-Release场景**

  除了以上场景之外都不支持Sublink提升，因此关联子查询会被计划成SubPlan +Broadcast的执行计划，当inner表的数据量较大时则会产生性能风险。

  如果相关子查询中跟外层的两张表做join，那么无法提升该子查询，需要通过将父SQL创建成with子句，然后再跟子查询中的表做相关子查询查询。

  例如：

```
select distinct t1.a, t2.a
from t1 left join t2 on t1.a=t2.a and not exists (select a,b from test1 where test1.a=t1.a and test1.b=t2.a);
```

  改写为

```
with temp as
(
      select * from (select t1.a as a, t2.a as b from t1 left join t2 on t1.a=t2.a)

)
select distinct a,b
from temp
where not exists (select a,b from test1 where temp.a=test1.a and temp.b=test1.b);
```

  – 出现在targetlist里的相关子查询无法提升(不含count)

    例如：

```
explain (costs off)
select (select c2 from t2 where t1.c1 = t2.c1) ssq, t1.c2
```



```
from t1
where t1.c2 > 10;
```

执行计划为：

```
explain (costs off)
select (select c2 from t2 where t1.c1 = t2.c1) ssq, t1.c2
from t1
where t1.c2 > 10;
                    QUERY PLAN
-----------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on t1
         Filter: (c2 > 10)
         SubPlan 1
           ->  Result
                 Filter: (t1.c1 = t2.c1)
                 ->  Materialize
                       ->  Streaming(type: BROADCAST)
                             Spawn on: All datanodes
                             ->  Seq Scan on t2
(11 rows)
```

由于相关子查询出现在targetlist(查询返回列表)里，对于t1.c1=t2.c1不匹配的场景仍然需要输出值，因此使用right-outerjoin关联t2&t1，以确保t1.c1=t2.c1在不匹配时，子SSQ能够返回不匹配的补空值。

说明

SSQ和CSSQ的解释如下：

- SSQ：ScalarSubQuery一般指返回1行1列scalar值的sublink，简称SSQ。
- CSSQ：Correlated-ScalarSubQuery和SSQ相同不过是指包含相关条件的SSQ。

上述SQL语句可以改写为：

```
with ssq as
(
   select * from t1 where t1.c2 >10
)
select t2.c2,ssq.c2 from t2 right join ssq on ssq.c1 = t2.c1;
```

改写后的执行计划为：

```
          QUERY PLAN
---------------------------------
 Hash Right Join
   Hash Cond: (t2.c1 = t1.c1)
   ->  Seq Scan on t2
   ->  Hash
         ->  Seq Scan on t1
               Filter: (c2 > 10)
(6 rows)
```

可以看到出现在SSQ返回列表里的相关子查询SSQ，已经被提升成Right Join，从而避免当内表T2较大时出现SubPlan+Broadcast计划导致性能变差。

- 出现在targetlist里的相关子查询无法提升(带count)

例如：

```
select (select count(*) from t2 where t2.c1=t1.c1) cnt, t1.c1, t3.c1
from t1,t3
where t1.c1=t3.c1 order by cnt, t1.c1;
```

执行计划为

```
                          QUERY PLAN
-------------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Sort
         Sort Key: ((SubPlan 1)), t1.c1
```



```
      ->  Hash Join
            Hash Cond: (t1.c1 = t3.c1)
            ->  Seq Scan on t1
            ->  Hash
                  ->  Seq Scan on t3
            SubPlan 1
              ->  Aggregate
                    ->  Result
                          Filter: (t2.c1 = t1.c1)
                          ->  Materialize
                                ->  Streaming(type: BROADCAST)
                                      Spawn on: All datanodes
                                      ->  Seq Scan on t2
(17 rows)
```

由于相关子查询出现在targetlist(查询返回列表)里，对于t1.c1=t2.c1不匹配的场景仍然需要输出值，因此使用left-outerjoin关联t1&t2确保t1.c1=t2.c1在不匹配时子SSQ能够返回不匹配的补空值，但是这里带了count语句及时在t1.c1=t2.t1不匹配时需要输出0，因此可以使用一个case-when NULL then 0 else count(*)来代替。

上述SQL语句可以改写为：

```
with ssq as
(
   select count(*) cnt, c1 from t2 group by c1
)
select case when
          ssq.cnt is null then 0
          else ssq.cnt
     end cnt, t1.c1, t3.c1
from t1 left join ssq on ssq.c1 = t1.c1,t3
where t1.c1 = t3.c1
order by ssq.cnt, t1.c1;
```

改写后的执行计划为

```
 QUERY PLAN
-----------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Sort
         Sort Key: (count(*)), t1.c1
         ->  Hash Join
               Hash Cond: (t1.c1 = t3.c1)
               ->  Hash Left Join
                     Hash Cond: (t1.c1 = t2.c1)
                     ->  Seq Scan on t1
                     ->  Hash
                           ->  HashAggregate
                                 Group By Key: t2.c1
                                 ->  Seq Scan on t2
               ->  Hash
                     ->  Seq Scan on t3
(15 rows)
```

- 相关条件为不等值场景

  例如：

  ```
  select t1.c1, t1.c2
  from t1
  where t1.c1 = (select agg() from t2.c2 > t1.c2);
  ```

  对于非等值相关条件的SubLink目前无法提升，从语义上可以通过做2次join（一次CorrelationKey，一次rownum自关联）达到提升改写的目的。

  改写方案有两种。

  - 子查询改写方式

    ```
    select t1.c1, t1.c2
    from t1, (
    ```



```
    select t1.rowid, agg() aggref
    from t1,t2
    where t1.c2 > t2.c2 group by t1.rowid
) dt /* derived table */
where t1.rowid = dt.rowid AND t1.c1 = dt.aggref;
```

- CTE改写方式

```
WITH dt as
(
    select t1.rowid, agg() aggref
    from t1,t2
    where t1.c2 > t2.c2 group by t1.rowid
)
select t1.c1, t1.c2
from t1, dt
where t1.rowid = dt.rowid AND
t1.c1 = dt.aggref;
```

须知

- 目前尚无高效的实现表、中间结果集的全局唯一rowid，因此目前此类场景很难改写，建议通过业务层进行规避，或者可以使用t1.xc_nodeid + t1.ctid进行rowid关联，但是xc_nodeid的重复率较高会导致join关联效率变低，而xc_node_id+ctid类型无法作为hashjoin的关联条件。
- 对于AGG类型为count(*)时需要进行CASE-WHEN对没有match的场景补0处理，非COUNT(*)场景NULL处理。
- CTE改写方式如果有sharescan支持性能上能够更优。

更多优化示例

示例1：修改基表为replicate表，并且在过滤列上创建索引。

```
create table master_table (a int);
create table sub_table(a int, b int);
select a from master_table group by a having a in (select a from sub_table);
```

上述事例中存在一个相关性子查询，为了提升查询的性能，可以将sub_table修改为一个relication表，并且在字段a上创建一个index。

示例2：修改select语句，将子查询修改为和主表的join，或者修改为可以提升的subquery，但是在修改前后需要保证语义的正确性。

```
explain (costs off)select * from master_table as t1 where t1.a in (select t2.a from sub_table as t2 where t1.a
= t2.b);
                      QUERY PLAN
-------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on master_table t1
         Filter: (SubPlan 1)
         SubPlan 1
           ->  Result
                 Filter: (t1.a = t2.b)
                 ->  Materialize
                       ->  Streaming(type: BROADCAST)
                             Spawn on: All datanodes
                             ->  Seq Scan on sub_table t2
(11 rows)
```

上面事例计划中存在一个subPlan，为了消除这个subPlan可以修改语句为：


```
explain(costs off) select * from master_table as t1 where exists (select t2.a from sub_table as t2 where t1.a
= t2.b and t1.a = t2.a);
                  QUERY PLAN
-----------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Hash Semi Join
         Hash Cond: (t1.a = t2.b)
         ->  Seq Scan on master_table t1
         ->  Hash
               ->  Streaming(type: REDISTRIBUTE)
                     Spawn on: All datanodes
                     ->  Seq Scan on sub_table t2
(9 rows)
```

从计划可以看出，subPlan消除了，计划变成了两个表的semi join，这样会大大提高执行效率。

3.3.6.4 统计信息调优

统计信息调优介绍

GaussDB是基于代价估算生成的最优执行计划。优化器需要根据analyze收集的统计信息进行行数估算和代价估算，因此统计信息对优化器行数估算和代价估算起着至关重要的作用。通过analyze收集全局统计信息，主要包括：pg_class表中的relpages和reltuples；pg_statistic表中的stadistinct、stanullfrac、stanumbersN、stavaluesN、histogram_bounds等。

实例分析 1：未收集统计信息导致查询性能差

在很多场景下，由于查询中涉及到的表或列没有收集统计信息，会对查询性能有很大的影响。

表结构如下所示：

```
CREATE TABLE LINEITEM
(
L_ORDERKEY        BIGINT        NOT NULL
, L_PARTKEY       BIGINT        NOT NULL
, L_SUPPKEY       BIGINT        NOT NULL
, L_LINENUMBER    BIGINT        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
) with (orientation = column, COMPRESSION = MIDDLE) distribute by hash(L_ORDERKEY);

CREATE TABLE ORDERS
(
O_ORDERKEY        BIGINT        NOT NULL
, O_CUSTKEY       BIGINT        NOT NULL
, O_ORDERSTATUS   CHAR(1)       NOT NULL
, O_TOTALPRICE    DECIMAL(15,2) NOT NULL
, O_ORDERDATE     DATE NOT NULL
, O_ORDERPRIORITY CHAR(15)      NOT NULL
, O_CLERK         CHAR(15)      NOT NULL
```

```
, O_SHIPPRIORITY  BIGINT        NOT NULL
, O_COMMENT       VARCHAR(79)   NOT NULL
)with (orientation = column, COMPRESSION = MIDDLE) distribute by hash(O_ORDERKEY);
```

查询语句如下所示：

```
explain verbose select
count(*) as numwait
from
lineitem l1,
orders
where
o_orderkey = l1.l_orderkey
and o_orderstatus = 'F'
and l1.l_receiptdate > l1.l_commitdate
and not exists (
select
*
from
lineitem l3
where
l3.l_orderkey = l1.l_orderkey
and l3.l_suppkey <> l1.l_suppkey
and l3.l_receiptdate > l3.l_commitdate
)
order by
numwait desc;
```

当出现该问题时，可以通过如下方法确认查询中涉及到的表或列有没有做过analyze收集统计信息。

1. 通过explain verbose执行query分析执行计划时会提示WARNING信息，如下所示：
   ```
   WARNING:Statistics in some tables or columns(public.lineitem.l_receiptdate,
   public.lineitem.l_commitdate, public.lineitem.l_orderkey, public.lineitem.l_suppkey,
   public.orders.o_orderstatus, public.orders.o_orderkey) are not collected.
   HINT:Do analyze for them in order to generate optimized plan.
   ```
2. 可以通过在pg_log目录下的日志文件中查找以下信息来确认当前执行的query是否由于没有收集统计信息导致查询性能变差。
   ```
   2017-06-14 17:28:30.336 CST 140644024579856 20971684 [BACKEND] LOG:Statistics in some tables
   or columns(public.lineitem.l_receiptdate, public.lineitem.l_commitdate, public.lineitem.l_orderkey,
   public.linei
   tem.l_suppkey, public.orders.o_orderstatus, public.orders.o_orderkey) are not collected.
   2017-06-14 17:28:30.336 CST 140644024579856 20971684 [BACKEND] HINT:Do analyze for them in
   order to generate optimized plan.
   ```

当通过以上方法查看到哪些表或列没有做analyze，可以通过对WARNING或日志中上报的表或列做analyze来解决由于未收集统计信息导致查询变慢的问题。

实例分析 3：多表 join 的复杂查询存在中间结果不准调优

现象描述：查询与指定人在前后15分钟内、同一网吧登记上网的人员信息：

```
SELECT
C.WBM,
C.DZQH,
C.DZ,
B.ZJHM,
B.SWKSSJ,
B.XWSJ
FROM
b_zyk_wbswxx A,
b_zyk_wbswxx B,
b_zyk_wbcs C
WHERE
A.ZJHM = '522522******3824'
```



```
AND A.WBDM = B.WBDM
AND A.WBDM = C.WBDM
AND abs(to_date(A.SWKSSJ,'yyyymmddHH24MISS') - to_date(B.SWKSSJ,'yyyymmddHH24MISS')) <
INTERVAL '15 MINUTES'
ORDER BY
B.SWKSSJ,
B.ZJHM
limit 10 offset 0
;
```

执行计划如图3-6所示。该查询实际耗时约12秒。

图 3-6 应用 unlogged table 案例（一）

```
                                  QUERY PLAN
------------------------------------------------------------------------------------------
 Limit  (cost=221021.41..221021.43 rows=10 width=120)
   ->  Sort  (cost=221021.41..221022.01 rows=240 width=120)
         Sort Key: b.swkssj, b.zjhm
         ->  Streaming (type: GATHER)  (cost=221015.62..221016.22 rows=240 width=120)
               Node/s: All datanodes
               ->  Limit  (cost=9208.98..9209.01 rows=10 width=120)
                     ->  Sort  (cost=9208.98..9211.60 rows=1048 width=120)
                           Sort Key: b.swkssj, b.zjhm
                           ->  Nested Loop  (cost=23.27..9186.34 rows=1048 width=120)
                                 Join Filter: (((a.zjhm)::text <> (b.zjhm)::text) AND ((a.wbdm)::text = (b.wbdm)::text)
                                 AND (abs(((to_date((a.swkssj)::text, 'yyyymmddHH24MISS'::text)
                                   - to_date((b.swkssj)::text, 'yyyymmddHH24MISS'::text)))::numeric) < .0104166666666667))
                                 ->  Streaming(type: BROADCAST)  (cost=0.00..6.33 rows=24 width=135)
                                       Spawn on: All datanodes
                                       ->  Nested Loop  (cost=0.00..106.80 rows=1 width=135)
                                             ->  Streaming(type: BROADCAST)  (cost=0.00..24.75 rows=264 width=48)
                                                   Spawn on: All datanodes
                                                   ->  Partition Iterator  (cost=0.00..48.44 rows=11 width=48)
                                                         Iterations: 25
                                                         ->  Partitioned Index Scan using idx_b_zyk_wbswxx_zjhm on b_zyk_wbswxx a  (cost=0.00..48.44 rows=11 width=48)
                                                               Index Cond: ((zjhm)::text = '522522******3824'::text)
                                                               Selected Partitions:  1..25
                                             ->  Index Scan using idx_b_zyk_wbcs_wbdm on b_zyk_wbcs c  (cost=0.00..2.82 rows=1 width=87)
                                                   Index Cond: ((wbdm)::text = (a.wbdm)::text)
                                 ->  Partition Iterator  (cost=23.27..7306.33 rows=2454 width=63)
                                       Iterations: 25
                                       ->  Partitioned Bitmap Heap Scan on b_zyk_wbswxx b  (cost=23.27..7306.33 rows=2454 width=63)
                                             Recheck Cond: ((wbdm)::text = (c.wbdm)::text)
                                             Filter: ('522522198405243824'::text <> (zjhm)::text)
                                             Selected Partitions:  1..25
                                             ->  Partitioned Bitmap Index Scan on idx_b_zyk_wbswxx_wbdm  (cost=0.00..22.65 rows=2454 width=0)
                                                   Index Cond: ((wbdm)::text = (c.wbdm)::text)
(30 rows)
```

优化分析：分析过程如下：

1. 分析该执行计划发现，扫描节点已使用Index Scan，耗时主要在最外层Nest Loop Join的Join Filter计算中，且该计算执行了字符串的加减法和不等值比较。
2. 考虑使用unlogged table保存目标人的上网信息，且在插入时处理上网开始时间和终止时间，以避免后续进行时间加减。

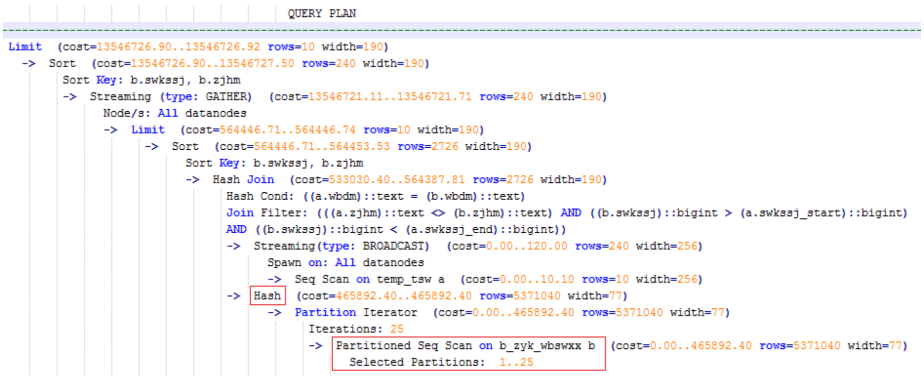
```
//创建临时unlogged table
CREATE UNLOGGED TABLE temp_tsw
(
ZJHM         NVARCHAR2(18),
WBDM          NVARCHAR2(14),
SWKSSJ_START NVARCHAR2(14),
SWKSSJ_END   NVARCHAR2(14),
WBM           NVARCHAR2(70),
DZQH          NVARCHAR2(6),
DZ          NVARCHAR2(70),
IPDZ          NVARCHAR2(39)
)
;
//插入目标人的上网记录，并处理上网开始和结束时间。
INSERT INTO
temp_tsw
SELECT
A.ZJHM,
A.WBDM,
to_char((to_date(A.SWKSSJ,'yyyymmddHH24MISS') - INTERVAL '15
MINUTES'),'yyyymmddHH24MISS'),
to_char((to_date(A.SWKSSJ,'yyyymmddHH24MISS') + INTERVAL '15
MINUTES'),'yyyymmddHH24MISS'),
B.WBM,B.DZQH,B.DZ,B.IPDZ
FROM
```

```
b_zyk_wbswxx A,
b_zyk_wbcs B
WHERE
A.ZJHM='522522******3824' AND A.WBDM = B.WBDM
;

//查询和目标人在前后十五分钟内在同一网吧上网的人员信息，比较大小时强制转换为int8。
SELECT
A.WBM,
A.DZQH,
A.DZ,
A.IPDZ,
B.ZJHM,
B.XM,
to_date(B.SWKSSJ,'yyyymmddHH24MISS') as SWKSSJ,
to_date(B.XWSJ,'yyyymmddHH24MISS') as XWSJ,
B.SWZDH
FROM temp_tsw A,
b_zyk_wbswxx B
WHERE
A.ZJHM <> B.ZJHM
AND A.WBDM = B.WBDM
AND (B.SWKSSJ)::int8 > (A.swkssj_start)::int8
AND (B.SWKSSJ)::int8 < (A.swkssj_end)::int8
order by
B.SWKSSJ,
B.ZJHM
limit 10 offset 0
;
```

上述查询耗时约7秒，执行计划如图3-7所示。

图 3-7 应用 unlogged table 案例（二）



```
QUERY PLAN
------------------------------------------------------------------------------------------
Limit  (cost=13546726.90..13546726.92 rows=10 width=190)
  ->  Sort  (cost=13546726.90..13546727.50 rows=240 width=190)
        Sort Key: b.swkssj, b.zjhm
        ->  Streaming (type: GATHER)  (cost=13546721.11..13546721.71 rows=240 width=190)
              Node/s: All datanodes
              ->  Limit  (cost=564446.71..564446.74 rows=10 width=190)
                    ->  Sort  (cost=564446.71..564453.53 rows=2726 width=190)
                          Sort Key: b.swkssj, b.zjhm
                          ->  Hash Join  (cost=533030.40..564387.81 rows=2726 width=190)
                                Hash Cond: ((a.wbdm)::text = (b.wbdm)::text)
                                Join Filter: (((a.zjhm)::text <> (b.zjhm)::text) AND ((b.swkssj)::bigint > (a.swkssj_start)::bigint)
                                AND ((b.swkssj)::bigint < (a.swkssj_end)::bigint))
                                ->  Streaming(type: BROADCAST)  (cost=0.00..120.00 rows=240 width=256)
                                      Spawn on: All datanodes
                                      ->  Seq Scan on temp_tsw a  (cost=0.00..10.10 rows=10 width=256)
                                ->  Hash  (cost=465892.40..465892.40 rows=5371040 width=77)
                                      ->  Partition Iterator  (cost=0.00..465892.40 rows=5371040 width=77)
                                            Iterations: 25
                                            ->  Partitioned Seq Scan on b_zyk_wbswxx b  (cost=0.00..465892.40 rows=5371040 width=77)
                                                  Selected Partitions:  1..25
```

3. 分析上述执行计划，发现执行了Hash Join，对大表b_zyk_wbswxx建立了Hash Table。由于该表数据量大，创建过程耗时较长。

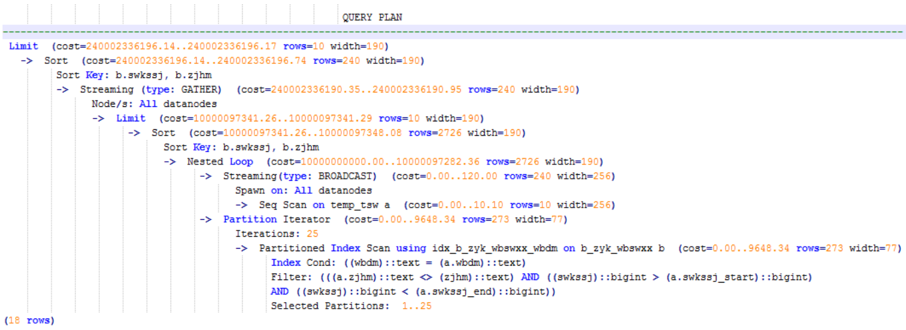
   由于temp_tsw中仅包含几百条记录，且temp_tsw和b_zyk_wbswxx均通过wbdm（网吧代码）执行等值连接。因此，如果Join方式改为Nest Loop Join，则扫描节点可以实现Index Scan，性能预计将会提升。

4. 执行如下语句，将Join方式改为Nest Loop Join。

```
SET enable_hashjoin = off;
```

   执行计划如图3-8所示。查询耗时约3秒。

图 3-8 应用 unlogged table 案例（三）



```
QUERY PLAN
----------------------------------------------------------------------------------------------------------------
Limit  (cost=240002336196.14..240002336196.17 rows=10 width=190)
  ->  Sort  (cost=240002336196.14..240002336196.74 rows=240 width=190)
        Sort Key: b.swkssj, b.zjhm
        ->  Streaming (type: GATHER)  (cost=240002336190.35..240002336190.95 rows=240 width=190)
              Node/s: All datanodes
              ->  Limit  (cost=10000097341.26..10000097341.29 rows=10 width=190)
                    ->  Sort  (cost=10000097341.26..10000097348.08 rows=2726 width=190)
                          Sort Key: b.swkssj, b.zjhm
                          ->  Nested Loop  (cost=10000000000.00..10000097282.36 rows=2726 width=190)
                                ->  Streaming(type: BROADCAST)  (cost=0.00..120.00 rows=240 width=256)
                                      Spawn on: All datanodes
                                      ->  Seq Scan on temp_tsw a  (cost=0.00..10.10 rows=10 width=256)
                                ->  Partition Iterator  (cost=0.00..9648.34 rows=273 width=77)
                                      Iterations: 25
                                      ->  Partitioned Index Scan using idx_b_zyk_wbswxx_wbdm on b_zyk_wbswxx b  (cost=0.00..9648.34 rows=273 width=77)
                                            Index Cond: ((wbdm)::text = (a.wbdm)::text)
                                            Filter: (((a.zjhm)::text <> (zjhm)::text) AND ((swkssj)::bigint > (a.swkssj_start)::bigint)
                                            AND ((swkssj)::bigint < (a.swkssj_end)::bigint))
                                            Selected Partitions:  1..25
(18 rows)
```

5. 使用unlogged table保存结果集并用于分页显示。

   如果需要在上层应用页面实现分页显示，需要修改offset值确定显示目标页的结果集。按此实现，每次翻页时均执行上面查询语句，耗时较长。

   为解决上述问题，建议使用unlogged table保存结果集。

```
//创建保存结果集的unlogged table
CREATE UNLOGGED TABLE temp_result
(
WBM      NVARCHAR2(70),
DZQH     NVARCHAR2(6),
DZ       NVARCHAR2(70),
IPDZ     NVARCHAR2(39),
ZJHM     NVARCHAR2(18),
XM       NVARCHAR2(30),
SWKSSJ   date,
XWSJ     date,
SWZDH    NVARCHAR2(32)
);

//将结果集插入unlogged table，插入耗时约3秒。
INSERT INTO
temp_result
SELECT
A.WBM,
A.DZQH,
A.DZ,
A.IPDZ,
B.ZJHM,
B.XM,
to_date(B.SWKSSJ,'yyyymmddHH24MISS') as SWKSSJ,
to_date(B.XWSJ,'yyyymmddHH24MISS') as XWSJ,
B.SWZDH
FROM temp_tsw A,
b_zyk_wbswxx B
WHERE
A.ZJHM <> B.ZJHM
AND A.WBDM = B.WBDM
AND (B.SWKSSJ)::int8 > (A.swkssj_start)::int8
AND (B.SWKSSJ)::int8 < (A.swkssj_end)::int8
;

//查询结果集表进行分页显示，分页查询耗时约10ms。
SELECT
*
FROM
temp_result
ORDER BY
SWKSSJ,
ZJHM
LIMIT 10 OFFSET 0;
```

注意

收集更准确的统计信息，通常会改善查询性能，但是也有可能使性能劣化。如果遇到性能劣化，可以考虑：

- 恢复默认的统计信息。
- 使用plan hint来调整到之前的查询计划。（详细参见**使用Plan Hint进行调优**）

3.3.6.5 算子级调优

算子级调优介绍

一个查询语句要经过多个算子步骤才会输出最终的结果。由于各别算子耗时过长导致整体查询性能下降的情况比较常见。这些算子是整个查询的瓶颈算子。通用的优化手段是EXPLAIN ANALYZE/PERFORMANCE命令查看执行过程的瓶颈算子，然后进行针对性优化。

如下面的执行过程信息中，Hashagg算子的执行时间占总时间的：(51016-13535)/56476 ≈66%，此处Hashagg算子就是这个查询的瓶颈算子，在进行性能优化时应当优先考虑此算子的优化。

id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	-> Row Adapter	56476.397	10000000	237060	19KB			20	20933222.75
2	-> Vector Streaming (type: GATHER)	55664.220	10000000	237060	243KB			20	20933222.75
3	-> Vector Hash Aggregate	[55124.685,55132.180]	10000000	237060	[29349KB, 29441KB]	16MB	[20,20]	20	20918406.50
4	-> Vector Streaming(type: REDISTRIBUTE)	[52519.781,53709.779]	339364604	4856184	[1219KB, 1219KB]	1MB		20	10461210.85
5	-> Vector Hash Aggregate	[35675.636,51016.424]	339364604	4856184	[732850KB, 746894KB]	16MB	[20,20]	20	10457195.65
6	-> Vector Partition Iterator	[9035.202,13565.884]	970000000	935838097	[9KB, 9KB]	1MB		20	10195891.68
7	-> Partitioned CStore Scan on xuji.e_mp_day_energy_csv_1	[9015.645,13535.346]	970000000	935838097	[845KB, 845KB]	1MB		20	10195891.68

(7 rows)

算子级调优示例

示例1：基表扫描时，对于点查或者范围扫描等过滤大量数据的查询，如果使用SeqScan全表扫描会比较耗时，可以在条件列上建立索引选择IndexScan进行索引扫描提升扫描效率。

```
gaussdb=#  explain (analyze on, costs off) select * from store_sales where ss_sold_date_sk = 2450944;
 id |              operation           |         A-time         | A-rows | Peak Memory  | A-width
----+--------------------------------+---------------------+--------+--------------+---------
  1 | ->  Streaming (type: GATHER)    | 3666.020             |   3360 | 195KB         |
  2 |    ->  Seq Scan on store_sales | [3594.611,3594.611] |   3360 | [34KB, 34KB] |
(2 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
   2 --Seq Scan on store_sales
         Filter: (ss_sold_date_sk = 2450944)
         Rows Removed by Filter: 4968936
gaussdb=#  create index idx on store_sales_row(ss_sold_date_sk);
CREATE INDEX
gaussdb=#  explain (analyze on, costs off) select * from store_sales_row where ss_sold_date_sk = 2450944;
 id |                    operation                     |     A-time      | A-rows | Peak Memory  | A-width
----+------------------------------------------------+-----------------+--------+--------------+----------
  1 | ->  Streaming (type: GATHER)                     | 81.524          |   3360 | 195KB         |
  2 |    ->  Index Scan using idx on store_sales_row | [13.352,13.352] |   3360 | [34KB, 34KB] |
(2 rows)
```

上述例子中，全表扫描返回3360条数据，过滤掉大量数据，在ss_sold_date_sk列上建立索引后，使用IndexScan扫描效率显著提高，从3.6秒提升到13毫秒。

示例2：如果从执行计划中看，两表join选择了NestLoop，而实际行数比较大时，NestLoop Join可能执行比较慢。如下的例子中NestLoop耗时181秒，如果设置参数

enable_mergejoin=off关掉Merge Join，同时设置参数enable_nestloop=off关掉NestLoop，让优化器选择HashJoin，则Join耗时提升至200多毫秒。

```
gaussdb =#  explain analyze select count(*) from store_sales ss, item i where ss.ss_item_sk = i.i_item_sk;
 id |                    operation                    |        A-time         | A-rows  | E-rows  |  Peak Memory   | E-memory | A-width | E-width |   E-costs
----+-------------------------------------------------+-----------------------+---------+---------+----------------+----------+---------+---------+-------------
  1 | ->  Row Adapter                                 | [84300.301]           |       1 |       1 | [11KB]         |          |         |       0 | 48629179.77
  2 |   ->  Vector Aggregate                          | [84300.280]           |       1 |       1 | [18KB]         |          |         |       0 | 48629179.77
  3 |     ->  Vector Streaming (type: GATHER)         | [84300.186]           |       4 |       4 | [88KB]         |          |         |       0 | 48629179.77
  4 |       ->  Vector Aggregate                      | [165575.384,184252.368] |       4 |       4 | [140KB, 140KB] | 1MB      |         |       0 | 48629179.61
  5 |         ->  Vector Nest Loop (6,7)              | [162918.848,181438.162] | 2880404 | 2880404 | [74KB, 74KB]   | 1MB      |         |       0 | 48627379.35
  6 |           ->  CStore Scan on store_sales ss     | [15.660,16.229]       | 2880404 | 2880404 | [490KB, 490KB] | 1MB      |         |       4 | 16683.10
  7 |           ->  Vector Materialize                | [118314.521,132478.454] | 12968211302 |   18000 | [869KB, 900KB] | 16MB     | [8,8]   |       4 | 3890.00
  8 |             ->  CStore Scan on item i           | [0.234,0.243]         |   18000 |   18000 | [476KB, 476KB] | 1MB      |         |       4 | 3867.50
(8 rows)
```

```
gaussdb =#  set enable_nestloop=off;
SET
gaussdb =#  set enable_mergejoin=off;
SET
gaussdb =# explain analyze select count(*) fpostgres=# ales ss, item i where ss.ss_item_sk = i.i_item_sk;
 id |                    operation                    |        A-time         | A-rows  | E-rows  |  Peak Memory   | E-memory | A-width | E-width | E-costs
----+-------------------------------------------------+-----------------------+---------+---------+----------------+----------+---------+---------+-------------
  1 | ->  Row Adapter                                 | 291.066               |       1 |       1 | 11KB           |          |         |       0 | 32308.66
  2 |   ->  Vector Aggregate                          | 291.052               |       1 |       1 | 18KB           |          |         |       0 | 32308.66
  3 |     ->  Vector Streaming (type: GATHER)         | 290.973               |       4 |       4 | 88KB           |          |         |       0 | 32308.66
  4 |       ->  Vector Aggregate                      | [220.792,234.532]     |       4 |       4 | [140KB, 140KB] | 1MB      |         |       0 | 32308.50
  5 |         ->  Vector Hash Join (6,7)              | [209.987,223.345]     | 2880404 | 2880404 | [236KB, 241KB] | 16MB     | [8,8]   |       0 | 30508.24
  6 |           ->  CStore Scan on store_sales ss     | [13.132,13.717]       | 2880404 | 2880404 | [490KB, 490KB] | 1MB      |         |       4 | 16683.10
  7 |           ->  CStore Scan on item i             | [0.214,0.246]         |   18000 |   18000 | [477KB, 477KB] | 1MB      |         |       4 | 3867.50
(7 rows)
```

示例3：通常情况下Agg选择HashAgg性能较好，如果大结果集选择了Sort +GroupAgg，则需要设置enable_sort=off，HashAgg耗时明显优于Sort+GroupAgg。

```
gaussdb =#  explain analyze select count(*) from store_sales group by ss_item_sk;
 id |                    operation                    |        A-time         | A-rows  | E-rows  |      Peak Memory      | E-memory | A-width | E-width | E-costs
----+-------------------------------------------------+-----------------------+---------+---------+-----------------------+----------+---------+---------+-----------
  1 | ->  Row Adapter                                 | 1977.385              |   18000 |   17644 | 20KB                  |          |         |       4 | 92875.24
  2 |   ->  Vector Streaming (type: GATHER)           | 1973.617              |   18000 |   17644 | 1946KB                |          |         |       4 | 92875.24
  3 |     ->  Vector Sort Aggregate                   | [1784.800,1883.243]   |   18000 |   17644 | [273KB, 273KB]        | 1MB      |         |       4 | 92186.02
  4 |       ->  Vector Sort                           | [1752.270,1848.357]   | 2880404 | 2880404 | [128466KB, 135135KB]  | 16MB     | [8,8]   |       4 | 88541.40
  5 |         ->  CStore Scan on store_sales          | [12.483,13.548]       | 2880404 | 2880404 | [490KB, 490KB]        | 1MB      |         |       4 | 16683.10
(5 rows)
```

```
gaussdb =#  set enable_sort=off;
SET
gaussdb =#  explain analyze select count(*) from store_sales group by ss_item_sk;
 id |                    operation                    |        A-time         | A-rows  | E-rows  |      Peak Memory      | E-memory | A-width | E-width | E-costs
----+-------------------------------------------------+-----------------------+---------+---------+-----------------------+----------+---------+---------+-----------
  1 | ->  Row Adapter                                 | 838.218               |   18000 |   17644 | 20KB                  |          |         |       4 | 21016.93
  2 |   ->  Vector Streaming (type: GATHER)           | 834.264               |   18000 |   17644 | 228KB                 |          |         |       4 | 21016.93
  3 |     ->  Vector Hash Aggregate                   | [585.017,758.204]     |   18000 |   17644 | [262552KB, 262564KB]  | 16MB     | [8,8]   |       4 | 20327.72
  4 |       ->  CStore Scan on store_sales            | [12.540,13.941]       | 2880404 | 2880404 | [490KB, 490KB]        | 1MB      |         |       4 | 16683.10
(4 rows)
```

3.3.6.6 数据倾斜调优

数据倾斜问题是分布式架构的重要难题，它破坏了MPP架构中各个节点对等的要求，导致单节点（倾斜节点）所存储或者计算的数据量远大于其他节点，所以会造成以下危害：

- 存储上的倾斜会严重限制系统容量，在系统容量不饱和的情况下，由于单节点倾斜的限制，使得整个系统容量无法继续增长。
- 计算上的倾斜会严重影响系统性能，由于倾斜节点所需要运算的数据量远大于其它节点，导致倾斜节点降低系统整体性能。
- 数据倾斜还严重影响了MPP架构的扩展性。由于在存储或者计算时，往往会将相同值的数据放到同一节点，因此当倾斜数据（大量数据的值相同）出现之后，即使增加节点，系统瓶颈仍然受限于倾斜节点的容量或者性能。

GaussDB数据库针对数据倾斜问题给出了完整的解决方案，包括存储倾斜和计算倾斜两大问题，下面分别进行介绍。

存储层数据倾斜

GaussDB数据库中，数据分布存储在各个DN上，通过分布式执行提高查询的效率。但是，如果数据分布存在倾斜，则会导致分布式执行某些DN成为瓶颈，影响查询性能。这种情况通常是由于分布列选择不合理，可以通过调整分布列的方式解决。

例如下例：

```
gaussdb=# explain performance select count(*) from inventory;
5 --CStore Scan on lmz.inventory
        dn_6001_6002 (actual time=0.444..83.127 rows=42000000 loops=1)
        dn_6003_6004 (actual time=0.512..63.554 rows=27000000 loops=1)
```

```
      dn_6005_6006 (actual time=0.722..99.033 rows=45000000 loops=1)
      dn_6007_6008 (actual time=0.529..100.379 rows=51000000 loops=1)
      dn_6009_6010 (actual time=0.382..71.341 rows=36000000 loops=1)
      dn_6011_6012 (actual time=0.547..100.274 rows=51000000 loops=1)
      dn_6013_6014 (actual time=0.596..118.289 rows=60000000 loops=1)
      dn_6015_6016 (actual time=1.057..132.346 rows=63000000 loops=1)
      dn_6017_6018 (actual time=0.940..110.310 rows=54000000 loops=1)
      dn_6019_6020 (actual time=0.231..41.198 rows=21000000 loops=1)
      dn_6021_6022 (actual time=0.927..114.538 rows=54000000 loops=1)
      dn_6023_6024 (actual time=0.637..118.385 rows=60000000 loops=1)
      dn_6025_6026 (actual time=0.288..32.240 rows=15000000 loops=1)
      dn_6027_6028 (actual time=0.566..118.096 rows=60000000 loops=1)
      dn_6029_6030 (actual time=0.423..82.913 rows=42000000 loops=1)
      dn_6031_6032 (actual time=0.395..78.103 rows=39000000 loops=1)
      dn_6033_6034 (actual time=0.376..51.052 rows=24000000 loops=1)
      dn_6035_6036 (actual time=0.569..79.463 rows=39000000 loops=1)
```

在performance信息中，可以看到inventory表各DN的scan行数，发现各DN的行数差距较大，最大的为63000000，最小的只有15000000，差了4倍。这个差距对于数据扫描的性能影响还可以接受，但如果上层有join算子，则影响较大。

通常，数据表在各DN上是hash分布的，因此分布列的选择很重要。通过table_skewness()来查看上述inventory表在各DN的数据分布倾斜，查询结果如下：

```
gaussdb=# select table_skewness('inventory');
             table_skewness
--------------------------------------------
 ("dn_6015_6016       ",63000000,8.046%)
 ("dn_6013_6014       ",60000000,7.663%)
 ("dn_6023_6024       ",60000000,7.663%)
 ("dn_6027_6028       ",60000000,7.663%)
 ("dn_6017_6018       ",54000000,6.897%)
 ("dn_6021_6022       ",54000000,6.897%)
 ("dn_6007_6008       ",51000000,6.513%)
 ("dn_6011_6012       ",51000000,6.513%)
 ("dn_6005_6006       ",45000000,5.747%)
 ("dn_6001_6002       ",42000000,5.364%)
 ("dn_6029_6030       ",42000000,5.364%)
 ("dn_6031_6032       ",39000000,4.981%)
 ("dn_6035_6036       ",39000000,4.981%)
 ("dn_6009_6010       ",36000000,4.598%)
 ("dn_6003_6004       ",27000000,3.448%)
 ("dn_6033_6034       ",24000000,3.065%)
 ("dn_6019_6020       ",21000000,2.682%)
 ("dn_6025_6026       ",15000000,1.916%)
(18 rows)
```

通过查询建表定义，可以发现，目前该表是以inv_date_sk作为分布列的，导致存在倾斜。通过查看各列的数据分布情况，改为inv_item_sk作为分布列，则倾斜情况分布如下：

```
gaussdb=# select table_skewness('inventory');
             table_skewness
--------------------------------------------
 ("dn_6001_6002       ",43934200,5.611%)
 ("dn_6007_6008       ",43829420,5.598%)
 ("dn_6003_6004       ",43781960,5.592%)
 ("dn_6031_6032       ",43773880,5.591%)
 ("dn_6033_6034       ",43763280,5.589%)
 ("dn_6011_6012       ",43683600,5.579%)
 ("dn_6013_6014       ",43551660,5.562%)
 ("dn_6027_6028       ",43546340,5.561%)
 ("dn_6009_6010       ",43508700,5.557%)
 ("dn_6023_6024       ",43484540,5.554%)
 ("dn_6019_6020       ",43466800,5.551%)
 ("dn_6021_6022       ",43458500,5.550%)
 ("dn_6017_6018       ",43448040,5.549%)
 ("dn_6015_6016       ",43247700,5.523%)
```

```
 ("dn_6005_6006        ",43200240,5.517%)
 ("dn_6029_6030        ",43181360,5.515%)
 ("dn_6025_6026        ",43179700,5.515%)
 ("dn_6035_6036        ",42960080,5.487%)
(18 rows)
```

数据分布倾斜的问题得到解决。

计算层数据倾斜

即使通过修改表的分布键，使得数据存储在各个节点上是均衡的，但是在执行查询的过程中，仍然可能出现数据倾斜的问题。在运算过程中某个算子在DN上输出的结果集出现倾斜，从而导致此算子上层的运算出现计算倾斜。一般来说，这是由于在执行过程中，数据重分布导致的。

在查询执行的过程中，join key、group by key等往往不是表的分布列，因此需要按照join key、group by key上数据的hash值，让数据在各个DN之间进行重新分布，这个过程对应于计划中的Redistribute算子。当重分布列上的数据存在倾斜时，就会导致运行时的数据倾斜，即重分布后部分节点的数据远远大于其他。倾斜节点需要处理更多的数据，导致倾斜节点的计算性能远远低于其他节点。

如下例中，s表和t表join，join条件中的s.x和t.x均不是表的分布列，因此需要重分布（REDISTRIBUTE算子）。其中s.x列上存在倾斜值，t.x上不存在倾斜。id=6的stream算子在datanode2节点输出的结果集是其他DN的3倍，从而导致了计算倾斜。

```
select * from skew s,test t where s.x = t.x order by s.a limit 1;
 id |                    operation                    |        A-time
----+-------------------------------------------------+----------------------
  1 | ->  Limit                                       | 52622.382
  2 |    ->  Streaming (type: GATHER)                 | 52622.374
  3 |       ->  Limit                                 | [30138.494,52598.994]
  4 |          ->  Sort                               | [30138.486,52598.986]
  5 |             ->  Hash Join (6,8)                 | [30127.013,41483.275]
  6 |                ->  Streaming(type: REDISTRIBUTE)    | [11365.110,22024.845]
  7 |                   ->  Seq Scan on public.skew s     | [2019.168,2175.369]
  8 |                ->  Hash                         | [2460.108,2499.850]
  9 |                   ->  Streaming(type: REDISTRIBUTE) | [1056.214,1121.887]
 10 |                      ->  Seq Scan on public.test t  | [310.848,325.569]
(10 rows)
 6 --Streaming(type: REDISTRIBUTE)
        datanode1 (rows=5050368)
        datanode2 (rows=15276032)
        datanode3 (rows=5174272)
        datanode4 (rows=5219328)
```

和存储倾斜相比，计算倾斜更难以提前识别，因此GaussDB提出了RLBT(Runtime Load Balance Technology)方案，用以解决运行时的计算倾斜问题。RLBT方案主要分为两个层面，第一步是计算倾斜识别，第二步是计算倾斜解决。下面分别进行介绍。

1. 倾斜识别

   计算倾斜的识别，即预先识别计算过程中的重分布列是否存在倾斜数据。RLBT方案中给出了三个解决手段，统计信息识别，hint方式指定以及规则识别：

   - 统计信息识别

     需要用户先执行analyze收集各表的统计信息，然后优化器能够自动利用统计信息对重分布键上的倾斜数据进行提前识别，对于存在倾斜的查询，生成相应的优化计划。在重分布键有多列的情况，只有所有列都属于同一个基表才能利用统计信息进行识别。

     统计信息只能给出基表的倾斜情况，当基表某一列存在倾斜，其他列上带有过滤条件，或者经过和其他表的join之后，无法准确判断倾斜列上倾斜数据是否依旧存在。

- hint方式指定

  统计信息有着一定的局限性，对于较为复杂的查询，其中间结果难以通过统计信息进行估算和识别倾斜数据。对于这种情况，设计了hint手段，通过用户手动指定的方式，给定倾斜信息。优化器根据用户给定的倾斜信息，来对查询进行优化。详细hint使用语法参见**运行倾斜的hint**。

- 规则识别

  现在BI系统往往会产生大量带有outer join（left join、right join、full join）的SQL，outer join在匹配失败的情况下会补空产生大量NULL值，如果接下来在补空列上进行join或者group by操作，就会导致NULL值倾斜。当前RLBT技术会自动识别这种场景，并生成相应的NULL值倾斜优化计划。

2. 计算倾斜解决

   在解决倾斜时，目前针对最常见的join和agg算子进行了优化。

   - join优化

   基本思路是将倾斜数据和非倾斜数据进行隔离处理。主要分为以下三种情况：

   a. join两侧都需要做重分布：

      对倾斜侧做PART_REDISTRIBUTE_PART_ROUNDROBIN，其中对倾斜数据做roundrobin，非倾斜数据做redistribute；

      对非倾斜侧做PART_REDISTRIBUTE_PART_BROADCAST，其中对倾斜数据做broadcast，非倾斜数据做redistribute；

   b. join一侧需要重分布，另一侧不需要重分布：

      对需要重分布的一侧做PART_REDISTRIBUTE_PART_ROUNDROBIN；

      对不需要重分布的一侧做PART_LOCAL_PART_BROADCAST，其中对等于倾斜值的部分做broadcast，其余数据保留在本地。

   c. 对于有补NULL值的表：

      对该表做PART_REDISTERIBUTE_PART_LOCAL，其中将NULL值保留在本地，其余数据做redistribute。

   以前面的查询为例，s.x列上存在倾斜数据，倾斜数据的值为0。优化器通过统计信息，识别到了该倾斜数据，生成了倾斜优化计划如下：

```
 id |                                operation                                 |        A-time
----+--------------------------------------------------------------------------+----------------------
  1 | ->  Limit                                                                | 23642.049
  2 |    ->  Streaming (type: GATHER)                                          | 23642.041
  3 |       ->  Limit                                                          | [23310.768,23618.021]
  4 |          ->  Sort                                                        | [23310.761,23618.012]
  5 |             ->  Hash Join (6,8)                                          | [20898.341,21115.272]
  6 |                ->  Streaming(type: PART REDISTRIBUTE PART ROUNDROBIN)    |
[7125.834,7472.111]
  7 |                   ->  Seq Scan on public.skew s                          | [1837.079,1911.025]
  8 |                ->  Hash                                                  | [2612.484,2640.572]
  9 |                   ->  Streaming(type: PART REDISTRIBUTE PART BROADCAST)  | [1193.548,1297.894]
 10 |                      ->  Seq Scan on public.test t                       | [314.343,328.707]
(10 rows)
  5 --Vector Hash Join (6,8)
      Hash Cond: s.x = t.x
      Skew Join Optimized by Statistic
  6 --Streaming(type: PART REDISTRIBUTE PART ROUNDROBIN)
      datanode1 (rows=7635968)
      datanode2 (rows=7517184)
      datanode3 (rows=7748608)
      datanode4 (rows=7818240)
```

   上述执行计划中，可以看到Skew Join Optimized by Statistic的字样，代表该计划为倾斜优化计划，其中Statistic关键字代表该倾斜优化来自于统计信息，除此之外还有Hint和Rule，分别代表倾斜优化来自于hint语句和规则。对比前面的计划可以

看到，这里对于非倾斜数据和倾斜数据做了分别处理。对于s表中的非倾斜数据，依旧按照原有的方案，根据数据的hash值进行重分布；而对于倾斜数据（即等于0的数据），则通过轮询发送的方式，均衡地发送到所有节点。通过这样的方式，解决了倾斜数据分布不均衡的问题。

同时，为了保证结果的正确性，需要对t表做相应的处理。对于t表中等于0（s.x表中的倾斜值）的数据做广播，对于其他数据，依旧根据数据的hash值进行重分布。

通过这样的方式，就解决了join操作中，数据倾斜的问题。从上面的结果来看，id=6的stream算子各个DN的输出结果已经非常均衡，同时查询端到端性能提升了1倍。

- agg优化

对于agg操作，解决倾斜的思路与join操作不同，这里是通过首先在本DN内按照group by key进行去重操作，然后再进行重分布。因为经过DN内部去重之后，从全局来看，每个值的数量都不会超过DN数，因此不会出现严重的数据倾斜问题。以如下query为例：

select c1, c2, c3, c4, c5, c6, c7, c8, c9, count(*) from t group by c1, c2, c3, c4, c5, c6, c7, c8, c9 limit 10;

原执行结果如下：

```
id |                operation                |         A-time         | A-rows    ----
+-----------------------------------------+------------------------+----------    1 | ->  Streaming (type:
GATHER)             | 130621.783             |      12     2 |    ->  GroupAggregate                  |
[85499.711,130432.341] |      12     3 |      ->  Sort                          | [85499.509,103145.632] |
36679237     4 |         ->  Streaming(type: REDISTRIBUTE) | [25668.897,85499.050]  | 36679237     5
|           ->  Seq Scan on public.t      | [9835.069,10416.388]   | 36679237    (5 rows)       4 --
Streaming(type: REDISTRIBUTE)          datanode1 (rows=36678837)          datanode2
(rows=100)          datanode3 (rows=100)          datanode4 (rows=200)
```

其中存在大量倾斜数据，导致数据按照group by key进行重分布之后，datanode1的数据量是其他节点的数十万倍。在倾斜优化之后，首先在本DN进行一次group by操作，达到数据去重的效果，然后再进行重分布，可以发现基本没有数据倾斜的问题出现。

```
id |                operation                |         A-time          ----+-----------------------------------------------
+----------------------    1 | ->  Streaming (type: GATHER)             | 10961.337               2 |    ->
HashAggregate                  | [10953.014,10953.705]     3 |      ->  HashAggregate                 |
[10952.957,10953.632]     4 |         ->  Streaming(type: REDISTRIBUTE) | [10952.859,10953.502]     5
|           ->  HashAggregate            | [10084.280,10947.139]     6 |              ->  Seq Scan on
public.t    | [4757.031,5201.168]     (6 rows)     Predicate Information (identified by plan id)
-----------------------------------------------     3 --HashAggregate          Skew Agg Optimized by
Statistic  (2 rows)       4 --Streaming(type: REDISTRIBUTE)          datanode1 (rows=17)
datanode2 (rows=8)          datanode3 (rows=8)          datanode4 (rows=14)
```

适用范围

- join算子
  - 支持nest loop，merge join，hash join等join方式；
  - 当倾斜数据处于join的left侧时，支持inner join，left join，semi join，anti join；当倾斜属于位于join的right侧时，支持inner join，right join，right semi join，right anti join。
  - 通过统计信息得到的倾斜优化计划，优化器会根据代价判断该计划是否为最优计划。通过hint和规则会强制生成倾斜优化计划。
- agg算子
  - array_agg、string_agg、subplan in agg qual这几种场景不支持优化；

- 通过统计信息识别到的倾斜优化计划会受到代价、plan_mode_seed参数、best_agg_plan参数影响，而hint、规则识别到的不会。

3.3.7 经验总结：SQL 语句改写规则

根据数据库的SQL执行机制以及大量的实践，总结发现：通过一定的规则调整SQL语句，在保证结果正确的基础上，能够提高SQL执行效率。如果遵守这些规则，常常能够大幅度提升业务查询效率。

- **使用union all代替union**

  union在合并两个集合时会执行去重操作，而union all则直接将两个结果集合并、不执行去重。执行去重会消耗大量的时间，因此，在一些实际应用场景中，如果通过业务逻辑已确认两个集合不存在重叠，可用union all替代union以便提升性能。

- **join列增加非空过滤条件**

  若join列上的NULL值较多，则可以加上is not null过滤条件，以实现数据的提前过滤，提高join效率。

- **not in转not exists**

  not in语句需要使用nestloop anti join来实现，而not exists则可以通过hash anti join来实现。在join列不存在null值的情况下，not exists和not in等价。因此在确保没有null值时，可以通过将not in转换为not exists，通过生成hash join来提升查询效率。

  如下所示，如果t2.d2字段中没有null值(t2.d2字段在表定义中not null)查询可以修改为

  SELECT * FROM t1 WHERE NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);

  产生的计划如下：

  图 3-9 not exists 执行计划

```
 id |                 operation
----+-------------------------------------------
  1 | ->  Streaming (type: GATHER)
  2 |    ->  Hash Anti Join (3, 4)
  3 |       ->  Seq Scan on t1
  4 |       ->  Hash
  5 |          ->  Streaming(type: REDISTRIBUTE)
  6 |             ->  Seq Scan on t2
(6 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
   2 --Hash Anti Join (3, 4)
         Hash Cond: (t1.c1 = t2.d2)
(2 rows)
```

- **选择hashagg。**

  查询中GROUP BY语句如果生成了groupagg+sort的plan性能会比较差，可以通过加大work_mem的方法生成hashagg的plan，因为不用排序而提高性能。

- **尝试将函数替换为case语句。**

GaussDB函数调用性能较低，如果出现过多的函数调用导致性能下降很多，可以根据情况把可下推函数的函数改成CASE表达式。

- **避免对索引使用函数或表达式运算。**

  对索引使用函数或表达式运算会停止使用索引转而执行全表扫描。

- **尽量避免在where子句中使用!=或<>操作符、null值判断、or连接、参数隐式转换。**
- **对复杂SQL语句进行拆分。**

  对于过于复杂并且不易通过以上方法调整性能的SQL可以考虑拆分的方法，把SQL中某一部分拆分成独立的SQL并把执行结果存入临时表，拆分常见的场景包括但不限于：

  - 作业中多个SQL有同样的子查询，并且子查询数据量较大。
  - Plan cost计算不准，导致子查询hash bucket太小，比如实际数据1000W行，hash bucket只有1000。
  - 函数（如substr,to_number）导致大数据量子查询选择度计算不准。
  - 多DN环境下对大表做broadcast的子查询。

3.3.8 SQL 调优关键参数调整

本节将介绍影响GaussDB SQL调优性能的关键CN配置参数。

表 3-2 CN 配置参数

参数/参考值	描述
enable_nestloop=on	控制查询优化器对嵌套循环连接（Nest Loop Join）类型的使用。当设置为“on”后，优化器优先使用Nest Loop Join；当设置为“off”后，优化器在存在其他方法时将优先选择其他方法。 **说明** 如果只需要在当前数据库连接（即当前Session）中临时更改该参数值，则只需要在SQL语句中执行如下命令： SET enable_nestloop to off; 实际调优中应根据情况选择是否关闭。一般情况下，在三种join方式（Nested Loop、Merge Join和Hash Join）里，Nested Loop适合小数据量或者有索引的场景，Hash Join适合大数据分析场景。
enable_bitmapscan=on	控制查询优化器对位图扫描规划类型的使用。设置为“on”，表示使用；设置为“off”，表示不使用。 **说明** 如果只需要在当前数据库连接（即当前Session）中临时更改该参数值，则只需要在SQL语句中执行命令如下命令： SET enable_bitmapscan to off; bitmapscan扫描方式适用于“where a > 1 and b > 1”且a列和b列都有索引这种查询条件，但有时其性能不如indexscan。因此，现场调优如发现查询性能较差且计划中有bitmapscan算子，可以关闭bitmapscan，看性能是否有提升。

参数/参考值	描述
enable_fast_query_shipping=on	控制查询优化器是否使用分布式框架，执行快速执行计划。设置为“on”，表示执行计划在CN和DN上各自生成；设置为“off”，表示使用分布式框架，即执行计划在CN上生成，然后发送到DN中执行。 **说明** 如果只需要在当前数据库连接（即当前Session）中临时更改该参数值，则只需要在SQL语句中执行如下命令： SET enable_fast_query_shipping to off;
enable_hashagg=on	控制优化器对Hash聚集规划类型的使用。
enable_hashjoin=on	控制优化器对Hash连接规划类型的使用。
enable_mergejoin=on	控制优化器对融合连接规划类型的使用。
enable_indexscan=on	控制优化器对索引扫描规划类型的使用。
enable_indexonlyscan=on	控制优化器对仅索引扫描规划类型的使用。
enable_seqscan=on	控制优化器对顺序扫描规划类型的使用。完全消除顺序扫描是不可能的，但是关闭这个变量会让优化器在存在其他方法的时候优先选择其他方法。
enable_sort=on	控制优化器使用的排序步骤。该设置不可能完全消除明确的排序，但是关闭这个变量可以让优化器在存在其他方法的时候优先选择其他方法。
enable_broadcast=on	控制查询优化器对于broadcast广播模式数据传输的使用。此方式网络传输数据量较大，因此当网络传输节点（Stream）实际数据量较大而估算不准时，可以将该参数设置为off，看性能是否有提升。
rewrite_rule	控制优化器是否启用LAZYAGG\MAGICSET\PARTIALPUSH\UNIQUECHECK\DISABLEREP\INTARGETLIST\PREDPUSH重写规则。

3.3.9 使用 Plan Hint 进行调优

3.3.9.1 Plan Hint 调优概述

Plan Hint为用户提供了直接影响执行计划生成的手段，用户可以通过指定join顺序，join、stream、scan方法，指定结果行数，指定重分布过程中的倾斜信息等多个手段来进行执行计划的调优，以提升查询的性能。

功能描述

Plan Hint仅支持在SELECT关键字后通过如下形式指定：

```
/*+ <plan hint>*/
```

可以同时指定多个hint，之间使用空格分隔。hint只能hint当前层的计划，对于子查询计划的hint，需要在子查询的select关键字后指定hint。

例如：

```
select /*+ <plan_hint1> <plan_hint2> */ * from t1, (select /*+ <plan_hint3> */ from t2) where 1=1;
```

其中<plan_hint1>，<plan_hint2>为外层查询的hint，<plan_hint3>为内层子查询的hint。

须知

如果在视图定义（CREATE VIEW）时指定hint，则在该视图每次被应用时会使用该hint。

说明

当使用random plan功能（参数plan_mode_seed不为0）时，查询指定的plan hint不会被使用。

支持范围

当前版本Plan Hint支持的范围如下，后续版本会进行增强。

- 指定Join顺序的Hint - leading hint。
- 指定Join方式的Hint，仅支持除semi/anti join，unique plan之外的常用hint。
- 指定结果集行数的Hint。
- 指定Stream方式的Hint。
- 指定Scan方式的Hint，仅支持常用的tablescan，indexscan和indexonlyscan的hint。
- 指定子链接块名的Hint。
- 指定倾斜信息的Hint，仅支持Join与HashAgg的重分布过程倾斜。

注意事项

- 不支持Agg、Sort、Setop和Subplan的hint。
- 不支持SMP和Node Group场景下的Hint。

示例

本章节使用同一个语句进行示例，便于Plan Hint支持的各方法作对比，示例语句及不带hint的原计划如下所示：

```
explain
select i_product_name product_name
,i_item_sk item_sk
,s_store_name store_name
,s_zip store_zip
,ad2.ca_street_number c_street_number
,ad2.ca_street_name c_street_name
,ad2.ca_city c_city
,ad2.ca_zip c_zip
```



```
,count(*) cnt
,sum(ss_wholesale_cost) s1
,sum(ss_list_price) s2
,sum(ss_coupon_amt) s3
FROM   store_sales
,store_returns
,store
,customer
,promotion
,customer_address ad2
,item
WHERE  ss_store_sk = s_store_sk AND
ss_customer_sk = c_customer_sk AND
ss_item_sk = i_item_sk and
ss_item_sk = sr_item_sk and
ss_ticket_number = sr_ticket_number and
c_current_addr_sk = ad2.ca_address_sk and
ss_promo_sk = p_promo_sk and
i_color in ('maroon','burnished','dim','steel','navajo','chocolate') and
i_current_price between 35 and 35 + 10 and
i_current_price between 35 + 1 and 35 + 15
group by i_product_name
,i_item_sk
,s_store_name
,s_zip
,ad2.ca_street_number
,ad2.ca_street_name
,ad2.ca_city
,ad2.ca_zip
;
```

```
 id |                                operation                                 |   E-rows   | E-memory | E-width |  E-costs
----+--------------------------------------------------------------------------+------------+----------+---------+------------
  1 | ->  Row Adapter                                                          |          6 |          |     273 | 3401632.49
  2 |    ->  Vector Streaming (type: GATHER)                                   |          6 |          |     273 | 3401632.49
  3 |       ->  Vector Hash Aggregate                                          |          6 | 16MB     |     273 | 3401630.82
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)                        |          6 | 1MB      |     169 | 3401630.78
  5 |             ->  Vector Hash Join (6,21)                                  |          6 | 16MB     |     169 | 3401630.42
  6 |                ->  Vector Hash Join (7,20)                               |          7 | 43MB     |     173 | 3400343.15
  7 |                   ->  Vector Streaming(type: REDISTRIBUTE)               |          7 | 1MB      |     123 | 3395775.64
  8 |                      ->  Vector Hash Join (9,19)                         |          7 | 27MB     |     123 | 3395775.48
  9 |                         ->  Vector Streaming(type: REDISTRIBUTE)         |          7 | 1MB      |     123 | 3386294.72
 10 |                            ->  Vector Hash Join (11,18)                  |          7 | 16MB     |     123 | 3386294.56
 11 |                               ->  Vector Hash Join (12,14)               |          7 | 19MB     |     112 | 3384018.02
 12 |                                  ->  Vector Partition Iterator           |  287999764 | 1MB      |      12 | 227383.99
 13 |                                     ->  Partitioned CStore Scan on store_returns | 287999764 | 1MB |      12 | 227383.99
 14 |                                  ->  Vector Hash Join (15,17)            |    1516824 | 16MB     |     124 | 3065686.08
 15 |                                     ->  Vector Partition Iterator        | 2879987999 | 1MB      |      66 | 2756066.50
 16 |                                        ->  Partitioned CStore Scan on store_sales | 2879987999 | 1MB |      66 | 2756066.50
 17 |                                     ->  CStore Scan on item              |        158 | 1MB      |      58 | 4051.25
 18 |                                  ->  CStore Scan on store                |      24048 | 1MB      |      19 | 2264.00
 19 |                               ->  CStore Scan on customer                |   12000000 | 1MB      |       8 | 12923.00
 20 |                            ->  CStore Scan on customer_address ad2       |    6000000 | 1MB      |      58 | 5770.00
 21 |                         ->  CStore Scan on promotion                     |      36000 | 1MB      |       4 | 1268.50
(21 rows)
```

3.3.9.2 Join 顺序的 Hint

功能描述

指明join的顺序，包括内外表顺序。

语法格式

- 仅指定join顺序，不指定内外表顺序。

```
leading(join_table_list)
```

- 同时指定join顺序和内外表顺序，内外表顺序仅在最外层生效。

```
leading((join_table_list))
```

参数说明

join_table_list为表示表join顺序的hint字符串，可以包含当前层的任意个表（别名），或对于子查询提升的场景，也可以包含子查询的hint别名，同时任意表可以使用括号指定优先级，表之间使用空格分隔。

须知

表只能用单个字符串表示，不能带schema。

说明

表如果存在别名，需要优先使用别名来表示该表。

join table list中指定的表需要满足以下要求，否则会报语义错误。

- list中的表必须在当前层或提升的子查询中存在。
- list中的表在当前层或提升的子查询中必须是唯一的。如果不唯一，需要使用不同的别名进行区分。
- 同一个表只能在list里出现一次。
- 如果表存在别名，则list中的表需要使用别名。

例如：

leading(t1 t2 t3 t4 t5)表示：t1, t2, t3, t4, t5先join，五表join顺序及内外表不限。

leading((t1 t2 t3 t4 t5))表示：t1和t2先join，t2做内表；再和t3 join，t3做内表；再和t4 join，t4做内表；再和t5 join，t5做内表。

leading(t1 (t2 t3 t4) t5)表示：t2, t3, t4先join，内外表不限；再和t1, t5 join，内外表不限。

leading((t1 (t2 t3 t4) t5))表示：t2, t3, t4先join，内外表不限；在最外层，t1再和t2, t3, t4的join表join，t1为外表，再和t5 join，t5为内表。

leading((t1 (t2 t3) t4 t5)) leading((t3 t2))表示：t2，t3先做join，t2做内表；然后再和t1做join，t2，t3的join表做内表；然后再跟t4做join，t4做内表，最后和t5做join，t5做内表。

示例

对**示例**中原语句使用如下hint:

```
explain
select /*+ leading((((((store_sales store) promotion) item) customer) ad2) store_returns) leading((store store_sales))*/ i_product_name product_name ...
```

该hint表示：表之间的join关系是：store_sales和store先join， store_sales做内表，然后依次跟promotion, item, customer, ad2, store_returns做join。生成计划如下所示：

```
WARNING:  Duplicated or conflict hint: Leading(store_sales store), will be discarded.
 id |                                  operation                                  |   E-rows   | E-memory | E-width |   E-costs
----+-----------------------------------------------------------------------------+------------+----------+---------+-------------
  1 | ->  Row Adapter                                                             |          6 |          |     273 | 16308094.34
  2 |    ->  Vector Streaming (type: GATHER)                                      |          6 |          |     273 | 16308094.34
  3 |       ->  Vector Hash Aggregate                                             |          6 | 16MB     |     273 | 16308092.67
  4 |          ->  Vector Hash Join (5,20)                                        |          6 | 585MB    |     169 | 16308092.63
  5 |             ->  Vector Streaming(type: REDISTRIBUTE)                        |    1320811 | 1MB      |     181 | 16069870.93
  6 |                ->  Vector Hash Join (7,19)                                  |    1320811 | 43MB     |     181 | 16061891.00
  7 |                   ->  Vector Streaming(type: REDISTRIBUTE)                  |    1320811 | 1MB      |     131 | 16056566.78
  8 |                      ->  Vector Hash Join (9,18)                            |    1320811 | 27MB     |     131 | 16048586.85
  9 |                         ->  Vector Streaming(type: REDISTRIBUTE)            |    1383248 | 1MB      |     131 | 16038321.62
 10 |                            ->  Vector Hash Join (11,17)                     |    1383248 | 16MB     |     131 | 16029664.50
 11 |                               ->  Vector Hash Join (12,16)                  | 2626366951 | 16MB     |      73 | 15751384.88
 12 |                                  ->  Vector Hash Join (13,14)               | 2750085660 | 2156MB   |      77 | 14226077.19
 13 |                                     ->  CStore Scan on store                |      24048 | 1MB      |      19 | 2264.00
 14 |                                     ->  Vector Partition Iterator           | 2879987999 | 1MB      |      66 | 2756066.50
 15 |                                        ->  Partitioned CStore Scan on store_sales | 2879987999 | 1MB |      66 | 2756066.50
 16 |                                  ->  CStore Scan on promotion               |      36000 | 1MB      |       4 | 1268.50
 17 |                               ->  CStore Scan on item                       |        158 | 1MB      |      58 | 4051.25
 18 |                         ->  CStore Scan on customer                         |   12000000 | 1MB      |       8 | 12923.00
 19 |                      ->  CStore Scan on customer_address ad2                |    6000000 | 1MB      |      58 | 5770.00
 20 |                ->  Vector Partition Iterator                                | 287999764 | 1MB      |      12 | 227383.99
 21 |                   ->  Partitioned CStore Scan on store_returns              | 287999764 | 1MB      |      12 | 227383.99
(21 rows)
```

图中计划顶端warning的提示详见**Hint的错误、冲突及告警**的说明。

3.3.9.3 Join 方式的 Hint

功能描述

指明Join使用的方法，可以为Nested Loop，Hash Join和Merge Join。

语法格式

```
[no] nestloop|hashjoin|mergejoin(table_list)
```

参数说明

- **no**表示hint的join方式不使用。
- **table_list**为表示hint表集合的字符串，该字符串中的表与**join_table_list**相同，只是中间不允许出现括号指定join的优先级。

例如：

no nestloop(t1 t2 t3)表示：生成t1，t2，t3三表连接计划时，不使用nestloop。三表连接计划可能是t2 t3先join，再跟t1 join，或t1 t2先join，再跟t3 join。此hint只hint最后一次join的join方式，对于两表连接的方法不hint。如果需要，可以单独指定，例如：任意表均不允许nestloop连接，且希望t2 t3先join，则增加hint：no nestloop(t2 t3)。

示例

对**示例**中原语句使用如下hint:

```
explain
select /*+ nestloop(store_sales store_returns item) */ i_product_name product_name ...
```

该hint表示：生成store_sales，store_returns和item三表的结果集时，最后的两表关联使用nestloop。生成计划如下所示：


```
 id |                                operation                                 |   E-rows   | E-memory | E-width |     E-costs
----+--------------------------------------------------------------------------+------------+----------+---------+-----------------
  1 | ->  Row Adapter                                                          |          6 |          |     273 | 100061693161.06
  2 |    ->  Vector Streaming (type: GATHER)                                   |          6 |          |     273 | 100061693161.06
  3 |       ->  Vector Hash Aggregate                                          |          6 | 16MB     |     273 | 100061693159.40
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)                        |          6 | 1MB      |     169 | 100061693159.36
  5 |             ->  Vector Hash Join (6,22)                                  |          6 | 43MB     |     169 | 100061693158.99
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                  |          6 | 1MB      |     119 | 100061688591.48
  7 |                   ->  Vector Hash Join (8,21)                            |          6 | 16MB     |     119 | 100061688591.30
  8 |                      ->  Vector Hash Join (9,20)                         |          7 | 27MB     |     123 | 100061687304.04
  9 |                         ->  Vector Streaming(type: REDISTRIBUTE)         |          7 | 1MB      |     123 | 100061677823.27
 10 |                            ->  Vector Hash Join (11,19)                  |          7 | 16MB     |     123 | 100061677823.12
 11 |                               ->  Vector Nest Loop (12,17)               |          7 | 1MB      |     112 | 100061675546.57
 12 |                                  ->  Vector Hash Join (13,15)            |      13670 | 585MB    |      62 | 6163443.54
 13 |                                     ->  Vector Partition Iterator        | 2879987999 | 1MB      |      66 | 2756066.50
 14 |                                        ->  Partitioned CStore Scan on store_sales | 2879987999 | 1MB | 66 | 2756066.50
 15 |                                     ->  Vector Partition Iterator        |  287999764 | 1MB      |      12 | 227383.99
 16 |                                        ->  Partitioned CStore Scan on store_returns | 287999764 | 1MB | 12 | 227383.99
 17 |                                  ->  Vector Materialize                  |        158 | 16MB     |      58 | 4051.28
 18 |                                     ->  CStore Scan on item              |        158 | 1MB      |      58 | 4051.25
 19 |                               ->  CStore Scan on store                   |      24048 | 1MB      |      19 | 2264.00
 20 |                            ->  CStore Scan on customer                   |   12000000 | 1MB      |       8 | 12923.00
 21 |                         ->  CStore Scan on promotion                     |      36000 | 1MB      |       4 | 1268.50
 22 |                      ->  CStore Scan on customer_address ad2             |    6000000 | 1MB      |      58 | 5770.00
(22 rows)
```

3.3.9.4 行数的 Hint

功能描述

指明中间结果集的大小，支持绝对值和相对值的hint。

语法格式

```
rows(table_list #|+|-|* const)
```

参数说明

- **#,+,-,***，进行行数估算hint的四种操作符号。#表示直接使用后面的行数进行hint。+,-,*表示对原来估算的行数进行加、减、乘操作，运算后的行数最小值为1行。table_list为hint对应的单表或多表join结果集，与**Join方式的Hint**中table_list相同。
- **const**可以是任意非负数，支持科学计数法。

例如：

rows(t1 #5)表示：指定t1表的结果集为5行。

rows(t1 t2 t3 *1000)表示：指定t1, t2, t3 join完的结果集的行数乘以1000。

建议

- 推荐使用两个表*的hint。对于两个表的采用*操作符的hint，只要两个表出现在join的两端，都会触发hint。例如：设置hint为rows(t1 t2 * 3)，对于(t1 t3 t4)和(t2 t5 t6)join时，由于t1和t2出现在join的两端，所以其join的结果集也会应用该hint规则乘以3。
- rows hint支持在单表、多表、function table及subquery scan table的结果集上指定hint。

示例

对**示例**中原语句使用如下hint:

```
explain
select /*+ rows(store_sales store_returns *50) */ i_product_name product_name ...
```


该hint表示：store_sales，store_returns关联的结果集估算行数在原估算行数基础上乘以50。生成计划如下所示：

```
 id |                                operation                                 |   E-rows   | E-memory | E-width |  E-costs
----+--------------------------------------------------------------------------+------------+----------+---------+------------
  1 | ->  Row Adapter                                                          |        312 |          |     273 | 3401656.58
  2 |    ->  Vector Streaming (type: GATHER)                                   |        312 |          |     273 | 3401656.58
  3 |       ->  Vector Hash Aggregate                                          |        312 | 16MB     |     273 | 3401634.91
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)                        |        313 | 1MB      |     169 | 3401634.39
  5 |             ->  Vector Hash Join (6,21)                                  |        313 | 43MB     |     169 | 3401633.06
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                  |        313 | 1MB      |     119 | 3397065.38
  7 |                   ->  Vector Hash Join (8,20)                            |        313 | 27MB     |     119 | 3397064.31
  8 |                      ->  Vector Streaming(type: REDISTRIBUTE)            |        328 | 1MB      |     119 | 3387583.37
  9 |                         ->  Vector Hash Join (10,19)                     |        328 | 16MB     |     119 | 3387582.18
 10 |                            ->  Vector Hash Join (11,18)                  |        344 | 16MB     |     123 | 3386294.74
 11 |                               ->  Vector Hash Join (12,14)               |        360 | 19MB     |     112 | 3384018.02
 12 |                                  ->  Vector Partition Iterator           |  287999764 | 1MB      |      12 | 227383.99
 13 |                                     ->  Partitioned CStore Scan on store_returns | 287999764 | 1MB |      12 | 227383.99
 14 |                                  ->  Vector Hash Join (15,17)            |    1516824 | 16MB     |     124 | 3065686.08
 15 |                                     ->  Vector Partition Iterator        | 2879987999 | 1MB      |      66 | 2756066.50
 16 |                                        ->  Partitioned CStore Scan on store_sales | 2879987999 | 1MB |      66 | 2756066.50
 17 |                                     ->  CStore Scan on item              |        158 | 1MB      |      58 | 4051.25
 18 |                               ->  CStore Scan on store                   |      24048 | 1MB      |      19 | 2264.00
 19 |                            ->  CStore Scan on promotion                  |      36000 | 1MB      |       4 | 1268.50
 20 |                      ->  CStore Scan on customer                         |   12000000 | 1MB      |       8 | 12923.00
 21 |                ->  CStore Scan on customer_address ad2                   |    6000000 | 1MB      |      58 | 5770.00
(21 rows)
```

第11行算子的估算行数修正为360行，原估算行数为7行（四舍五入后取值）。

3.3.9.5 Stream 方式的 Hint

功能描述

指明stream使用的方法，可以为broadcast和redistribute。

语法格式

```
[no] broadcast|redistribute(table_list)
```

参数说明

- **no**表示hint的stream方式不使用。
- **table_list**为进行stream操作的单表或多表join结果集，见**参数说明**。

示例

对**示例**中原语句使用如下hint:

```
explain
select /*+ no redistribute(store_sales store_returns item store) leading(((store_sales store_returns item store) customer)) */ i_product_name product_name ...
```

原计划中，(store_sales store_returns item store)和customer做join时，前者做了重分布，此hint表示禁止前者混合表做重分布，但仍然保持join顺序，则生成计划如下所示：

```
 id |                                operation                                 |   E-rows   | E-memory | E-width |  E-costs
----+--------------------------------------------------------------------------+------------+----------+---------+------------
  1 | ->  Row Adapter                                                          |          6 |          |     273 | 5718448.94
  2 |    ->  Vector Streaming (type: GATHER)                                   |          6 |          |     273 | 5718448.94
  3 |       ->  Vector Hash Aggregate                                          |          6 | 16MB     |     273 | 5718447.27
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)                        |          6 | 1MB      |     169 | 5718447.23
  5 |             ->  Vector Hash Join (6,21)                                  |          6 | 16MB     |     169 | 5718446.86
  6 |                ->  Vector Hash Join (7,20)                               |          7 | 43MB     |     173 | 5717159.60
  7 |                   ->  Vector Streaming(type: REDISTRIBUTE)               |          7 | 1MB      |     123 | 5712592.09
  8 |                      ->  Vector Hash Join (9,18)                         |          7 | 585MB    |     123 | 5712591.93
  9 |                         ->  Vector Hash Join (10,17)                     |          7 | 16MB     |     123 | 3386294.56
 10 |                            ->  Vector Hash Join (11,13)                  |          7 | 19MB     |     112 | 3384018.02
 11 |                               ->  Vector Partition Iterator              |  287999764 | 1MB      |      12 | 227383.99
 12 |                                  ->  Partitioned CStore Scan on store_returns | 287999764 | 1MB |      12 | 227383.99
 13 |                               ->  Vector Hash Join (14,16)               |    1516824 | 16MB     |     124 | 3065686.08
 14 |                                  ->  Vector Partition Iterator           | 2879987999 | 1MB      |      66 | 2756066.50
 15 |                                     ->  Partitioned CStore Scan on store_sales | 2879987999 | 1MB |      66 | 2756066.50
 16 |                                  ->  CStore Scan on item                 |        158 | 1MB      |      58 | 4051.25
 17 |                            ->  CStore Scan on store                      |      24048 | 1MB      |      19 | 2264.00
 18 |                         ->  Vector Streaming(type: BROADCAST)            |  288000000 | 1MB      |       8 | 2176297.36
 19 |                            ->  CStore Scan on customer                   |   12000000 | 1MB      |       8 | 12923.00
 20 |                      ->  CStore Scan on customer_address ad2             |    6000000 | 1MB      |      58 | 5770.00
 21 |                   ->  CStore Scan on promotion                           |      36000 | 1MB      |       4 | 1268.50
(21 rows)
```

3.3.9.6 Scan 方式的 Hint

功能描述

指明scan使用的方法，可以是tablescan、indexscan和indexonlyscan。

语法格式

```
[no] tablescan|indexscan|indexonlyscan(table [index])
```

参数说明

- **no**表示hint的scan方式不使用。
- **table**表示hint指定的表，只能指定一个表，如果表存在别名应优先使用别名进行hint。
- **index**表示使用indexscan或indexonlyscan的hint时，指定的索引名称，当前只能指定一个。

对于indexscan或indexonlyscan，只有hint的索引属于hint的表时，才能使用该hint。

scan hint支持在行列存表、obs表、子查询表上指定。

示例

为了hint使用索引扫描，需要首先在表item的i_item_sk列上创建索引，名称为i。

```
create index i on item(i_item_sk);
```

对**示例**中原语句使用如下hint:

```
explain
select /*+ indexscan(item i) */ i_product_name product_name ...
```

该hint表示：item表使用索引i进行扫描。生成计划如下所示：

```
 id |                                operation                                 |   E-rows   | E-memory | E-width |     E-costs
----+--------------------------------------------------------------------------+------------+----------+---------+-----------------
  1 | ->  Row Adapter                                                          |          6 |          |     273 | 100061674938.26
  2 |    ->  Vector Streaming (type: GATHER)                                   |          6 |          |     273 | 100061674938.26
  3 |       ->  Vector Hash Aggregate                                          |          6 | 16MB     |     273 | 100061674936.59
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)                        |          6 | 1MB      |     169 | 100061674936.55
  5 |             ->  Vector Hash Join (6,21)                                  |          6 | 43MB     |     169 | 100061674936.19
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                  |          6 | 1MB      |     119 | 100061670368.67
  7 |                   ->  Vector Hash Join (8,20)                            |          6 | 16MB     |     119 | 100061670368.50
  8 |                      ->  Vector Hash Join (9,19)                         |          7 | 27MB     |     123 | 100061669081.23
  9 |                         ->  Vector Streaming(type: REDISTRIBUTE)         |          7 | 1MB      |     123 | 100061659600.47
 10 |                            ->  Vector Hash Join (11,18)                  |          7 | 16MB     |     123 | 100061659600.31
 11 |                               ->  Vector Nest Loop (12,17)               |          7 | 1MB      |     112 | 100061657323.77
 12 |                                  ->  Vector Hash Join (13,15)            |      13670 | 585MB    |      62 | 6163443.54
 13 |                                     ->  Vector Partition Iterator        | 2879987999 | 1MB      |      66 | 2756066.50
 14 |                                        ->  Partitioned CStore Scan on store_sales | 2879987999 | 1MB |      66 | 2756066.50
 15 |                                     ->  Vector Partition Iterator        |  287999764 | 1MB      |      12 | 227383.99
 16 |                                        ->  Partitioned CStore Scan on store_returns | 287999764 | 1MB |      12 | 227383.99
 17 |                                  ->  CStore Index Scan using i_ on item  |          1 | 1MB      |      58 | 4.01
 18 |                               ->  CStore Scan on store                   |      24048 | 1MB      |      19 | 2264.00
 19 |                            ->  CStore Scan on customer                   |   12000000 | 1MB      |       8 | 12923.00
 20 |                         ->  CStore Scan on promotion                     |      36000 | 1MB      |       4 | 1268.50
 21 |                   ->  CStore Scan on customer_address ad2                |    6000000 | 1MB      |      58 | 5770.00
(21 rows)
```

3.3.9.7 子链接块名的 hint

功能描述

指明子链接块的名称。

语法格式

```
blockname (table)
```

参数说明

- **table**表示为该子链接块hint的别名的名称。

说明

- **blockname hint**仅在对应的子链接块提升时才会被上层查询使用。目前支持的子链接提升包括IN子链接提升、EXISTS子链接提升和包含Agg等值相关子链接提升。该hint通常会和前面章节提到的hint联合使用。
- 对于FROM关键字后的子查询，则需要使用子查询的别名进行hint，blockname hint不会被用到。
- 如果子链接中含有多个表，则提升后这些表可与外层表以任意优化顺序连接，hint也不会被用到。

示例

```
explain select /*+nestloop(store_sales tt) */ * from store_sales where ss_item_sk in (select /*
+blockname(tt)*/ i_item_sk from item group by 1);
```

该hint表示：子链接的别名为tt，提升后与上层的store_sales表关联时使用nestloop。生成计划如下所示：

```
 id |                    operation                     |   E-rows   | E-memory | E-width |     E-costs
----+--------------------------------------------------+------------+----------+---------+-----------------
  1 | ->  Row Adapter                                  | 1439994000 |          |     216 | 325105765847.91
  2 |    ->  Vector Streaming (type: GATHER)           | 1439994000 |          |     216 | 325105765847.91
  3 |       ->  Vector Nest Loop Semi Join (4, 6)      | 1439994000 | 1MB      |     216 | 325026664615.00
  4 |          ->  Vector Partition Iterator           | 2879987999 | 1MB      |     216 | 2756066.50
  5 |             ->  Partitioned CStore Scan on store_sales | 2879987999 | 1MB |     216 | 2756066.50
  6 |          ->  Vector Materialize                  |     300000 | 16MB     |       4 | 4176.25
  7 |             ->  Vector Hash Aggregate            |     300000 | 16MB     |       4 | 3988.75
  8 |                ->  CStore Scan on item           |     300000 | 1MB      |       4 | 3832.50
(8 rows)
```

3.3.9.8 运行倾斜的 hint

功能描述

指明查询运行时重分布过程中存在倾斜的重分布键和倾斜值，针对Join和HashAgg运算中的重分布进行优化。

语法格式

- 指定单表倾斜：

  skew(table (column) [(value)])

- 指定中间结果倾斜：

  skew((join_rel) (column) [(value)])

参数说明

- **table**表示存在倾斜的单个表名。
- **join_rel**表示参与join的两个或多个表，如（t1 t2）表示t1和t2join后的结果存在倾斜。
- **column**表示倾斜表中存在倾斜的一个或多个列。
- **value**表示倾斜的列中存在倾斜的一个或多个值。

说明

- skew hint仅在需要重分布且指定的倾斜信息与查询执行过程中的重分布信息相匹配时才会被使用。
- skew hint目前仅处理普通表和子查询类型的表关系，支持基表hint、子查询hint、with as子句hint。对于子查询，无论提升与否都支持在skew hint中使用，这点与其它hint不一样。
- 对于倾斜表，如果定义了别名，则在hint中必须使用别名。
- 对于倾斜列，在不产生歧义的情况下，可以使用原名也可以使用别名。skew hint的column不支持表达式，如果需要指定采用分布键为表达式的重分布存在倾斜，需要将重分布键指定为新的列，以新的列进行hint。
- 对于倾斜值，个数需为列数的整数倍并按列的顺序进行组合，组合的个数不能超过10个。如果各倾斜列的倾斜值的个数不一样，为了满足按列组合，值可以重复指定。如，表t1的c1和c2存在倾斜，c1列的倾斜值只有a1，而c2列的倾斜有b1和b2，则skew hint如下：skew(t1 (c1 c2) ((a1 b1)(a1 b2)))。例中(a1 b1)为一个值组合，NULL可以作为倾斜值出现，每个hint中的值组合不超过十个， 且需为列的整数倍。
- 在Join的重分布优化中，skew hint中的value不可缺省，在HashAgg中可以缺省。
- 对于表、列、值中若指定多个，则同类间需以空格分离。
- 对于倾斜值，不支持在hint中进行类型强转；对于string类型，需要使用单引号。

例如：

- 指定单表倾斜

  每一个skew hint用来表示一个表关系存在的倾斜信息，如果想要指定在查询中的多个表关系存在的倾斜信息，则通过指定多个skew hint实现。

  在指定skew时，包括以下四个场景的用法：

  - 单列单值： skew(t (c1) (v1))

    说明：表关系t的c1列中的v1值在查询执行中存在倾斜。
  - 单列多值：skew(t (c1) (v1 v2 v3 ...))

说明：表关系t的c1列中的v1、v2、v3…等值在查询执行中存在倾斜。

- 多列单值：skew(t (c1 c2) (v1 v2))

  说明：表关系t的c1列的v1值和c2列的v2值在查询执行中存在倾斜。

- 多列多值：skew(t (c1 c2) ((v1 v2) (v3 v4) (v5 v6) ...))

  说明：表关系t的c1列的v1、v3、v5…值和c2列的v2、v4、v6…值在查询执行中存在倾斜。

须知

多列多值时，各组倾斜值间也可以不使用括号，如：skew(t (c1 c2) (v1 v2 v3 v4 v5 v6 ...))。是否使用括号必须统一，不可混合，

如：skew(t (c1 c2) (v1 v2 v3 v4 （ v5 v6 ） ...)) 将会产生语法报错。

- 指定中间结果倾斜

  如果基表不存在倾斜，而是查询执行中的中间结果出现倾斜，则需要通过指定中间结果倾斜的skew hint来进行倾斜的调优。skew((t1 t2) (c1) (v1))

  说明：表关系t1和t2 Join后的结果存在倾斜，倾斜的是t1表的c1列，c1列的倾斜值是v1。

  为了避免产生歧义，“c1”只能存在于join_rel的一个表关系中，如果存在同名列则通过别名进行规避。

建议

- 如果查询具有多层，则哪一层出现倾斜，则将hint写在哪一层中。
- 对于提升的子查询，skew hint支持直接使用子查询名进行hint。如果明确子查询提升后的哪一个基表存在倾斜，则直接使用基表进行hint的可用性更高。
- 无论对于表或列，若存在别名，则优先使用别名进行hint。

示例

指定单表倾斜

- 原query中进行hint。

  采用如下查询进行skew hint倾斜调优的举例，查询语句及不带hint的原计划如下所示：

```
explain
with customer_total_return as
(select sr_customer_sk as ctr_customer_sk
,sr_store_sk as ctr_store_sk
,sum(SR_FEE) as ctr_total_return
from store_returns
,date_dim
where sr_returned_date_sk = d_date_sk
and d_year =2000
group by sr_customer_sk
,sr_store_sk)
 select  c_customer_id
from customer_total_return ctr1
,store
,customer
where ctr1.ctr_total_return > (select avg(ctr_total_return)*1.2
from customer_total_return ctr2
where ctr1.ctr_store_sk = ctr2.ctr_store_sk)
```



```
and s_store_sk = ctr1.ctr_store_sk
and s_state = 'NM'
and ctr1.ctr_customer_sk = c_customer_sk
order by c_customer_id
limit 100;
```

```
 id |                                operation                                 |  E-rows   |    E-memory    | E-width |  E-costs
----+--------------------------------------------------------------------------+-----------+----------------+---------+-----------
  1 | ->  Row Adapter                                                          |       100 |                |      20 | 911254.47
  2 |    ->  Vector Limit                                                      |       100 |                |      20 | 911254.47
  3 |       ->  Vector Streaming (type: GATHER)                                |      2400 |                |      20 | 911325.75
  4 |          ->  Vector Limit                                                |      2400 | 1MB            |      20 | 911247.62
  5 |             ->  Vector Sort                                              |   3684816 | 16MB           |      20 | 911631.21
  6 |                ->  Vector Hash Join (7,29)                               |   3684817 | 41MB(12374MB)  |      20 | 905379.41
  7 |                   ->  Vector Streaming(type: REDISTRIBUTE)               |   3684817 | 384KB          |       4 | 883010.31
  8 |                      ->  Vector Hash Join (9,19)                         |   3684817 | 16MB           |       4 | 861302.05
  9 |                         ->  Vector Hash Join (10,18)                     | 110544450 | 16MB           |      44 | 427109.71
 10 |                            ->  Vector Hash Aggregate                     |  50247501 | 397MB(12671MB) |      54 | 395302.57
 11 |                               ->  Vector Streaming(type: REDISTRIBUTE)   |  50247501 | 384KB          |      22 | 358663.76
 12 |                                  ->  Vector Hash Join (13,15)            |  50247501 | 16MB           |      22 | 294300.51
 13 |                                     ->  Vector Partition Iterator        | 287999764 | 1MB            |      26 | 227383.99
 14 |                                        ->  Partitioned CStore Scan on store_returns | 287999764 | 1MB |      26 | 227383.99
 15 |                                     ->  Vector Streaming(type: BROADCAST)|      8712 | 384KB          |       4 | 975.56
 16 |                                        ->  Vector Partition Iterator     |       363 | 1MB            |       4 | 910.65
 17 |                                           ->  Partitioned CStore Scan on date_dim | 363 | 1MB          |       4 | 910.65
 18 |                               ->  CStore Scan on store                   |        44 | 1MB            |       4 | 1006.39
 19 |                            ->  Vector Hash Aggregate                     |       192 | 16MB           |      68 | 426707.38
 20 |                               ->  Vector Subquery Scan on ctr2           |  50247501 | 1MB            |      36 | 416239.03
 21 |                                  ->  Vector Hash Aggregate               |  50247501 | 397MB(12671MB) |      54 | 395302.57
 22 |                                     ->  Vector Streaming(type: REDISTRIBUTE) |  50247501 | 384KB      |      22 | 358663.76
 23 |                                        ->  Vector Hash Join (24,26)      |  50247501 | 16MB           |      22 | 294300.51
 24 |                                           ->  Vector Partition Iterator  | 287999764 | 1MB            |      26 | 227383.99
 25 |                                              ->  Partitioned CStore Scan on store_returns | 287999764 | 1MB |      26 | 227383.99
 26 |                                           ->  Vector Streaming(type: BROADCAST) |      8712 | 384KB      |       4 | 975.56
 27 |                                              ->  Vector Partition Iterator |       363 | 1MB            |       4 | 910.65
 28 |                                                 ->  Partitioned CStore Scan on date_dim | 363 | 1MB      |       4 | 910.65
 29 |                   ->  CStore Scan on customer                            |  12000000 | 1MB            |      24 | 12923.00
(29 rows)
```

对内层with子句中的HashAgg和外层的Hash Join进行hint指定，带hint的查询如下：

```
explain
with customer_total_return as
(select /*+ skew(store_returns(sr_store_sk sr_customer_sk)) */sr_customer_sk as ctr_customer_sk
,sr_store_sk as ctr_store_sk
,sum(SR_FEE) as ctr_total_return
from store_returns
,date_dim
where sr_returned_date_sk = d_date_sk
and d_year =2000
group by sr_customer_sk
,sr_store_sk)
 select  /*+ skew(ctr1(ctr_customer_sk)(11))*/  c_customer_id
from customer_total_return ctr1
,store
,customer
where ctr1.ctr_total_return > (select avg(ctr_total_return)*1.2
from customer_total_return ctr2
where ctr1.ctr_store_sk = ctr2.ctr_store_sk)
and s_store_sk = ctr1.ctr_store_sk
and s_state = 'NM'
and ctr1.ctr_customer_sk = c_customer_sk
order by c_customer_id
limit 100;
```

该hint表示：内层with子句中的group by在做HashAgg中进行重分布时存在倾斜，对应原计划的10和21号Hash Agg算子；外层ctr1表的ctr_customer_sk列在做Hash Join中进行重分布时存在倾斜，对应原计划的6号算子。生成计划如下所示：


```
 id |                                                                operation                                                                |  E-rows   |   E-memory    | E-width |  E-costs
----+-----------------------------------------------------------------------------------------------------------------------------------------+-----------+---------------+---------+-----------
  1 | ->  Row Adapter                                                                                                                         |       100 |               |      20 | 1061778.14
  2 |    ->  Vector Limit                                                                                                                     |       100 |               |      20 | 1061778.14
  3 |       ->  Vector Streaming (type: GATHER)                                                                                               |      2400 |               |      20 | 1061849.41
  4 |          ->  Vector Limit                                                                                                               |      2400 | 1MB           |      20 | 1061771.29
  5 |             ->  Vector Sort                                                                                                             |   3684816 | 16MB          |      20 | 1062154.87
  6 |                ->  Vector Hash Join (7,31)                                                                                              |   3684817 | 41MB(12344MB) |      20 | 1055903.08
  7 |                   ->  Vector Streaming(type: PART REDISTRIBUTE PART ROUNDROBIN)                                                         |   3684817 | 384KB         |       4 | 1013056.49
  8 |                      ->  Vector Hash Join (9,20)                                                                                        |   3684817 | 16MB          |       4 | 1000006.10
  9 |                         ->  Vector Hash Join (10,19)                                                                                    |  11054450 | 16MB          |      44 | 496461.73
 10 |                            ->  Vector Hash Aggregate                                                                                    |  50247501 | 397MB(12010MB) |      54 | 464654.59
 11 |                               ->  Vector Streaming(type: REDISTRIBUTE)                                                                  |  50247501 | 384KB         |      54 | 428015.79
 12 |                                  ->  Vector Hash Aggregate                                                                              |  50247501 | 397MB(12010MB) |      54 | 330939.31
 13 |                                     ->  Vector Hash Join (14,16)                                                                        |  50247501 | 16MB          |      22 | 294300.51
 14 |                                        ->  Vector Partition Iterator                                                                    | 287999764 | 1MB           |      26 | 227383.99
 15 |                                           ->  Partitioned CStore Scan on store_returns                                                  | 287999764 | 1MB           |      26 | 227383.99
 16 |                                        ->  Vector Streaming(type: BROADCAST)                                                            |      8712 | 384KB         |       4 | 975.56
 17 |                                           ->  Vector Partition Iterator                                                                 |       363 | 1MB           |       4 | 910.65
 18 |                                              ->  Partitioned CStore Scan on date_dim                                                    |       363 | 1MB           |       4 | 910.65
 19 |                               ->  CStore Scan on store                                                                                  |        44 | 1MB           |       4 | 1006.39
 20 |                         ->  Vector Hash Aggregate                                                                                       |       192 | 16MB          |      68 | 496059.40
 21 |                            ->  Vector Subquery Scan on ctr2                                                                             |  50247501 | 1MB           |      36 | 485591.05
 22 |                               ->  Vector Hash Aggregate                                                                                 |  50247501 | 397MB(12010MB) |      54 | 464654.59
 23 |                                  ->  Vector Streaming(type: REDISTRIBUTE)                                                               |  50247501 | 384KB         |      54 | 428015.79
 24 |                                     ->  Vector Hash Aggregate                                                                           |  50247501 | 397MB(12010MB) |      54 | 330939.31
 25 |                                        ->  Vector Hash Join (26,28)                                                                     |  50247501 | 16MB          |      22 | 294300.51
 26 |                                           ->  Vector Partition Iterator                                                                 | 287999764 | 1MB           |      26 | 227383.99
 27 |                                              ->  Partitioned CStore Scan on store_returns                                               | 287999764 | 1MB           |      26 | 227383.99
 28 |                                           ->  Vector Streaming(type: BROADCAST)                                                         |      8712 | 384KB         |       4 | 975.56
 29 |                                              ->  Vector Partition Iterator                                                              |       363 | 1MB           |       4 | 910.65
 30 |                                                 ->  Partitioned CStore Scan on date_dim                                                 |       363 | 1MB           |       4 | 910.65
 31 |                   ->  Vector Streaming(type: PART LOCAL PART BROADCAST)                                                                 |  12000000 | 384KB         |      24 | 34485.50
 32 |                      ->  CStore Scan on customer                                                                                        |  12000000 | 1MB           |      24 | 12923.00
(32 rows)
```

从优化后的计划可以看出：

a. 对于Hash Agg，由于其重分布存在倾斜，所以优化为双层Agg。

b. 对于Hash Join，同样由于其重分布存在倾斜，所以优化为采用新的重分布算子。

- 需要改写query后进行hint

  不带hint的查询和计划如下：

```
explain select count(*) from store_sales_1 group by round(ss_list_price);
```

```
 id |                                                  operation                                                  |  E-rows   | E-memory | E-width |  E-costs
----+-------------------------------------------------------------------------------------------------------------+-----------+----------+---------+-----------
  1 | ->  Row Adapter                                                                                             |   16672   |          |      14 | 62261.28
  2 |    ->  Vector Streaming (type: GATHER)                                                                      |   16672   |          |      14 | 62261.28
  3 |       ->  Vector Streaming(type: LOCAL GATHER dop: 1/2)                                                     |   16672   | 32KB     |      14 | 61479.78
  4 |          ->  Vector Hash Aggregate                                                                          |   16672   | 16MB     |      14 | 61452.00
  5 |             ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2)                                         | 3112836   | 128KB    |       6 | 57498.43
  6 |                ->  CStore Scan on store_sales_1                                                             | 3112836   | 1MB      |       6 | 21810.25
(6 rows)
```

由于hint中列不支持表达式，在进行倾斜优化时需要借助subquery改写查询，改写后的查询和计划如下：

```
explain
select count(*)
from (select round(ss_list_price),ss_hdemo_sk
from store_sales_1)tmp(a,ss_hdemo_sk)
group by a;
```

```
 id |                                                  operation                                                  |  E-rows   | E-memory | E-width |  E-costs
----+-------------------------------------------------------------------------------------------------------------+-----------+----------+---------+-----------
  1 | ->  Row Adapter                                                                                             |   16672   |          |      14 | 62261.28
  2 |    ->  Vector Streaming (type: GATHER)                                                                      |   16672   |          |      14 | 62261.28
  3 |       ->  Vector Streaming(type: LOCAL GATHER dop: 1/2)                                                     |   16672   | 32KB     |      14 | 61479.78
  4 |          ->  Vector Hash Aggregate                                                                          |   16672   | 16MB     |      14 | 61452.00
  5 |             ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2)                                         | 3112836   | 128KB    |       6 | 57498.43
  6 |                ->  CStore Scan on store_sales_1                                                             | 3112836   | 1MB      |       6 | 21810.25
(6 rows)
```

改写注意不要影响到业务逻辑。

采用改写后的查询进行hint，带hint的查询和计划如下：

```
explain
select /*+ skew(tmp(a)) */ count(*)
from (select round(ss_list_price),ss_hdemo_sk
from store_sales_1)tmp(a,ss_hdemo_sk)
group by a;
```

```
 id |                                                  operation                                                  |  E-rows   | E-memory | E-width |  E-costs
----+-------------------------------------------------------------------------------------------------------------+-----------+----------+---------+-----------
  1 | ->  Row Adapter                                                                                             |   16672   |          |      14 | 27771.82
  2 |    ->  Vector Streaming (type: GATHER)                                                                      |   16672   |          |      14 | 27771.82
  3 |       ->  Vector Streaming(type: LOCAL GATHER dop: 1/2)                                                     |   16672   | 32KB     |      14 | 26990.32
  4 |          ->  Vector Hash Aggregate                                                                          |   16671   | 16MB     |      14 | 26962.54
  5 |             ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2)                                         |   66216   | 128KB    |      14 | 26838.09
  6 |                ->  Vector Hash Aggregate                                                                    |   66216   | 16MB     |      14 | 25949.61
  7 |                   ->  CStore Scan on store_sales_1                                                          | 3112836   | 1MB      |       6 | 21810.25
(7 rows)
```

从计划可以看出，对Hash Agg进行倾斜优化后，采用了双层agg实现，大大过滤了进行重分布时的数据量，减少了重分布时间。

此外，需要说明的是，对于子查询，支持使用查询内部的列进行hint，如：

```
explain
select /*+ skew(tmp(b)) */ count(*)
from (select round(ss_list_price) b,ss_hdemo_sk
from store_sales_1)tmp(a,ss_hdemo_sk)
group by a;
```

3.3.9.9 Hint 的错误、冲突及告警

Plan Hint的结果会体现在计划的变化上，可以通过explain来查看变化。

Hint中的错误不会影响语句的执行，只是不能生效，该错误会根据语句类型以不同方式提示用户。对于explain语句，hint的错误会以warning形式显示在界面上，对于非explain语句，会以debug1级别日志显示在日志中，关键字为PLANHINT。

hint的错误分为以下类型：

- 语法错误

  语法规则树归约失败，会报错，指出出错的位置。

  例如：hint关键字错误，leading hint或join hint指定2个表以下，其它hint未指定表等。一旦发现语法错误，则立即终止hint的解析，所以此时只有错误前面的解析完的hint有效。

  例如：

  leading((t1 t2)) nestloop(t1) rows(t1 t2 #10)

  nestloop(t1)存在语法错误，则终止解析，可用hint只有之前解析的leading((t1 t2))。

- 语义错误
  - 表不存在，存在多个，或在leading或join中出现多次，均会报语义错误。
  - scanhint中的index不存在，会报语义错误。
  - 另外，如果子查询提升后，同一层出现多个名称相同的表，且其中某个表需要被hint，hint会存在歧义，无法使用，需要为相同表增加别名规避。
- hint重复或冲突

  如果存在hint重复或冲突，只有第一个hint生效，其它hint均会失效，会给出提示。
  - hint重复是指，hint的方法及表名均相同。例如：nestloop(t1 t2) nestloop(t1 t2)。
  - hint冲突是指，table list一样的hint，存在不一样的hint，hint的冲突仅对于每一类hint方法检测冲突。

    例如：nestloop (t1 t2) hashjoin (t1 t2)，则后面与前面冲突，此时hashjoin的hint失效。注意：nestloop(t1 t2)和no mergejoin(t1 t2)不冲突。

须知

leading hint中的多个表会进行拆解。例如：leading ((t1 t2 t3))会拆解成：leading((t1 t2)) leading(((t1 t2) t3))，此时如果存在leading((t2 t1))，则两者冲突，后面的会被丢弃。（例外：指定内外表的hint若与不指定内外表的hint重复，则始终丢弃不指定内外表的hint。）

- 子链接提升后hint失效

子链接提升后的hint失效，会给出提示。通常出现在子链接中存在多个表连接的场景。提升后，子链接中的多个表不再作为一个整体出现在join中。

- 列类型不支持重分布
  - 对于skew hint来说，目的是为了进行重分布时的调优，所以当hint列的类型不支持重分布时，hint将无效。
- hint未被使用
  - 非等值join使用hashjoin hint或mergejoin hint。
  - 不包含索引的表使用indexscan hint或indexonlyscan hint。
  - 通常只有在索引列上使用过滤条件才会生成相应的索引路径，全表扫描将不会使用索引，因此使用indexscan hint或indexonlyscan hint将不会使用。
  - indexonlyscan只有输出列仅包含索引列才会使用，否则指定时hint不会被使用。
  - 多个表存在等值连接时，仅尝试有等值连接条件的表的连接，此时没有关联条件的表之间的路径将不会生成，所以指定相应的leading，join，rows hint将不使用，例如：t1 t2 t3表join，t1和t2, t2和t3有等值连接条件，则t1和t3不会优先连接，leading(t1 t3)不会被使用。
  - 生成stream计划时，如果表的分布列与join列相同，则不会生成redistribute的计划；如果不同，且另一表分布列与join列相同，只能生成redistribute的计划，不会生成broadcast的计划，指定相应的hint则不会被使用。
  - 如果子链接未被提升，则blockname hint不会被使用。
  - 对于skew hint，hint未被使用可能由于：
    - 计划中不需要进行重分布。
    - hint指定的列为包含分布键。
    - hint指定倾斜信息有误或不完整，如对于join优化未指定值。
    - 倾斜优化的GUC参数处于关闭状态。

3.3.9.10 Plan Hint 实际调优案例

本节以TPC-DS标准测试的Q24的部分语句为例，在1000X，24DN环境上，说明使用plan hint进行实际调优的过程。示例如下：

```
select avg(netpaid) from
(select c_last_name
,c_first_name
,s_store_name
,ca_state
,s_state
,i_color
,i_current_price
,i_manager_id
,i_units
,i_size
,sum(ss_sales_price) netpaid
from store_sales
,store_returns
,store
,item
,customer
,customer_address
where ss_ticket_number = sr_ticket_number
and ss_item_sk = sr_item_sk
and ss_customer_sk = c_customer_sk
```

```
and ss_item_sk = i_item_sk
and ss_store_sk = s_store_sk
and c_birth_country = upper(ca_country)
and s_zip = ca_zip
and s_market_id=7
group by c_last_name
,c_first_name
,s_store_name
,ca_state
,s_state
,i_color
,i_current_price
,i_manager_id
,i_units
,i_size);
```

1. 该语句的初始计划如下，运行时间110s：

```
 id |                                                operation                                                |          A-time          |  A-rows   |  E-rows
----+---------------------------------------------------------------------------------------------------------+--------------------------+-----------+-----------
  1 | ->  Row Adapter                                                                                         | 110324.107               |         1 |         1
  2 |    ->  Vector Aggregate                                                                                 | 110324.093               |         1 |         1
  3 |       ->  Vector Streaming (type: GATHER)                                                               | 110323.958               |        24 |        24
  4 |          ->  Vector Aggregate                                                                           | [110179.302,110309.653]  |        24 |        24
  5 |             ->  Vector Hash Aggregate                                                                   | [110178.388,110308.515]  |    647824 |     16656
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                                                 | [77616.177,96478.771]    |    666834733 |     16664
  7 |                   ->  Vector Hash Join (8,22)                                                           | [81727.257,84728.519]    |    666834733 |     16664
  8 |                      ->  Vector Streaming(type: REDISTRIBUTE)                                           | [78770.520,82021.087]    |    666834733 |     16664
  9 |                         ->  Vector Hash Join (10,21)                                                    | [88066.755,90701.860]    |    666834733 |     16664
 10 |                            ->  Vector Streaming(type: BROADCAST)                                        | [7940.962,21430.725]     |    591882336 |     51360
 11 |                               ->  Vector Hash Join (12,20)                                              | [2419.995,5319.606]      |    24661764 |      2140
 12 |                                  ->  Vector Streaming(type: REDISTRIBUTE)                               | [1750.448,4659.581]      |    25258268 |      2241
 13 |                                     ->  Vector Hash Join (14,18)                                        | [15240.666,17159.616]    |    25258268 |      2241
 14 |                                        ->  Vector Hash Join (15,17)                                     | [12112.913,13563.366]    |    252564412 |    472070592
 15 |                                           ->  Vector Partition Iterator                                 | [11148.731,12473.230]    | 2879987999 | 2879987999
 16 |                                              ->  Partitioned CStore Scan on public.store_sales          | [11097.921,12412.596]    | 2879987999 | 2879987999
 17 |                                           ->  CStore Scan on public.store                               | [0.447,0.689]            |        2064 |        2064
 18 |                                        ->  Vector Partition Iterator                                    | [296.805,319.014]        |    287999764 |    287999764
 19 |                                           ->  Partitioned CStore Scan on public.store_returns           | [292.938,314.787]        |    287999764 |    287999764
 20 |                                     ->  CStore Scan on public.customer                                  | [114.358,144.462]        |    12000000 |    12000000
 21 |                                  ->  CStore Scan on public.customer_address                             | [38.426,56.753]          |    6000000 |    6000000
 22 |                            ->  CStore Scan on public.item                                               | [3.160,5.026]            |    300000 |    300000
(22 rows)
```

该计划中，第10层算子使用broadcast性能较差，由于第11层算子估算行数为2140，比实际行数严重低估。错误行数估算主要来源于第13层算子的行数低估，根因是第13层hashjoin中，使用store_sales的(ss_ticket_number, ss_item_sk)列和store_returns的(sr_ticket_number, sr_item_sk)列进行关联，由于缺少多列相关性的估算导致行数严重低估。

2. 使用如下的rows hint进行调优后，计划如下，运行时间318s：

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns * 11270)*/ c_last_name ...
```

```
 id |                                                operation                                                |          A-time          |  A-rows   |  E-rows
----+---------------------------------------------------------------------------------------------------------+--------------------------+-----------+-----------
  1 | ->  Row Adapter                                                                                         | 318585.246               |         1 |         1
  2 |    ->  Vector Aggregate                                                                                 | 318585.232               |         1 |         1
  3 |       ->  Vector Streaming (type: GATHER)                                                               | 318585.082               |        24 |        24
  4 |          ->  Vector Aggregate                                                                           | [318323.324,318499.290]  |        24 |        24
  5 |             ->  Vector Hash Aggregate                                                                   | [318320.813,318497.054]  |    647824 |    187770504
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                                                 | [288074.860,305601.698]  |    666834733 |    187770507
  7 |                   ->  Vector Hash Join (8,22)                                                           | [253642.468,315808.664]  |    666834733 |    187770507
  8 |                      ->  Vector Hash Join (9,18)                                                        | [250904.317,315684.018]  |    666834733 |    187770507
  9 |                         ->  Vector Streaming(type: REDISTRIBUTE)                                        | [4552.500,310602.307]    |    275042158 |    147106999
 10 |                            ->  Vector Hash Join (11,17)                                                 | [7658.951,14053.823]     |    275042158 |    147106999
 11 |                               ->  Vector Streaming(type: REDISTRIBUTE)                                  | [3953.255,10264.943]     |    287999764 |    154060900
 12 |                                  ->  Vector Hash Join (13,15)                                           | [28196.188,32838.794]    |    287999764 |    154060900
 13 |                                     ->  Vector Partition Iterator                                       | [11477.673,12324.583]    | 2879987999 | 2879987999
 14 |                                        ->  Partitioned CStore Scan on public.store_sales                | [11411.382,12250.209]    | 2879987999 | 2879987999
 15 |                                     ->  Vector Partition Iterator                                       | [304.188,403.205]        |    287999764 |    287999764
 16 |                                        ->  Partitioned CStore Scan on public.store_returns              | [299.838,398.255]        |    287999764 |    287999764
 17 |                               ->  CStore Scan on public.customer                                        | [122.246,170.128]        |    12000000 |    12000000
 18 |                            ->  Vector Streaming(type: REDISTRIBUTE)                                     | [57.558,117.461]         |    492915 |    146467
 19 |                               ->  Vector Hash Join (20,21)                                              | [45.554,96.238]          |    492915 |    146467
 20 |                                  ->  CStore Scan on public.customer_address                             | [39.738,89.412]          |    6000000 |    6000000
 21 |                                  ->  CStore Scan on public.store                                        | [0.361,1.095]            |    2064 |    2064
 22 |                      ->  Vector Streaming(type: BROADCAST)                                              | [48.986,91.170]          |    7200000 |    7200000
 23 |                         ->  CStore Scan on public.item                                                  | [4.506,6.602]            |    300000 |    300000
(23 rows)
```

时间反而劣化了，原因是第8层hashjoin过慢引起第9层redistribute时间过慢导致，其中第9层redistribute并没有数据倾斜，hashjoin慢的原因是由于第18层redistribute后数据倾斜导致。

3. 经过实际数据查证，customer_address的两个join列的不同值数目较少，使用其进行join容易出现数据倾斜，故把customer_address放到最后进行join。使用如下的hint进行调优后，计划如下，运行时间116s：

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns *11270)
leading((store_sales store_returns store item customer) customer_address)*/
c_last_name ...
```

```
 id |                                operation                                |        A-time         |  A-rows   |  E-rows   |
----+-------------------------------------------------------------------------+-----------------------+-----------+-----------+
  1 | ->  Row Adapter                                                         | 116326.597            |         1 |         1 |
  2 |    ->  Vector Aggregate                                                 | 116326.590            |         1 |         1 |
  3 |       ->  Vector Streaming (type: GATHER)                               | 116326.473            |        24 |        24 |
  4 |          ->  Vector Aggregate                                           | [116157.161,116236.494] |        24 |        24 |
  5 |             ->  Vector Hash Aggregate                                   | [116155.328,116233.946] |    647824 |  187770504 |
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                 | [84103.951,102052.326] |  666834733 |  187770507 |
  7 |                   ->  Vector Hash Join (8,10)                           | [23229.469,47484.697] |  666834733 |  187770507 |
  8 |                      ->  Vector Streaming(type: REDISTRIBUTE)           | [38.367,74.930]       |   6000000 |   6000000 |
  9 |                         ->  CStore Scan on public.customer_address      | [69.877,121.460]      |   6000000 |   6000000 |
 10 |                      ->  Vector Streaming(type: REDISTRIBUTE)           | [17404.744,17567.550] |  24661764 |  24112909 |
 11 |                         ->  Vector Hash Join (12,22)                    | [16123.627,16397.246] |  24661764 |  24112909 |
 12 |                            ->  Vector Streaming(type: REDISTRIBUTE)     | [15320.663,15741.646] |  25258268 |  25252751 |
 13 |                               ->  Vector Hash Join (14,21)              | [14962.342,16375.458] |  25258268 |  25252751 |
 14 |                                  ->  Vector Hash Join (15,19)           | [14449.031,15825.949] |  25258268 |  25252751 |
 15 |                                     ->  Vector Hash Join (16,18)        | [11439.959,12510.065] |  252564412 |  472070592 |
 16 |                                        ->  Vector Partition Iterator    | [10531.986,11536.213] | 2879987999 | 2879987999 |
 17 |                                           ->  Partitioned CStore Scan on public.store_sales | [10483.634,11474.944] | 2879987999 | 2879987999 |
 18 |                                        ->  CStore Scan on public.store  | [0.347,0.463]         |      2064 |      2064 |
 19 |                                     ->  Vector Partition Iterator       | [293.977,365.021]     |  287999764 |  287999764 |
 20 |                                        ->  Partitioned CStore Scan on public.store_returns | [289.936,360.808] |  287999764 |  287999764 |
 21 |                                  ->  CStore Scan on public.item         | [3.109,5.245]         |    300000 |    300000 |
 22 |                            ->  CStore Scan on public.customer           | [113.871,141.791]     |  12000000 |  12000000 |
(22 rows)
```

发现时间基本花在了第6层redistribute算子上，需要进一步优化。

4. 由于最后一层redistribute包含倾斜，所以时间较长。为了避免倾斜，需要将item表放在最后join，由于item表的join并不能使行数减少。修改hint如下并执行，计划如下，运行时间120s：

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns *11270)
leading((customer_address (store_sales store_returns store customer) item))
c_last_name ...
```

```
 id |                                operation                                |        A-time         |  A-rows   |  E-rows   |
----+-------------------------------------------------------------------------+-----------------------+-----------+-----------+
  1 | ->  Row Adapter                                                         | 120377.258            |         1 |         1 |
  2 |    ->  Vector Aggregate                                                 | 120377.245            |         1 |         1 |
  3 |       ->  Vector Streaming (type: GATHER)                               | 120377.091            |        24 |        24 |
  4 |          ->  Vector Aggregate                                           | [120184.884,120301.704] |        24 |        24 |
  5 |             ->  Vector Hash Aggregate                                   | [120183.119,120297.845] |    647824 |  187770504 |
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                 | [87775.682,106070.878] |  666834733 |  187770507 |
  7 |                   ->  Vector Hash Join (8,22)                           | [22323.764,49878.523] |  666834733 |  187770507 |
  8 |                      ->  Vector Hash Join (9,11)                        | [21129.236,45208.255] |  666834733 |  187770507 |
  9 |                         ->  Vector Streaming(type: REDISTRIBUTE)        | [37.859,75.412]       |   6000000 |   6000000 |
 10 |                            ->  CStore Scan on public.customer_address   | [74.798,114.449]      |   6000000 |   6000000 |
 11 |                         ->  Vector Streaming(type: REDISTRIBUTE)        | [15714.458,15824.928] |  24661764 |  24112909 |
 12 |                            ->  Vector Hash Join (13,21)                 | [14637.516,14955.464] |  24661764 |  24112909 |
 13 |                               ->  Vector Streaming(type: REDISTRIBUTE)  | [13898.593,14333.200] |  25258268 |  25252751 |
 14 |                                  ->  Vector Hash Join (15,19)           | [14166.917,15378.244] |  25258268 |  25252751 |
 15 |                                     ->  Vector Hash Join (16,18)        | [11272.239,12052.532] |  252564412 |  472070592 |
 16 |                                        ->  Vector Partition Iterator    | [10409.566,11127.981] | 2879987999 | 2879987999 |
 17 |                                           ->  Partitioned CStore Scan on public.store_sales | [10365.838,11077.601] | 2879987999 | 2879987999 |
 18 |                                        ->  CStore Scan on public.store  | [0.431,0.609]         |      2064 |      2064 |
 19 |                                     ->  Vector Partition Iterator       | [343.780,408.254]     |  287999764 |  287999764 |
 20 |                                        ->  Partitioned CStore Scan on public.store_returns | [339.844,403.923] |  287999764 |  287999764 |
 21 |                            ->  CStore Scan on public.customer           | [117.234,163.598]     |  12000000 |  12000000 |
 22 |                      ->  Vector Streaming(type: BROADCAST)              | [44.571,130.129]      |   7200000 |   7200000 |
 23 |                         ->  CStore Scan on public.item                  | [4.169,6.347]         |    300000 |    300000 |
(23 rows)
```

该计划中的redistribute问题并没有解决，因为第22层item表做了broadcast，导致与customer_address表join后的倾斜并没有被消除掉。

5. 增加如下禁止item表做broadcast的hint，使与customer_address join的表做redistribute（也可以进行join表redistribute的hint），计划如下，运行时间105s：

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns *11270)
leading((customer_address (store_sales store_returns store customer) item))
no broadcast(item)*/
c_last_name ...
```

```
 id |                                operation                                |        A-time         | A-rows  |  E-rows   |
----+-------------------------------------------------------------------------+-----------------------+---------+-----------+
  1 | ->  Row Adapter                                                         | 105854.957            |       1 |         1 |
  2 |    ->  Vector Aggregate                                                 | 105854.948            |       1 |         1 |
  3 |       ->  Vector Streaming (type: GATHER)                               | 105854.825            |      24 |        24 |
  4 |          ->  Vector Aggregate                                           | [105706.709,105776.135] |      24 |        24 |
  5 |             ->  Vector Hash Aggregate                                   | [105705.061,105773.013] |  647824 | 187770504 |
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                 | [70701.966,89973.672] |  666834733 | 187770507 |
  7 |                   ->  Vector Hash Join (8,23)                           | [71759.500,79018.433] |  666834733 | 187770507 |
  8 |                      ->  Vector Streaming(type: REDISTRIBUTE)           | [69794.307,77269.178] |  666834733 | 187770507 |
  9 |                         ->  Vector Hash Join (10,12)                    | [21443.307,46714.378] |  666834733 | 187770507 |
 10 |                            ->  Vector Streaming(type: REDISTRIBUTE)     | [41.295,83.419]       |  6000000 |   6000000 |
 11 |                               ->  CStore Scan on public.customer_address | [70.405,166.072]      |  6000000 |   6000000 |
 12 |                            ->  Vector Streaming(type: REDISTRIBUTE)     | [15689.053,15788.475] |  24661764 |  24112909 |
 13 |                               ->  Vector Hash Join (14,22)              | [14517.847,14712.929] |  24661764 |  24112909 |
 14 |                                  ->  Vector Streaming(type: REDISTRIBUTE) | [13806.733,14089.770] |  25258268 |  25252751 |
 15 |                                     ->  Vector Hash Join (16,20)        | [13709.384,15095.449] |  25258268 |  25252751 |
 16 |                                        ->  Vector Hash Join (17,19)     | [10944.796,11827.285] |  252564412 | 472070592 |
 17 |                                           ->  Vector Partition Iterator | [10070.316,10884.728] |  2879987999 | 2879987999 |
 18 |                                              ->  Partitioned CStore Scan on public.store_sales | [10018.966,10828.990] |  2879987999 | 2879987999 |
 19 |                                           ->  CStore Scan on public.store | [0.447,0.568]         |  2064 |      2064 |
 20 |                                        ->  Vector Partition Iterator    | [293.042,329.056]     |  287999764 | 287999764 |
 21 |                                           ->  Partitioned CStore Scan on public.store_returns | [288.631,324.782]     |  287999764 | 287999764 |
 22 |                                  ->  CStore Scan on public.customer     | [113.735,138.235]     |  12000000 |  12000000 |
 23 |                      ->  CStore Scan on public.item                     | [3.127,5.357]         |  300000 |    300000 |
(23 rows)
```

6. 发现最后一层使用单层Agg，但行数缩减较多。使用相同的hint，同时结合参数best_agg_plan=3进行双层Agg调优，最终计划如下图所示，运行时间94s，完成调优。

```
 id |                                operation                                |        A-time         | A-rows  |  E-rows   |
----+-------------------------------------------------------------------------+-----------------------+---------+-----------+
  1 | ->  Row Adapter                                                         | 94004.670             |       1 |         1 |
  2 |    ->  Vector Aggregate                                                 | 94004.655             |       1 |         1 |
  3 |       ->  Vector Streaming (type: GATHER)                               | 94004.504             |      24 |        24 |
  4 |          ->  Vector Aggregate                                           | [93833.832,93928.052] |      24 |        24 |
  5 |             ->  Vector Hash Aggregate                                   | [93832.460,93926.412] |  647824 | 187770507 |
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                 | [93640.866,93787.939] |  647824 | 183912384 |
  7 |                   ->  Vector Hash Aggregate                             | [93687.544,93791.242] |  647824 | 183912384 |
  8 |                      ->  Vector Hash Join (9,24)                        | [70025.469,72773.161] |  666834733 | 187770507 |
  9 |                         ->  Vector Streaming(type: REDISTRIBUTE)        | [68242.223,71275.972] |  666834733 | 187770507 |
 10 |                            ->  Vector Hash Join (11,13)                 | [21421.136,44830.306] |  666834733 | 187770507 |
 11 |                               ->  Vector Streaming(type: REDISTRIBUTE)  | [35.444,71.328]       |  6000000 |   6000000 |
 12 |                                  ->  CStore Scan on public.customer_address | [67.246,119.224]      |  6000000 |   6000000 |
 13 |                               ->  Vector Streaming(type: REDISTRIBUTE)  | [16089.853,16212.570] |  24661764 |  24112909 |
 14 |                                  ->  Vector Hash Join (15,23)           | [14822.972,15188.942] |  24661764 |  24112909 |
 15 |                                     ->  Vector Streaming(type: REDISTRIBUTE) | [14061.867,14604.162] |  25258268 |  25252751 |
 16 |                                        ->  Vector Hash Join (17,21)     | [13949.756,15492.311] |  25258268 |  25252751 |
 17 |                                           ->  Vector Hash Join (18,20)  | [10935.742,12160.719] |  252564412 | 472070592 |
 18 |                                              ->  Vector Partition Iterator | [10052.958,11194.962] |  2879987999 | 2879987999 |
 19 |                                                 ->  Partitioned CStore Scan on public.store_sales | [10008.415,11143.984] |  2879987999 | 2879987999 |
 20 |                                              ->  CStore Scan on public.store | [0.452,0.839]         |  2064 |      2064 |
 21 |                                           ->  Vector Partition Iterator | [298.235,332.736]     |  287999764 | 287999764 |
 22 |                                              ->  Partitioned CStore Scan on public.store_returns | [294.067,327.629]     |  287999764 | 287999764 |
 23 |                                        ->  CStore Scan on public.customer | [114.377,145.156]     |  12000000 |  12000000 |
 24 |                         ->  CStore Scan on public.item                  | [3.150,3.530]         |  300000 |    300000 |
(24 rows)
```

如果有统计信息变更引起的查询劣化，可以考虑用plan hint来调整到之前的查询计划。这里以TPCH-Q17为例，在收集default_statistics_target设置为–2的统计信息之后，计划相比于默认统计信息发生劣化。

1. 默认统计信息（default_statistics_target设置为100）的计划如下：

```
 id |                                operation                                |        A-time
----+-------------------------------------------------------------------------+-----------------------
  1 | ->  Row Adapter                                                         | 265006.779
  2 |    ->  Vector Aggregate                                                 | 265006.764
  3 |       ->  Vector Streaming (type: GATHER)                               | 265006.071
  4 |          ->  Vector Aggregate                                           | [263699.512,264503.084]
  5 |             ->  Vector Hash Join (6,17)                                 | [263676.665,264477.932]
  6 |                ->  Vector Streaming(type: LOCAL GATHER dop: 1/4)        | [1.998,7.594]
  7 |                   ->  Vector Hash Aggregate                             | [201775.393,202432.672]
  8 |                      ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 4/4) | [201567.130,202231.524]
  9 |                         ->  Vector Hash Join (10,12)                    | [170675.231,199908.410]
 10 |                            ->  Vector Partition Iterator                | [34847.797,51968.266]
 11 |                               ->  Partitioned CStore Scan on tpch10wx_col.lineitem | [33805.013,51137.657]
 12 |                            ->  Vector Hash Aggregate                    | [23283.387,25359.493]
 13 |                               ->  Vector Streaming(type: SPLIT BROADCAST dop: 4/4) | [12850.624,14608.515]
 14 |                                  ->  Vector Hash Aggregate             | [2690.439,3616.623]
 15 |                                     ->  Vector Partition Iterator      | [2659.700,3579.390]
 16 |                                        ->  Partitioned CStore Scan on tpch10wx_col.part | [2642.213,3559.093]
 17 |                ->  Vector Streaming(type: REDISTRIBUTE dop: 1/4)        | [262300.732,262961.078]
 18 |                   ->  Vector Hash Join (19,21)                          | [225749.727,260990.322]
 19 |                      ->  Vector Partition Iterator                      | [40046.078,56220.694]
 20 |                         ->  Partitioned CStore Scan on tpch10wx_col.lineitem | [39204.414,55328.448]
 21 |                      ->  Vector Streaming(type: SPLIT BROADCAST dop: 4/4) | [55748.177,61987.136]
 22 |                         ->  Vector Partition Iterator                   | [3042.864,3873.942]
 23 |                            ->  Partitioned CStore Scan on tpch10wx_col.part | [3027.023,3848.159]
(23 rows)
```

2. 统计信息变更（default_statistics_target设置为–2）的计划如下：


```
id |                                                operation                                                |          A-time
---+---------------------------------------------------------------------------------------------------------+---------------------------
 1 | ->  Row Adapter                                                                                         | 1440492.994
 2 |    ->  Vector Aggregate                                                                                 | 1440492.982
 3 |       ->  Vector Streaming  (type: GATHER)                                                              | 1440491.021
 4 |          ->  Vector Streaming(type: LOCAL GATHER dop: 1/6)                                              | [1439737.284,1440008.568]
 5 |             ->  Vector Aggregate                                                                        | [1439008.369,1439854.148]
 6 |                ->  Vector Hash Join (7,18)                                                              | [1439006.016,1439851.619]
 7 |                   ->  Vector Streaming(type: LOCAL BROADCAST dop: 6/6)                                  | [2.932,139.405]
 8 |                      ->  Vector Hash Aggregate                                                          | [190452.312,195910.748]
 9 |                         ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 6/6)                         | [190171.929,195653.119]
10 |                            ->  Vector Hash Join (11,13)                                                 | [161076.195,178831.123]
11 |                               ->  Vector Partition Iterator                                             | [27306.318,45564.565]
12 |                                  ->  Partitioned CStore Scan on tpch10wx_col.lineitem                   | [26752.444,44912.020]
13 |                               ->  Vector Hash Aggregate                                                 | [35601.624,39812.058]
14 |                                  ->  Vector Streaming(type: SPLIT BROADCAST dop: 6/6)                   | [23096.460,27057.137]
15 |                                     ->  Vector Hash Aggregate                                           | [2372.587,3052.445]
16 |                                        ->  Vector Partition Iterator                                    | [2345.381,3012.732]
17 |                                           ->  Partitioned CStore Scan on tpch10wx_col.part              | [2329.874,2989.393]
18 |                   ->  Vector Hash Join (19,22)                                                          | [1437388.414,1438470.781]
19 |                      ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 6/6)                            | [1392693.529,1408571.859]
20 |                         ->  Vector Partition Iterator                                                   | [29065.204,41264.514]
21 |                            ->  Partitioned CStore Scan on tpch10wx_col.lineitem                         | [28212.219,40133.491]
22 |                      ->  Vector Streaming(type: LOCAL REDISTRIBUTE dop: 6/6)                            | [2570.841,3438.567]
23 |                         ->  Vector Partition Iterator                                                   | [2447.569,3276.369]
24 |                            ->  Partitioned CStore Scan on tpch10wx_col.part                             | [2432.124,3263.641]
(24 rows)
```

3. 经过对比，劣化的原因主要为lineitem和part表join时stream类型由BroadCast变更为Redistribute导致。可以对语句进行stream方式的hint来调整到之前的计划，例如：

```
select /*+ no redistribute(part lineitem) */
    sum(l_extendedprice) / 7.0 as avg_yearly
from
    lineitem,
    part
where
    p_partkey = l_partkey
    and p_brand = 'Brand#23'
    and p_container = 'MED BOX'
    and l_quantity < (
        select
            0.2 * avg(l_quantity)
        from
            lineitem
        where
            l_partkey = p_partkey
    );
```

3.3.10 检查隐式转换的性能问题

在某些场景下，数据类型的隐式转换可能会导致潜在的性能问题。请看如下的场景：

```
SET enable_fast_query_shipping = off;
CREATE TABLE t1(c1 VARCHAR, c2 VARCHAR);
CREATE INDEX on t1(c1);
EXPLAIN verbose SELECT * FROM t1 WHERE c1 = 10;
```

上述查询的执行计划如下：

```
                              QUERY PLAN
-----------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=0.06..13.29 rows=1 width=64)
   Output: c1, c2
   Node/s: All datanodes
   ->  Seq Scan on public.t1  (cost=0.00..13.20 rows=1 width=64)
         Output: c1, c2
         Distribute Key: c1
         Filter: ((t1.c1)::bigint = 10)
(7 rows)
```

c1的数据类型是varchar，当查询的过滤条件为c1 = 10时，优化器默认将c1隐式转换为bigint类型，导致两个后果：

- 不能进行DN裁剪，计划下发到所有DN上执行。
- 计划中不能使用Index Scan方式扫描数据。

这会引起潜在的性能问题。

当知道了问题原因后，可以做针对性的SQL改写。对于上面的场景，只要将过滤条件中的常量显示转换为varchar类型，结果如下：

```
EXPLAIN verbose SELECT * FROM t1 WHERE c1 = 10::varchar;
```

```
                                  QUERY PLAN
------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=0.06..8.36 rows=1 width=64)
   Output: c1, c2
   Node/s: datanode2
   ->  Index Scan using t1_c1_idx on public.t1  (cost=0.00..8.27 rows=1 width=64)
         Output: c1, c2
         Distribute Key: c1
         Index Cond: ((t1.c1)::text = '10'::text)
(7 rows)
```

为了提前识别隐式类型转换可能带来的性能影响，GaussDB提供了一个guc option：check_implicit_conversions。打开该参数后，对于查询中出现的隐式类型转换的索引列，在路径生成阶段进行检查，如果发现索引列没有生成候选的索引扫描路径，则会通过报错的形式提示给用户。举例如下：

```
SET check_implicit_conversions = on;
SELECT * FROM t1 WHERE c1 = 10;
ERROR:  There is no optional index path for index column: "t1"."c1".
Please check for potential performance problem.
```

说明

- 参数check_implicit_conversions只用于检查隐式类型转换引起的潜在性能问题，在正式生产环境中请关闭该参数（该参数默认关闭）。
- 在将check_implicit_conversions打开时，必须同时关闭enable_fast_query_shipping参数，否则由于后一个参数的作用，无法查看对隐式类型转换修复的结果。
- 一个表的候选路径可能包括seq scan和index scan等多个可能的数据扫描方式，最终执行计划使用的表扫描方式是由执行计划的代价来决定的，因此即使生成了索引扫描的候选路径，也可能生成的最终执行计划中使用其它扫描方式。

3.4 实际调优案例

3.4.1 案例：选择合适的分布列

现象描述

表定义如下：

```
CREATE TABLE t1 (a int, b int);
CREATE TABLE t2 (a int, b int);
```

执行如下查询：

```
SELECT * FROM t1, t2 WHERE t1.a = t2.b;
```

优化分析

如果将a作为t1和t2的分布列：

```
CREATE TABLE t1 (a int, b int) DISTRIBUTE BY HASH (a);
CREATE TABLE t2 (a int, b int) DISTRIBUTE BY HASH (a);
```

则执行计划将存在“Streaming”，导致DN之间存在较大通信数据量，如图3-10所示。

图 3-10 选择合适的分布列案例（一）

```
gaussdb => explain select * from t1, t2 where t1.a = t2.b;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=245.40..582.15 rows=240 width=16)
   Node/s: All datanodes
   ->  Hash Join  (cost=10.22..24.26 rows=10 width=16)
         Hash Cond: (t1.a = t2.b)
         ->  Seq Scan on t1  (cost=0.00..10.10 rows=10 width=8)
         ->  Hash  (cost=3.79..3.79 rows=10 width=8)
               ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..3.79 rows=10 width=8)
                     Spawn on: All datanodes
                     ->  Seq Scan on t2  (cost=0.00..10.10 rows=10 width=8)
(9 rows)
```

如果将a作为t1的分布列，将b作为t2的分布列：

```
CREATE TABLE t1 (a int, b int) DISTRIBUTE BY HASH (a);
CREATE TABLE t2 (a int, b int) DISTRIBUTE BY HASH (b);
```

则执行计划将不包含“Streaming”，减少DN之间存在的通信数据量，从而提升查询性能，如图3-11所示。

图 3-11 选择合适的分布列案例（二）

```
gaussdb => explain select * from t1, t2 where t1.a = t2.b;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=245.40..491.10 rows=240 width=16)
   Node/s: All datanodes
   ->  Hash Join  (cost=10.22..20.46 rows=10 width=16)
         Hash Cond: (t1.a = t2.b)
         ->  Seq Scan on t1  (cost=0.00..10.10 rows=10 width=8)
         ->  Hash  (cost=10.10..10.10 rows=10 width=8)
               ->  Seq Scan on t2  (cost=0.00..10.10 rows=10 width=8)
(7 rows)
```

3.4.2 案例：建立合适的索引

现象描述

查询所有员工的信息：

```
SELECT staff_id,first_name,last_name,employment_id,state_name,city
FROM staffs,sections,states,places
WHERE sections.section_name='Sales'
AND staffs.section_id = sections.section_id
AND sections.place_id = places.place_id
AND places.state_id = states.state_id
ORDER BY staff_id;
```

优化分析

在优化前，没有创建places.place_id和states.state_id索引，执行计划如下：

```
 id |                                operation                                |      A-time       | A-rows | E-rows | Peak Memory  | E-memory | A-width | E-width | E-costs
----+-------------------------------------------------------------------------+-------------------+--------+--------+--------------+----------+---------+---------+---------
  1 | ->  Streaming (type: GATHER)                                            | 69.801            |     34 |      2 |212KB         |          |         |     354 | 53.67
  2 |    ->  Sort                                                             | [21.508,23.283]   |     34 |      2 |[30KB, 36KB]  | 16MB     | [380,380] |     354 | 53.32
  3 |       ->  Hash Join (4,5)                                               | [21.483,23.153]   |     34 |      2 |[7KB, 7KB]    | 1MB      |         |     354 | 53.31
  4 |          ->  Seq Scan on hr.states                                      | [0.007,0.022]     |     17 |     20 |[12KB, 12KB]  | 1MB      |         |     110 | 13.13
  5 |          ->  Hash                                                       | [21.026,22.663]   |     34 |      2 |[262KB, 294KB]| 16MB     | [284,284] |     268 | 40.08
  6 |             ->  Streaming(type: REDISTRIBUTE)                           | [21.024,22.458]   |     34 |      2 |[85KB, 86KB]  | 1MB      |         |     268 | 40.08
  7 |                ->  Hash Join (8,9)                                      | [13.814,14.827]   |     34 |      2 |[8KB, 8KB]    | 1MB      |         |     268 | 39.80
  8 |                   ->  Seq Scan on hr.staffs                             | [0.035,0.043]     |    107 |     20 |[19KB, 19KB]  | 1MB      |         |     190 | 13.13
  9 |                   ->  Hash                                              | [13.361,14.348]   |      2 |      4 |[292KB, 292KB]| 16MB     | [124,124] |     102 | 26.57
 10 |                      ->  Streaming(type: BROADCAST)                     | [13.291,14.279]   |      2 |      4 |[85KB, 85KB]  | 1MB      |         |     102 | 26.57
 11 |                         ->  Hash Join (12,13)                           | [6.359,7.446]     |      1 |      2 |[6KB, 6KB]    | 1MB      |         |     102 | 26.48
 12 |                            ->  Seq Scan on hr.places                    | [0.008,0.018]     |     15 |     20 |[14KB, 14KB]  | 1MB      |         |     102 | 13.13
 13 |                            ->  Hash                                     | [5.999,7.077]     |      1 |      1 |[259KB, 291KB]| 16MB     | [32,32] |      24 | 13.28
 14 |                               ->  Streaming(type: REDISTRIBUTE)         | [5.999,6.958]     |      1 |      1 |[84KB, 85KB]  | 1MB      |         |      24 | 13.28
 15 |                                  ->  Seq Scan on hr.sections            | [0.021,0.022]     |      1 |      1 |[14KB, 14KB]  | 1MB      |         |      24 | 13.16
(15 rows)

         Predicate Information (identified by plan id)
-------------------------------------------------------------------
  3 --Hash Join (4,5)
       Hash Cond: (states.state_id = places.state_id)
  7 --Hash Join (8,9)
       Hash Cond: (staffs.section_id = sections.section_id)
 11 --Hash Join (12,13)
       Hash Cond: (places.place_id = sections.place_id)
 15 --Seq Scan on hr.sections
       Filter: ((sections.section_name)::text = 'Sales'::text)
       Rows Removed by Filter: 26
(9 rows)
```

建议在places.place_id和states.state_id列上建立2个索引，执行计划如下：

```
 id |                                operation                                |      A-time       | A-rows | E-rows | Peak Memory  | E-memory | A-width | E-width | E-costs
----+-------------------------------------------------------------------------+-------------------+--------+--------+--------------+----------+---------+---------+---------
  1 | ->  Streaming (type: GATHER)                                            | 50.411            |     34 |      2 | 212KB        |          |         |     354 | 46.84
  2 |    ->  Sort                                                             | [20.889,22.621]   |     34 |      2 | [30KB, 36KB] | 16MB     | [380,380] |     354 | 46.49
  3 |       ->  Nested Loop (4,13)                                            | [20.870,22.490]   |     34 |      2 | [5KB, 5KB]   | 1MB      |         |     354 | 46.48
  4 |          ->  Streaming(type: REDISTRIBUTE)                              | [20.869,21.852]   |     34 |      2 | [85KB, 86KB] | 1MB      |         |     268 | 35.46
  5 |             ->  Nested Loop (6,7)                                       | [13.433,14.452]   |     34 |      2 | [6KB, 6KB]   | 1MB      |         |     268 | 35.18
  6 |                ->  Seq Scan on hr.staffs                                | [0.027,0.032]     |    107 |     20 | [19KB, 19KB] | 1MB      |         |     190 | 13.13
  7 |                ->  Materialize                                          | [13.237,14.230]   |    109 |      4 | [10KB, 10KB] | 16MB     | [124,124] |     102 | 21.66
  8 |                   ->  Streaming(type: BROADCAST)                        | [13.150,14.137]   |      2 |      4 | [85KB, 85KB] | 1MB      |         |     102 | 21.65
  9 |                      ->  Nested Loop (10,12)                            | [5.784,6.822]     |      1 |      2 | [3KB, 3KB]   | 1MB      |         |     102 | 21.56
 10 |                         ->  Streaming(type: REDISTRIBUTE)               | [5.782,6.675]     |      1 |      1 | [84KB, 85KB] | 1MB      |         |      24 | 13.28
 11 |                            ->  Seq Scan on hr.sections                  | [0.019,0.020]     |      1 |      1 | [14KB, 14KB] | 1MB      |         |      24 | 13.16
 12 |                         ->  Index Scan using loc_id_pk on hr.places     | [0.091,0.091]     |      1 |      2 | [24KB, 24KB] | 1MB      |         |     102 | 8.27
 13 |          ->  Index Scan using state_c_id_pk on hr.states                | [0.352,0.352]     |     34 |      2 | [23KB, 23KB] | 1MB      |         |     110 | 5.50
(13 rows)

         Predicate Information (identified by plan id)
-------------------------------------------------------------------
  5 --Nested Loop (6,7)
       Join Filter: (sections.section_id = staffs.section_id)
       Rows Removed by Join Filter: 73
 11 --Seq Scan on hr.sections
       Filter: ((sections.section_name)::text = 'Sales'::text)
       Rows Removed by Filter: 26
 12 --Index Scan using loc_id_pk on hr.places
       Index Cond: (places.place_id = sections.place_id)
 13 --Index Scan using state_c_id_pk on hr.states
       Index Cond: (states.state_id = places.state_id)
(10 rows)
```

3.4.3 案例：增加 JOIN 列非空条件

现象描述

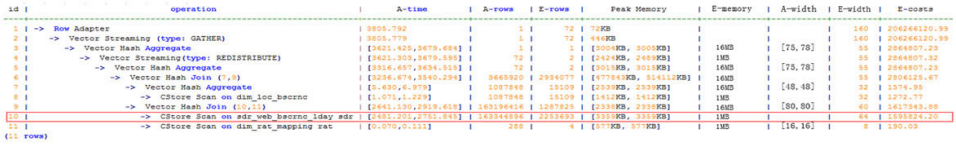
```
SELECT
 *
FROM
( ( SELECT
  STARTTIME STTIME,
  SUM(NVL(PAGE_DELAY_MSEL,0)) PAGE_DELAY_MSEL,
  SUM(NVL(PAGE_SUCCEED_TIMES,0)) PAGE_SUCCEED_TIMES,
  SUM(NVL(FST_PAGE_REQ_NUM,0)) FST_PAGE_REQ_NUM,
  SUM(NVL(PAGE_AVG_SIZE,0)) PAGE_AVG_SIZE,
  SUM(NVL(FST_PAGE_ACK_NUM,0)) FST_PAGE_ACK_NUM,
  SUM(NVL(DATATRANS_DW_DURATION,0)) DATATRANS_DW_DURATION,
  SUM(NVL(PAGE_SR_DELAY_MSEL,0)) PAGE_SR_DELAY_MSEL
 FROM
  PS.SDR_WEB_BSCRNC_1DAY SDR
  INNER JOIN (SELECT
      BSCRNC_ID,
      BSCRNC_NAME,
      ACCESS_TYPE,
      ACCESS_TYPE_ID
     FROM
      nethouse.DIM_LOC_BSCRNC
     GROUP BY
```



```
    BSCRNC_ID,
    BSCRNC_NAME,
    ACCESS_TYPE,
    ACCESS_TYPE_ID) DIM
 ON SDR.BSCRNC_ID = DIM.BSCRNC_ID
 AND DIM.ACCESS_TYPE_ID IN (0,1,2)
 INNER JOIN nethouse.DIM_RAT_MAPPING RAT
 ON (RAT.RAT = SDR.RAT)
WHERE
 ( (STARTTIME >= 1461340800
 AND STARTTIME < 1461427200) )
 AND RAT.ACCESS_TYPE_ID IN (0,1,2)
 --and SDR.BSCRNC_ID is not null
GROUP BY
 STTIME ) ) ;
```

执行计划如图3-12所示。

图 3-12 增加 JOIN 列非空条件（一）



优化分析

1. 分析执行计划图3-12可知，在顺序扫描阶段耗时较多。
2. 多表JOIN中，由于表PS.SDR_WEB_BSCRNC_1DAY的JOIN列“BSCRNC_ID”存在大量空值，JOIN性能差。

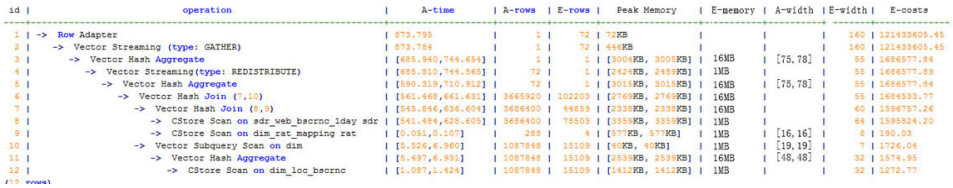
   建议在语句中手动添加JOIN列的非空判断，修改后的语句如下所示。

```
SELECT
 *
FROM
( ( SELECT
  STARTTIME STTIME,
  SUM(NVL(PAGE_DELAY_MSEL,0)) PAGE_DELAY_MSEL,
  SUM(NVL(PAGE_SUCCEED_TIMES,0)) PAGE_SUCCEED_TIMES,
  SUM(NVL(FST_PAGE_REQ_NUM,0)) FST_PAGE_REQ_NUM,
  SUM(NVL(PAGE_AVG_SIZE,0)) PAGE_AVG_SIZE,
  SUM(NVL(FST_PAGE_ACK_NUM,0)) FST_PAGE_ACK_NUM,
  SUM(NVL(DATATRANS_DW_DURATION,0)) DATATRANS_DW_DURATION,
  SUM(NVL(PAGE_SR_DELAY_MSEL,0)) PAGE_SR_DELAY_MSEL
 FROM
  PS.SDR_WEB_BSCRNC_1DAY SDR
  INNER JOIN (SELECT
      BSCRNC_ID,
      BSCRNC_NAME,
      ACCESS_TYPE,
      ACCESS_TYPE_ID
     FROM
      nethouse.DIM_LOC_BSCRNC
     GROUP BY
      BSCRNC_ID,
      BSCRNC_NAME,
      ACCESS_TYPE,
      ACCESS_TYPE_ID) DIM
  ON SDR.BSCRNC_ID = DIM.BSCRNC_ID
  AND DIM.ACCESS_TYPE_ID IN (0,1,2)
  INNER JOIN nethouse.DIM_RAT_MAPPING RAT
  ON (RAT.RAT = SDR.RAT)
 WHERE
  ( (STARTTIME >= 1461340800
  AND STARTTIME < 1461427200) )
```

```
 AND RAT.ACCESS_TYPE_ID IN (0,1,2)
 and SDR.BSCRNC_ID is not null
GROUP BY
 STTIME ) ) A;
```

执行计划如图3-13所示。

图 3-13 增加 JOIN 列非空条件（二）



3.4.4 案例：使排序下推

现象描述

在做场景性能测试时，发现某场景大部分时间是CN端在做window agg，占到总执行时间95%以上，系统资源不能充分利用。研究发现该场景的特点是：将两列分别求sum作为一个子查询，外层对两列的和再求和后做trunc，然后排序。

表结构如下所示：

```
CREATE TABLE public.test(imsi int,L4_DW_THROUGHPUT int,L4_UL_THROUGHPUT int)
with (orientation = column) DISTRIBUTE BY hash(imsi);
```

查询语句如下所示：

```
SELECT COUNT(1) over() AS DATACNT,
IMSI AS IMSI_IMSI,
CAST(TRUNC(((SUM(L4_UL_THROUGHPUT) + SUM(L4_DW_THROUGHPUT))), 0) AS
DECIMAL(20)) AS TOTAL_VOLOME_KPIID
FROM public.test AS test
GROUP BY IMSI
order by TOTAL_VOLOME_KPIID DESC;
```

执行计划如下：

```
Row Adapter  (cost=10.70..10.70 rows=10 width=12)
  ->  Vector Sort  (cost=10.68..10.70 rows=10 width=12)
        Sort Key: ((trunc((((sum(l4_ul_throughput)) + (sum(l4_dw_throughput))))::numeric,
0))::numeric(20,0))
        ->  Vector WindowAgg  (cost=10.09..10.51 rows=10 width=12)
              ->  Vector Streaming (type: GATHER)  (cost=242.04..246.84 rows=240 width=12)
                    Node/s: All datanodes
                    ->  Vector Hash Aggregate  (cost=10.09..10.29 rows=10 width=12)
                          Group By Key: imsi
                          ->  CStore Scan on test  (cost=0.00..10.01 rows=10 width=12)
```

可以看到window agg和sort全部在CN端执行，耗时非常严重。

优化分析

尝试将语句改写为子查询。

```
SELECT COUNT(1) over() AS DATACNT, IMSI_IMSI, TOTAL_VOLOME_KPIID
FROM (SELECT IMSI AS IMSI_IMSI,
CAST(TRUNC(((SUM(L4_UL_THROUGHPUT) + SUM(L4_DW_THROUGHPUT))),
0) AS DECIMAL(20)) AS TOTAL_VOLOME_KPIID
FROM public.test AS test
```

```
GROUP BY IMSI
ORDER BY TOTAL_VOLOME_KPIID DESC);
```

将trunc两列的和作为一个子查询，然后在子查询的外面做window agg，这样排序就可以下推了，执行计划如下：

```
Row Adapter  (cost=10.70..10.70 rows=10 width=24)
  ->  Vector WindowAgg  (cost=10.45..10.70 rows=10 width=24)
        ->  Vector Streaming (type: GATHER)  (cost=250.83..253.83 rows=240 width=24)
              Node/s: All datanodes
              ->  Vector Sort  (cost=10.45..10.48 rows=10 width=12)
                    Sort Key: ((trunc(((sum(test.l4_ul_throughput) + sum(test.l4_dw_throughput)))::numeric,
0))::numeric(20,0))
                    ->  Vector Hash Aggregate  (cost=10.09..10.29 rows=10 width=12)
                          Group By Key: test.imsi
                          ->  CStore Scan on test  (cost=0.00..10.01 rows=10 width=12)
```

经过SQL改写，性能由120s提升到7s，优化效果明显。

3.4.5 案例：设置 cost_param 对查询性能优化

现象描述 1

cost_param的bit0(set cost_param=1)值为1时，表示对于求由不等式（!=）条件连接的选择率时选择一种改良机制，此方法在自连接（两个相同的表之间连接）的估算中更加准确。下面查询的例子是cost_param的bit0为1时的优化场景。当前版本已弃用cost_param & 1不为0时的路径，默认选择已优化的估算公式。

注：选择率是两表join时，满足join条件的行数在join结果集中所占的比率。

表结构如下所示：

```
CREATE TABLE LINEITEM
(
L_ORDERKEY BIGINT NOT NULL
, L_PARTKEY BIGINT NOT NULL
, L_SUPPKEY BIGINT NOT NULL
, L_LINENUMBER BIGINT 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
) with (orientation = column, COMPRESSION = MIDDLE) distribute by hash(L_ORDERKEY);

CREATE TABLE ORDERS
(
O_ORDERKEY BIGINT NOT NULL
, O_CUSTKEY BIGINT NOT NULL
, O_ORDERSTATUS CHAR(1) NOT NULL
, O_TOTALPRICE DECIMAL(15,2) NOT NULL
, O_ORDERDATE DATE NOT NULL
, O_ORDERPRIORITY CHAR(15) NOT NULL
, O_CLERK CHAR(15) NOT NULL
, O_SHIPPRIORITY BIGINT NOT NULL
, O_COMMENT VARCHAR(79) NOT NULL
)with (orientation = column, COMPRESSION = MIDDLE) distribute by hash(O_ORDERKEY);
```

查询语句如下所示：

```
explain verbose select
count(*) as numwait
from
lineitem l1,
orders
where
o_orderkey = l1.l_orderkey
and o_orderstatus = 'F'
and l1.l_receiptdate > l1.l_commitdate
and not exists (
select
*
from
lineitem l3
where
l3.l_orderkey = l1.l_orderkey
and l3.l_suppkey <> l1.l_suppkey
and l3.l_receiptdate > l3.l_commitdate
)
order by
numwait desc;
```

执行计划如下图所示：（verbose条件下，新增distinct列，受cost off/on控制，hashjoin行显示内外表的distinct估值，其他行为空）

```
 id |                          operation                          | E-rows | E-distinct | E-width | E-costs
----+-------------------------------------------------------------+--------+------------+---------+---------
  1 | ->  Row Adapter                                             |      1 |            |       8 | 39.36
  2 |    ->  Vector Sort                                          |      1 |            |       8 | 39.36
  3 |       ->  Vector Aggregate                                  |      1 |            |       8 | 39.34
  4 |          ->  Vector Streaming (type: GATHER)                |      2 |            |       8 | 39.34
  5 |             ->  Vector Aggregate                            |      2 |            |       8 | 39.25
  6 |                ->  Vector Hash Anti Join (7, 10)            |      2 | 4, 5       |       0 | 39.24
  7 |                   ->  Vector Hash Join (8,9)                |      2 | 200, 1     |      16 | 26.12
  8 |                      ->  CStore Scan on public.lineitem l1  |      7 |            |      16 | 13.05
  9 |                      ->  CStore Scan on public.orders       |      1 |            |       8 | 13.05
 10 |                   ->  CStore Scan on public.lineitem l3     |      7 |            |      16 | 13.05
(10 rows)
```

优化分析 1

以上查询为lineitem表自连接的Anti Join，当使用cost_param的bit0为0时，估算Anti Join的行数与实际行数相差很大，导致查询性能下降。可以通过设置cost_param的bit0为1时，使Anti Join的行数估算更准确，从而提高查询性能。优化后的执行计划如下：

```
id |                          operation                          |  E-rows   | E-memory | E-width |  E-costs
---+-------------------------------------------------------------+-----------+----------+---------+-----------
 1 |  -> Row Adapter                                             |         1 |          |       0 | 9104892.37
 2 |    -> Vector Sort                                           |         1 |          |       0 | 9104892.37
 3 |      -> Vector Aggregate                                    |         1 |          |       0 | 9104892.35
 4 |        -> Vector Streaming (type: GATHER)                   |        48 |          |       0 | 9104892.35
 5 |          -> Vector Aggregate                                |        48 | 1MB      |       0 | 9104890.82
 6 |            -> Vector Hash Join (7,12)                       | 2526630903 | 929MB    |       0 | 8973295.45
 7 |              -> Vector Hash Anti Join (8, 10)               | 1999996587 | 3178MB   |       8 | 7198231.14
 8 |                -> Vector Partition Iterator                 | 1999996587 | 1MB      |      16 | 3000158.25
 9 |                  -> Partitioned CStore Scan on public.lineitem 11 | 1999996587 | 1MB | 16 | 3000158.25
10 |                -> Vector Partition Iterator                 | 1999996587 | 1MB      |      16 | 3000158.25
11 |                  -> Partitioned CStore Scan on public.lineitem 13 | 1999996587 | 1MB | 16 | 3000158.25
12 |              -> Vector Partition Iterator                   |  730839014 | 1MB      |       8 | 589611.00
13 |                -> Partitioned CStore Scan on public.orders  |  730839014 | 1MB      |       8 | 589611.00
(13 rows)
```

现象描述 2

当cost_param的bit1(set cost_param=2)为1时，表示求多个过滤条件（Filter）的选择率时，选择最小的作为总的选择率，而非两者乘积，此方法在过滤条件的列之间关联性较强时估算更加准确。下面查询的例子是cost_param的bit1为1时的优化场景。

表结构如下所示：

```
CREATE TABLE NATION
(
  N_NATIONKEY INT NOT NULL
, N_NAME      CHAR(25) NOT NULL
, N_REGIONKEY INT NOT NULL
, N_COMMENT   VARCHAR(152)
) distribute by replication;
CREATE TABLE SUPPLIER
(
  S_SUPPKEY   BIGINT NOT NULL
, S_NAME      CHAR(25) NOT NULL
, S_ADDRESS   VARCHAR(40) NOT NULL
, S_NATIONKEY INT NOT NULL
, S_PHONE     CHAR(15) NOT NULL
, S_ACCTBAL   DECIMAL(15,2) NOT NULL
, S_COMMENT   VARCHAR(101) NOT NULL
) distribute by hash(S_SUPPKEY);
CREATE TABLE PARTSUPP
(
  PS_PARTKEY    BIGINT NOT NULL
, PS_SUPPKEY    BIGINT NOT NULL
, PS_AVAILQTY   BIGINT NOT NULL
, PS_SUPPLYCOST DECIMAL(15,2)NOT NULL
, PS_COMMENT    VARCHAR(199) NOT NULL
)distribute by hash(PS_PARTKEY);
```

查询语句如下所示：

```
set cost_param=2;
explain verbose select
nation,
sum(amount) as sum_profit
from
(
select
n_name as nation,
l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
from
supplier,
lineitem,
partsupp,
nation
where
s_suppkey = l_suppkey
and ps_suppkey = l_suppkey
and ps_partkey = l_partkey
and s_nationkey = n_nationkey
) as profit
group by nation
order by nation;
```

当cost_param的bit1为0时，执行计划如下图所示：

id	operation	E-rows	E-distinct	E-width	E-costs
1	-> Sort	1		208	61.52
2	-> HashAggregate	1		208	61.51
3	-> Streaming (type: GATHER)	2		208	61.51
4	-> HashAggregate	2		208	61.36
5	-> Hash Join (6,7)	2	20, 15	176	61.33
6	-> Seq Scan on public.nation	40		108	20.20
7	-> Hash	2		76	41.04
8	-> Hash Join (9,16)	2	10, 13	76	41.04
9	-> Streaming(type: REDISTRIBUTE)	2		88	27.73
10	-> Hash Join (11,14)	2	10, 13	88	27.62
11	-> Streaming(type: REDISTRIBUTE)	20		70	14.19
12	-> Row Adapter	21		70	13.01
13	-> CStore Scan on public.lineitem	20		70	13.01
14	-> Hash	21		34	13.13
15	-> Seq Scan on public.partsupp	20		34	13.13
16	-> Hash	21		12	13.13
17	-> Seq Scan on public.supplier	20		12	13.13

(17 rows)

优化分析 2

在以上查询中，supplier、lineitem、partsupp三表做hashjoin的条件为(lineitem.l_suppkey = supplier.s_suppkey) AND (lineitem.l_partkey = partsupp.ps_partkey)，此hashjoin条件中存在两个过滤条件，这前一个过滤条件中的lineitem.l_suppkey和后一个过滤条件中的lineitem.l_partkey同为lineitem表的两列，这两列存在强相关的关联关系。在这种情况，估算hashjoin条件的选择率时，如果使用cost_param的bit1为0时，实际是将AND的两个过滤条件分别计算的2个选择率的值相乘来得到hashjoin条件的选择率，导致行数估算不准确，查询性能较差。所以需要将cost_param的bit1为1时，选择最小的选择率作为总的选择率估算行数比较准确，查询性能较好，优化后的计划如下图所示：

id	operation	E-rows	E-distinct	E-width	E-costs
1	-> Sort	10		208	64.42
2	-> HashAggregate	10		208	64.23
3	-> Streaming (type: GATHER)	20		208	64.23
4	-> HashAggregate	20		208	62.71
5	-> Hash Join (6,7)	20	20, 10	176	62.46
6	-> Seq Scan on public.nation	40		108	20.20
7	-> Hash	20		76	41.97
8	-> Hash Join (9,16)	20	10, 13	76	41.97
9	-> Streaming(type: REDISTRIBUTE)	20		82	28.54
10	-> Hash Join (11,14)	20	10, 13	82	27.63
11	-> Streaming(type: REDISTRIBUTE)	20		70	14.19
12	-> Row Adapter	21		70	13.01
13	-> CStore Scan on public.lineitem	20		70	13.01
14	-> Hash	21		12	13.13
15	-> Seq Scan on public.supplier	20		12	13.13
16	-> Hash	21		34	13.13
17	-> Seq Scan on public.partsupp	20		34	13.13

(17 rows)

3.4.6 案例：调整分布键

现象描述

某局点测试过程中EXPLAIN ANALYZE后有如下情况：

id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	-> Streaming (type: GATHER)	94138.404	0	670912	292KB			73	102576573.63
2	-> Insert on temp_calc_emprate0101_t3	[93259.538,93430.438]	310	670912	[1108KB, 1108KB]	1MB		73	102534641.63
3	-> Streaming(type: REDISTRIBUTE)	[93259.507,93430.400]	310	670912	[2091KB, 2093KB]	1MB		73	102534641.63
4	-> Subquery Scan on "*SELECT*"	[93212.430,93419.986]	310	670912	[7KB, 7KB]	1MB		73	102533776.78
5	-> HashAggregate	[93212.425,93419.980]	310	670912	[145KB, 197KB]	16MB	[65,65]	45	102533645.74
6	-> Streaming(type: REDISTRIBUTE)	[93212.374,93419.924]	5586	670934	[2091KB, 2093KB]	1MB		45	102533305.05
7	-> Hash Join (8,12)	[2657.406,93339.924]	5586	670934	[20KB, 20KB]	1MB		45	102532655.39
8	-> Seq Scan on s_riskrate_setting a	[18.885,2940.983]	77252027	78594218	[812KB, 903KB]	1MB		56	275264.71
9	-> Hash	[2241.418,2713.381]	8536241	8536241	[1031KB, 97803KB]	16MB	[48,48]	46	50870.88
10	-> Streaming(type: REDISTRIBUTE)	[210.226,2617.195]	8536241	8536241	[2091KB, 2093KB]	1MB		46	50870.88
11	-> Seq Scan on temp_calc_emprate0101 b	[86.790,141.293]	8536241	8536241	[16KB, 16KB]	1MB		46	11564.79

(11 rows)

从执行信息上比较明确的可以看出HashJoin是整个计划的性能瓶颈点，并且从HashJoin的执行时间信息[2657.406,93339.924](数值的具体含义请参见**SQL执行计划详解**)，上可以看出HashJoin在不同的DN上存在严重的计算偏斜。

同时在Memory Information(如下图)中可以看出各个节点的内存资源消耗也存在极为严重的偏斜。

```
                Memory Information (identified by plan id)
--------------------------------------------------------------------------------
Coordinator:
   Query Peak Memory: 4MB
Datanode:
   Max Query Peak Memory: 118MB
   Min Query Peak Memory: 24MB
  12 --Hash
         Max Buckets: 131072   Max Batches: 1   Max Memory Usage: 91857kB
         Min Buckets: 131072   Min Batches: 1   Min Memory Usage: 0kB
(8 rows)
```

优化分析

上述两个特征表明了此SQL语句存在极为严重的计算倾斜。进一步向HashJoin算子的下层分析发现Seq Scan on s_riskrate_setting也存在极为严重的计算倾斜[38.885,2940.983]。根据Scan的含义推测此计划性能问题的根源在于表s_riskrate_setting数据的分布倾斜。实际分析之后确实发现表s_riskrate_setting存在严重的数据倾斜。整改之后性能从94s提升为50s。

3.4.7 案例：改建分区表

现象描述

如下简单SQL语句查询， 性能瓶颈点在dwcjk的Scan上。

```
gaussdb =# explain performance select zqdh, count(1) from dwcjk where cjrq = '2015-05-02 00:00:00' group by zqdh;
 id |                    operation                    |       A-time        | A-rows | E-rows |   Peak Memory   | E-memory | A-width | E-width | E-costs
----+-------------------------------------------------+---------------------+--------+--------+-----------------+----------+---------+---------+-----------
  1 | ->  Row Adapter                                 | 1599.794            |     58 |     12 | 19KB            |          |         |       7 | 771106.83
  2 |   ->  Vector Streaming (type: GATHER)           | 1599.781            |     58 |     12 | 210KB           |          |         |       7 | 771106.83
  3 |     ->  Vector Hash Aggregate                   | [1445.092,1446.332] |     58 |      2 | [2315KB, 2315KB] | 16MB     | [16,16] |       7 | 128517.80
  4 |       ->  Vector Streaming(type: REDISTRIBUTE)  | [1444.996,1446.259] |    340 |     12 | [247KB, 247KB]  | 1MB      |         |       7 | 128518.09
  5 |         ->  Vector Hash Aggregate               | [573.150,1261.354]  |    340 |     12 | [2297KB, 2297KB] | 16MB     | [16,16] |       7 | 128517.80
  6 |           ->  CStore Scan on public.dwcjk       | [330.178,1021.695]  | 10000000 | 1623137 | [786KB, 786KB]  | 1MB      |         |       7 | 120402.00
(6 rows)
```

优化分析

从业务层确认表数据(在cjrq字段上)有明显的日期特征，符合分区表的特征。重新规划dwcjk表的表定义：字段cjrq为分区键、天为间隔单位定义分区表dwcjk_part。修改后结果如下，性能提升近1倍。

```
gaussdb =# explain performance select zqdh, count(1) from dwcjk_part where cjrq = '2015-05-02 00:00:00' group by zqdh;
 id |                         operation                          |       A-time        | A-rows | E-rows |   Peak Memory   | E-memory | A-width | E-width | E-costs
----+------------------------------------------------------------+---------------------+--------+--------+-----------------+----------+---------+---------+-----------
  1 | ->  Row Adapter                                            | 977.457             |     58 |     14 | 19KB            |          |         |       7 | 773142.84
  2 |   ->  Vector Streaming (type: GATHER)                      | 977.437             |     58 |     14 | 210KB           |          |         |       7 | 773142.84
  3 |     ->  Vector Hash Aggregate                              | [651.238,734.931]   |     58 |      2 | [2316KB, 2316KB] | 16MB     | [16,16] |       7 | 128857.14
  4 |       ->  Vector Streaming(type: REDISTRIBUTE)             | [651.137,734.834]   |    340 |     14 | [247KB, 247KB]  | 1MB      |         |       7 | 128857.47
  5 |         ->  Vector Hash Aggregate                          | [402.145,515.752]   |    340 |     14 | [2297KB, 2297KB] | 16MB     | [16,16] |       7 | 128857.14
  6 |           ->  Vector Partition Iterator                    | [162.630,275.990]   | 10000000 | 1691000 | [312BYTE, 312BYTE] | 1MB      |         |       7 | 120402.00
  7 |             ->  Partitioned CStore Scan on public.dwcjk_part | [161.746,275.207]   | 10000000 | 1691000 | [795KB, 795KB]  | 1MB      |         |       7 | 120402.00
(7 rows)
```

4 权限管理

4.1 创建用户并授权使用 GaussDB

如果您需要对您所拥有的GaussDB进行精细的权限管理，您可以使用**统一身份认证服务**（Identity and Access Management，简称IAM），通过IAM，您可以：

- 根据企业的业务组织，在您的华为云账号中，给企业中不同职能部门的员工创建IAM用户，让员工拥有唯一安全凭证，并使用GaussDB资源。
- 根据企业用户的职能，设置不同的访问权限，以达到用户之间的权限隔离。
- 将GaussDB资源委托给更专业、高效的其他华为云账号或者云服务，这些账号或者云服务可以根据权限进行代运维。

如果华为云账号已经能满足您的要求，不需要创建独立的IAM用户，您可以跳过本章节，不影响您使用GaussDB服务的其它功能。

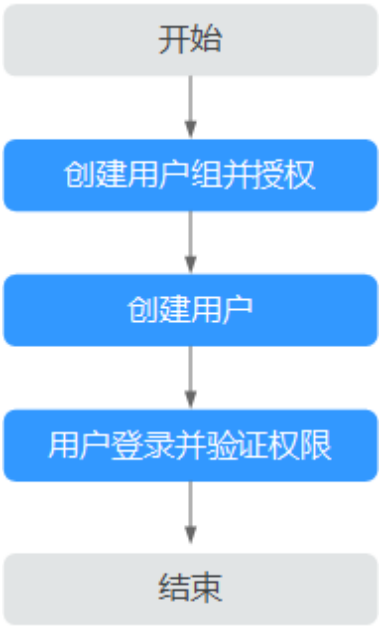
本章节为您介绍对用户授权的方法，操作流程如**图4-1**所示。

前提条件

给用户组授权之前，请您了解用户组可以添加的GaussDB系统策略，并结合实际需求进行选择。GaussDB支持的系统权限，请参见：**系统策略**。若您需要对除GaussDB之外的其它服务授权，IAM支持服务的所有策略请参见**权限策略**。

示例流程

图 4-1 给用户授权 GaussDB 权限流程



1. **创建用户组并授权**

   在IAM控制台创建用户组，并授予关系型数据库只读权限“GaussDB ReadOnlyAccess”。

2. **创建用户并加入用户组**

   在IAM控制台创建用户，并将其加入**1**中创建的用户组。

3. **用户登录**并验证权限

   新创建的用户登录控制台，切换至授权区域，验证权限：

   - 在“服务列表”中选择云数据库 GaussDB，进入GaussDB主界面，在左侧导航栏选择GaussDB > 实例管理。单击右上角“购买数据库实例”，尝试购买数据库实例，如果无法购买（假设当前权限仅包含GaussDB ReadOnlyAccess），表示“GaussDB ReadOnlyAccess”已生效。
   - 在“服务列表”中选择除云数据库 GaussDB外（假设当前策略仅包含GaussDB ReadOnlyAccess）的任一服务，若提示权限不足，表示“GaussDB ReadOnlyAccess”已生效。

4.2 自定义策略

如果系统预置的GaussDB权限，不满足您的授权要求，可以创建自定义策略。自定义策略中可以添加的授权项（Action）请参见**策略及授权项说明**。

目前支持以下两种方式创建自定义策略：

- 可视化视图创建自定义策略：无需了解策略语法，按可视化视图导航栏选择云服务、操作、资源、条件等策略内容，可自动生成策略。
- JSON视图创建自定义策略：可以在选择策略模板后，根据具体需求编辑策略内容；也可以直接在编辑框内编写JSON格式的策略内容。

具体创建步骤请参见：**创建自定义策略**。本章为您介绍常用的GaussDB自定义策略样例。

自定义策略样例

- 示例1：授权用户创建GaussDB实例

```
{
    "Version": "1.1",
    "Statement": [{
        "Effect": "Allow",
        "Action": ["gaussdb:instance:create"]
        }]
}
```

- 示例2：拒绝用户删除GaussDB实例

  拒绝策略需要同时配合其他策略使用，否则没有实际作用。用户被授予的策略中，一个授权项的作用如果同时存在Allow和Deny，则遵循Deny优先。

  如果您给用户授予GaussDB FullAccess的系统策略，但不希望用户拥有GaussDB FullAccess中定义的删除GaussDB实例权限，您可以创建一条拒绝删除云服务的自定义策略，然后同时将GaussDB FullAccess和拒绝策略授予用户，根据Deny优先原则，则用户可以对GaussDB实例执行除了删除GaussDB实例外的所有操作。拒绝策略示例如下：

```
{
    "Version": "1.1",
    "Statement": [{
        "Action": ["gaussdb:instance:delete"],
        "Effect": "Deny"
    }]
}
```

5 计费管理

5.1 实例续费

操作场景

您可根据业务需要，对单个“包年/包月”实例进行续费，暂不支持批量续费。

说明

“按需付费”的实例暂不支持续费。
运行状态为“正常”或“异常”的实例才可进行续费。

对当前实例续费

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择目标实例，单击“操作”列的“更多 > 续费”。

您也可以单击目标实例名称，进入实例的“基本信息”页面，在“计费信息”模块的“计费模式”处，单击“续费”。

步骤5 进入续费页面，对实例进行续费。

----结束

5.2 按需实例转包周期

操作场景

GaussDB服务支持按需实例转为包周期（包年/包月）实例。由于按需资源费用较高，需要长期使用资源的按需用户可以选择对按需资源进行转包周期，继续使用这些资源的同时，享受包周期的优惠资费。

说明

- 运行状态为冻结、创建失败、规格变更中、扩容中的实例不支持按需实例转包周期。
- 按需实例转包周期不会影响业务。

单个按需实例转包周期

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择目标实例，单击“操作”列的“更多 > 转包周期”，进入“按需转包周期”页面。

步骤5 选择续费规格，以月为单位，最小包周期时长为一个月。

- 如果订单确认无误，单击“去支付”，进入“支付”页面。
- 如果暂不确定实例规格，系统将保留您的订单，稍后可在“费用中心 > 订单管理 > 待支付订单”中支付或取消订单。

步骤6 选择支付方式，单击“确认付款”。

步骤7 按需转包周期创建成功后，用户可以在“实例管理”页面对其进行查看和管理。

在实例列表的右上角，单击刷新列表，可查看到按需转包周期完成后，实例状态显示为“正常”。“计费模式”显示为“包年/包月”。

----结束

按需实例批量转包周期

说明

仅“按需计费”模式的实例支持转包周期。

运行状态为“正常”或“异常”的实例才可转包周期。

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，勾选目标实例，单击实例列表上方的“转包周期”，进入“按需转包周期”页面。

步骤5 选择续费规格，以月为单位，最小包周期时长为一个月。

- 如果订单确认无误，单击“去支付”，进入“支付”页面。
- 如果暂不确定实例规格，系统将保留您的订单，稍后可在“费用中心 > 订单管理 > 待支付订单”中支付或取消订单。

步骤6 选择支付方式，单击“确认付款”。

步骤7 按需转包周期创建成功后，用户可以在“实例管理”页面对其进行查看和管理。

在实例列表的右上角，单击刷新列表，可查看到按需转包周期完成后，实例状态显示为“正常”。“计费模式”显示为“包年/包月”。

----结束

5.3 包周期实例转按需

操作场景

GaussDB支持单个包周期（包年/包月）实例转为按需实例，方便用户灵活使用该资费的实例。

须知

实例的按需计费方式需要等包周期到期后才会生效，且自动续费功能会同步失效。

单个包周期实例转按需

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择目标包周期实例，单击“操作”列的“更多 > 转按需”，进入“包周期转按需”页面。

步骤5 进入计费模式变更页面，对实例进行计费模式的变更。

步骤6 包周期转按需创建成功后，用户可以在“实例管理”页面对其进行查看和管理。

在实例列表的右上角，单击刷新列表，可查看到按需转包周期完成后，实例状态显示为“正常”。“计费模式”显示为“按需”。

----结束

包周期实例批量转按需

说明

仅“包年/包月”模式的实例支持转按需。
运行状态为“正常”或“异常”的实例才可转按需。
实例的按需计费方式需要等包周期到期后才会生效，且自动续费功能会同步失效。

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，勾选目标实例，单击实例列表上方的“转按需”，进入“包周期转按需”页面。

步骤5 进入计费模式变更页面，对实例进行计费模式的变更。

步骤6 包周期转按需创建成功后，用户可以在“实例管理”页面对其进行查看和管理。

在实例列表的右上角，单击  刷新列表，可查看到按需转包周期完成后，实例状态显示为“正常”。“计费模式”显示为“按需”。

----结束

5.4 退订包周期实例

操作场景

对于“包年/包月”模式的数据库实例，您需要退订订单，从而删除数据库实例资源。目前仅支持退订单个实例，具体退订操作请参考**退订单个实例**。关于退订费用，请参见**退订规则说明**。

对于“按需计费”模式的实例，您需要在“实例管理”页面对其进行删除，更多操作请参见**删除按需实例**。

退订单个实例（方法一）

您可在“实例管理”页面的实例列表中，退订包周期实例。

步骤1 **登录管理控制台**。

步骤2 单击管理控制台左上角的  ，选择区域和项目。

步骤3 在页面左上角单击  ，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择目标实例，单击“操作”列的“更多 > 退订”。

步骤5 在“退订资源”页面，确认待退订实例信息，并选择退订原因，单击“退订”。

资源退订相关信息，请参考**退订规则说明**。

步骤6 在弹出框中确认是否退订该资源，单击“退订”，提交退订申请。

须知

- 提交退订后，资源和数据将会被删除并无法找回。
- 如需保留数据，请务必确认完成数据备份后再提交退订。

步骤7 查看退订结果。数据库实例订单退订成功后，实例将会被删除，即“实例管理”页面将不再显示该订单对应的数据库实例。

----结束

退订单个实例（方法二）

您可前往“费用中心”，退订包周期实例。

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 单击页面右上角的“费用中心”，进入费用中心页面。

步骤5 在左侧导航栏中选择“订单管理 > 退订与退换货”，进入“退订与退换货”页面。

步骤6 在“云服务退订”页面，勾选需要退订的实例订单，单击“操作”列的“退订资源”。

- 您可通过产品类型筛选出账号下所有的云数据库GaussDB订单。

图 5-1 筛选服务

- 您还可以在订单列表上方，通过实例名称、订单号或实例ID搜索资源。

注意

单次操作允许最大退订资源数为100。

步骤7 在“退订使用中的资源”页面，确认待退订实例信息，并选择退订原因，单击“退订”。

资源退订相关信息，请参考退订规则说明。

步骤8 在弹出框中确认是否退订该资源，单击“是”，提交退订申请。

须知

1. 提交退订后，实例会被移至回收站，达到回收站设置的保留天数后实例会自动被永久删除。自动备份会随实例一起删除，手动备份不会自动删除，若保留会继续产生备份费用。如需删除，可在控制台的“备份恢复管理”页面删除。
2. 如需保留数据，请务必确认完成数据备份后再提交退订。

步骤9 查看退订结果。数据库实例订单退订成功后，实例将会被删除。

----结束

5.5 包周期实例开通自动续费

操作场景

GaussDB支持对单个包周期（包年/包月）实例开通自动续费。方便用户对实例进行续费管理。

须知

只有未开通自动续费功能的包周期实例，才有开通自动续费的操作。

对当前包周期实例开通自动续费

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择目标包周期实例，单击“操作”列的“更多 > 开通自动续费”，进入“开通自动续费”页面。

您也可以点击实例名称，进入实例的“基本信息”页面，在“计费信息”模块的“计费模式”处，单击“开通自动续费”。

步骤5 进入开通自动续费页面，对实例进行自动续费设置。

步骤6 包周期实例开通自动续费成功后，用户可以在“实例管理”页面对其进行查看和管理。

在实例列表的右上角，单击  刷新列表，可查看到开通自动续费后，实例状态显示为“正常”。“操作”列的“更多”显示“修改自动续费”。

----结束

5.6 包周期实例修改自动续费

操作场景

GaussDB支持对单个包周期（包年/包月）实例修改自动续费。方便用户对实例进行续费管理。

须知

只有开通了自动续费功能的包周期实例，才有修改自动续费的操作。

对当前包周期实例修改自动续费

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择目标包周期实例，单击“操作”列的“更多 > 修改自动续费”，进入“修改自动续费”页面。

您也可以点击实例名称，进入实例的“基本信息”页面，在“计费信息”模块的“计费模式”处，单击“修改自动续费”。

步骤5 进入修改自动续费页面，对实例进行自动续费设置。

步骤6 包周期实例修改自动续费成功后，用户可以在“实例管理”页面对其进行查看和管理。

----结束

6 数据库迁移

6.1 迁移方案总览

云数据库GaussDB提供了多种数据迁移方案，可满足从自建MySQL数据库、其他云上MySQL数据库、自建PostgreSQL数据库、其他云上PostgreSQL数据库、自建Oracle数据库、GaussDB、DB2 for LUW、RDS for SQL Server、本地SQL Server数据库、其他云SQL Server到云数据库GaussDB的迁移方案。

数据迁移工具有DRS和DAS。推荐使用DRS，DRS可以快速解决多场景下，数据库之间的数据流通问题，操作便捷、简单，仅需分钟级就能搭建完成迁移任务。通过服务化迁移，免去了传统的DBA人力成本和硬件成本，帮助您降低数据传输的成本。

DRS提供实时同步功能，在不同的系统之间，将数据通过同步技术从一个数据源拷贝到其他数据库，并保持一致，实现关键业务的数据实时流动。

更多内容，请参见**什么是云数据库GaussDB**。

表 6-1 GaussDB 迁移方案

源数据库类型	目标数据库类型	数据量	一次性或持续	应用程序停机时间	迁移方式	文档连接
• 本地自建MySQL数据库 • ECS自建MySQL数据库 • 其他云上MySQL数据库	GaussDB分布式版	任何	一次性或持续	最低	使用DRS将源库数据同步到GaussDB数据库	**将MySQL同步到GaussDB分布式版**（入云）
	GaussDB主备版	任何	一次性或持续	最低	使用DRS将源库数据同步到GaussDB数据库	**将MySQL同步到GaussDB主备版**（入云）

源数据库类型	目标数据库类型	数据量	一次性或持续	应用程序停机时间	迁移方式	文档连接
• 本地自建PostgreSQL数据库 • ECS自建PostgreSQL数据库 • 其他云上PostgreSQL数据库 • RDS for PostgreSQL	GaussDB主备版	任何	一次性或持续	最低	使用DRS将源库数据同步到GaussDB数据库	将PostgreSQL同步到GaussDB主备版
	GaussDB分布式版	任何	一次性或持续	最低	使用DRS将源库数据同步到GaussDB数据库	将PostgreSQL同步到GaussDB分布式版
• 本地自建Oracle数据库 • ECS自建Oracle数据库	GaussDB主备版	任何	一次性或持续	最低	使用DRS将源库数据同步到GaussDB数据库	将Oracle同步到GaussDB主备版（入云）
	GaussDB分布式版	任何	一次性或持续	最低	使用DRS将源库数据同步到GaussDB数据库	将Oracle同步到GaussDB分布式版（入云）
GaussDB分布式版	GaussDB分布式版	任何	一次性或持续	最低	使用DRS将源库数据同步到GaussDB数据库	将GaussDB分布式版同步到GaussDB分布式版（出云）
	GaussDB主备版 **说明** 目前仅支持白名单用户使用。	任何	一次性或持续	最低	使用DRS将源库数据同步到GaussDB数据库	将GaussDB分布式版同步到GaussDB主备版（出云）
	GaussDB分布式版	中	一次性	一段时间	使用DAS导出数据，再导入到GaussDB数据库	使用DAS的导出和导入功能迁移GaussDB数据
	GaussDB主备版					

源数据库类型	目标数据库类型	数据量	一次性或持续	应用程序停机时间	迁移方式	文档连接
GaussDB主备版	GaussDB分布式版	任何	一次性或持续	最低	使用DRS将源库数据同步到GaussDB数据库	将GaussDB主备版同步到GaussDB分布式版（出云）
	GaussDB主备版	任何	一次性或持续	最低	使用DRS将源库数据同步到GaussDB数据库	将GaussDB主备版同步到GaussDB主备版（出云）
	GaussDB分布式版	中	一次性	一段时间	使用DAS导出数据，再导入到GaussDB数据库	使用DAS的导出和导入功能迁移GaussDB数据
	GaussDB主备版					
DB2 for LUW	GaussDB分布式版	任何	一次性或持续	最低	使用DRS将源库数据同步到GaussDB数据库	将DB2 for LUW同步到GaussDB主备版（入云）
	GaussDB主备版	任何	一次性或持续	最低	使用DRS将源库数据同步到GaussDB数据库	将DB2 for LUW同步到GaussDB分布式版（入云）
• 本地自建Microsoft SQL Server数据库 • ECS自建Microsoft SQL Server数据库 • 其他云上Microsoft SQL Server数据库 • RDS for SQL Server	GaussDB主备版	任何	一次性或持续	最低	使用DRS将源库数据同步到GaussDB数据库	将Microsoft SQL Server同步到GaussDB主备版
	GaussDB分布式版	任何	一次性或持续	最低	使用DRS将源库数据同步到GaussDB数据库	将Microsoft SQL Server同步到GaussDB分布式版

6.2 使用 DAS 的导出和导入功能迁移 GaussDB 数据

操作场景

数据管理服务（Data Admin Service，简称DAS）是用来登录和操作华为云上数据库的Web服务，提供数据库开发、运维、智能诊断的一站式云上数据库管理平台，方便用户使用和运维数据库。

当进行数据备份或迁移时，支持使用DAS的数据导出功能，获取完整的数据信息，再将数据从本地或者从OBS桶导入目标数据表。

更多信息，请参见**导入导出**。

约束限制

- 导入单文件大小最大可达1GB。
- 可以支持导入的数据文件类型包括CSV格式和SQL文件格式。
- 暂不支持BINARY、VARBINARY、TINYBLOB、BLOB、MEDIUMBLOB、LONGBLOB等二进制类型字段的导入。
- 不支持使用跨区域的OBS桶导出导入数据。

导出数据

步骤1 **登录管理控制台**。

步骤2 单击管理控制台左上角的 ，选择区域和项目。

步骤3 在页面左上角单击 ，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择需要登录的目标数据库，单击操作列表中的“登录”，进入数据管理服务数据库登录界面。

步骤5 正确输入数据库用户名和密码，单击“登录”，即可进入您的数据库并进行管理。

步骤6 在顶部菜单栏选择“导入·导出”>“导出”。

步骤7 在导出页面，单击左上角“新建任务”，您可根据需要选择“导出数据库”或“导出SQL结果集”。下文以导出数据库为例，导出SQL结果集同理。

您也可通过单击“快速导出”，选择目标数据库，在创建快速导出任务弹窗中选择存储位置，并单击“确定”。

图 6-1 快速导出

步骤8 在导出数据库弹出页面，您可按需选填“基本信息”及“高级选项”，并在页面右侧选择需要导出的表。

图 6-2 创建数据库导出任务

说明

- SQL结果集导出任务中，执行SQL的最大限制是5M。

- 数据库分用户库和系统库，系统库不支持导出功能。如需导出，您需把创建用户数据库，业务部署到用户库，然后再执行导出操作。
- DAS在执行导出操作时，会连接到您的备库进行导出，可以有效规避导出时对主库的性能影响，但当备库复制延迟较大时，会存在“导出的数据不是最新数据“的可能性。

步骤9 设置完导出任务信息，单击弹出页面下部“确定”，创建导出任务。

步骤10 在导出任务列表页面，您可查看任务ID、任务类型、任务状态、进度等信息。

步骤11 您可在列表操作栏，单击“查看详情”，在任务详情弹出页面，查看本次导出任务执行的详情信息。

图 6-3 任务列表

----结束

导入数据

步骤1 在顶部菜单栏选择“导入·导出 > 导入”。

步骤2 DAS支持从本地选取文件导入，同时也支持从OBS桶中直接选择文件进行导入操作。

图 6-4 新建导入任务

- 上传文件

  在导入页面单击左上角的“新建任务”，在弹出框选择导入类型，选择文件来源为“上传文件”、附件存放位置等信息并上传文件，选择导入数据库，设置字符集类型，按需勾选选项设置及填写备注信息。

  为了保障数据安全，DAS将文件保存在OBS桶中。

  说明

  - 出于数据隐私性保护目的，DAS需要您提供一个您自己的OBS存储，用来接收您上传的附件信息，DAS会自动连接到该OBS文件，进行内存式读取，整个过程您的数据内容不会存储在DAS的任何存储介质上。
  - 导入完成后若勾选删除上传的文件选项，则该文件导入目标数据库成功后，将从OBS桶中自动删除。

- 从OBS桶中选择

  在导入页面单击左上角的“新建任务”，在弹出框设置导入类型，选择文件来源为“从OBS中选择”，在OBS文件浏览器弹窗中选择待导入文件，选择导入数据库，设置字符集类型，按需勾选选项设置及填写备注信息。

说明

从OBS桶中直接选择文件导入，导入目标数据库成功后，OBS桶不会删除该文件。

步骤3 导入信息设置完成后，单击“创建导入任务”即可。由于导入任务可能会覆盖您原有的数据，需再次确认无误后单击“确定”。

步骤4 您可在导入任务列表中查看导入进度等信息，在操作栏单击“查看详情”，您可在任务详情弹出框中，了解本次导入任务成功、失败等执行情况及耗时。

----结束

7 实例生命周期

7.1 重启实例

操作场景

当修改的实例参数需要重启生效时，可以重启实例。

须知

- 如果数据库实例未处于“正常”状态，则无法重启该实例。您的数据库可能会由于几个原因而不可用，例如，正在进行以前请求的修改操作。
- 重启数据库实例将导致数据库业务短暂中断，在此期间，数据库实例状态将显示为“重启中”。
- 重启过程中，实例将不可用。重启后实例会自动释放内存中的缓存，请注意对业务进行预热，避免业务高峰期出现阻塞。
- 建议重启过程中，减少数据库操作以快速完成重启。
- 存在大量慢SQL、会话或线程池满的情况下会增加一定的重启耗时。

操作步骤

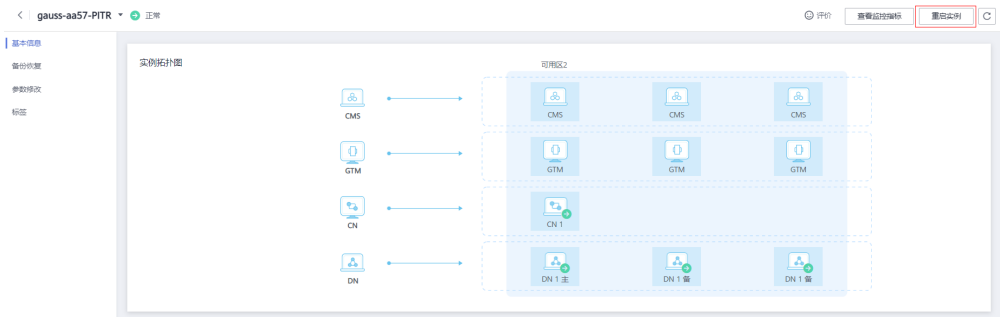
步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击 ☰，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击“操作”列的“更多 > 重启实例”。

您也可以在“实例管理”页面，单击目标实例名称，进入实例基本信息页面，在页面右上角，单击“重启实例”。



步骤5 在“重启实例”弹框中，单击“是”，重启实例。

数据库实例状态将显示为“重启中”，则说明重启指令下发成功。

步骤6 若您已开启高危操作保护，在弹出框中单击“去验证”，跳转至验证页面，单击“免费获取验证码”，正确输入验证码并单击“认证”，页面自动关闭。

通过进行二次认证再次确认您的身份，进一步提高账号安全性，有效保护您安全使用云产品。关于如何查看和开启高危操作保护，具体请参考**《统一身份认证服务用户指南》**的内容。

步骤7 稍后刷新实例列表，查看重启结果。如果实例状态为“正常”，说明实例重启成功。

----结束

7.2 删除按需实例

操作场景

- 用户需要删除不需要的数据库实例。
- 用户需要删除创建失败的数据库实例。

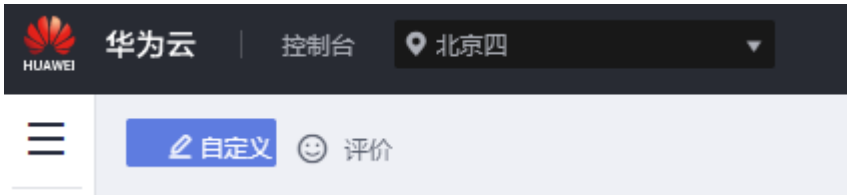
须知

- 实例删除后，不可恢复，请谨慎操作。如需保留数据，请务必确认完成数据备份后再删除实例。
- 执行操作中的实例不能手动删除，只有在实例操作完成后，才可删除实例。
- 通过数据库回收站中重建实例功能，可以恢复1~7天内删除的实例，具体请参见**回收站**。
- 回收站中待重建的实例不会计费。
- “按需计费”类型的实例删除后手动备份会继续保留。（对于包年/包月的实例，您需要进行订单退订才可删除实例，详细操作请参见**退订包周期实例**）。

删除按需实例

步骤1 **登录管理控制台**。

步骤2 单击管理控制台左上角的，选择区域和项目。



步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面的实例列表中，选择需要删除的实例，在“操作”列，单击“更多 > 删除实例”。

步骤5 在“删除实例”弹框，单击“是”下发请求，稍后刷新“实例管理”页面，查看删除结果。

步骤6 若您已开启高危操作保护，在“删除实例”弹框，单击“去验证”，跳转至验证页面，单击“免费获取验证码”，正确输入验证码并单击“认证”，页面自动关闭。

通过进行二次认证再次确认您的身份，进一步提高账号安全性，有效保护您安全使用云产品。关于如何开启操作保护，具体请参考《统一身份认证服务用户指南》的内容。

----结束

7.3 导出实例列表

操作场景

您可以导出实例列表，查看并分析实例信息。

使用限制

单租户最多支持同时导出3000个实例，具体导出耗时与实例数量有关。

导出所有实例

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，单击“导出实例列表”，默认选择所有的数据库实例。

步骤5 在导出弹框勾选所需导出信息，单击“导出”。

图 7-1 导出实例列表

步骤6 导出任务执行完成后，您可在本地查看到一个“.csv”文件。

----结束

7.4 回收站

GaussDB支持将删除的实例，加入回收站管理。您可以在回收站中重建实例恢复数据。

回收站策略机制默认开启，默认保留时间为7天，且不可关闭。

设置回收站策略

须知

- 修改回收站保留天数，仅对修改后新进入回收站的实例生效，对于修改前已经存在的实例，仍保持原来的回收策略，请您谨慎操作。
- 回收站中待重建的实例不会计费。

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在左侧导航栏，单击“回收站”。

步骤5 在“回收站”页面，单击“回收站策略”，设置已删除实例保留天数，可设置范围为1~7天。

步骤6 单击“确定”，完成设置。

----结束

重建实例

在回收站保留期限内的实例可以通过重建实例恢复数据。

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在左侧导航栏，单击“回收站”。

步骤5 在“回收站”页面，在实例列表中找到需要恢复的目标实例，单击操作列的“重建”。

步骤6 在“重建新实例”页面，选填配置后，提交重建任务。

重建相当于使用备份文件恢复到新实例，需要填写参数解释说明请参见创建实例。

----结束

8 变更实例

8.1 修改实例名称

操作场景

GaussDB支持修改实例名称，以方便用户识别。

约束

实例名称修改中，以下操作不可进行：

- 绑定弹性公网IP。
- 删除实例。
- 创建备份。

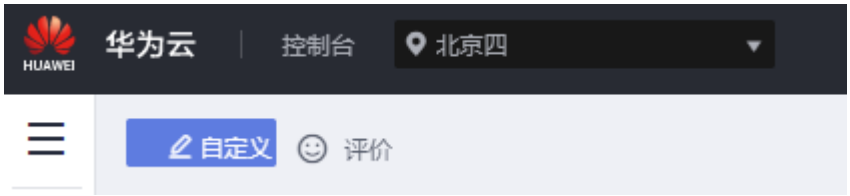
注意事项

- 修改实例名称时，允许和已有名称重复。
- 数据库实例名称修改后，数据库实例的标签仍与实例关联在一起。
- 数据库实例名称修改后，数据库实例的备份仍会保留。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。



步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，单击目标实例名称后的，编辑实例名称，单击“确认”，即可修改实例名称。

您也可以单击目标实例名称，进入实例的“基本信息”页面，在“数据库信息”模块实例名称处，单击，修改实例名称。

实例名称长度在4个到64个字符之间，必须以字母开头，可包含大写字母、小写字母、数字、中划线或下划线，不能包含其他特殊字符。

- 单击，提交修改。
- 单击，取消修改。

步骤5 在实例的“基本信息”页面，查看修改结果。

----结束

8.2 绑定和解绑弹性公网 IP

操作场景

GaussDB实例创建成功后，支持用户绑定弹性公网IP，在公共网络来访问数据库实例，绑定后也可根据需要解绑。

须知

为保证数据库可正常访问，请确保数据库使用的安全组开通了相关端口的访问权限，假设数据库的访问端口是1611，那么需确保安全组开通了1611端口的访问。

注意事项

- 对于已绑定弹性公网IP的实例，需解绑后，才可重新绑定其他弹性公网IP。
- 一个弹性公网IP只允许绑定一个数据库实例节点IP。

绑定弹性公网 IP

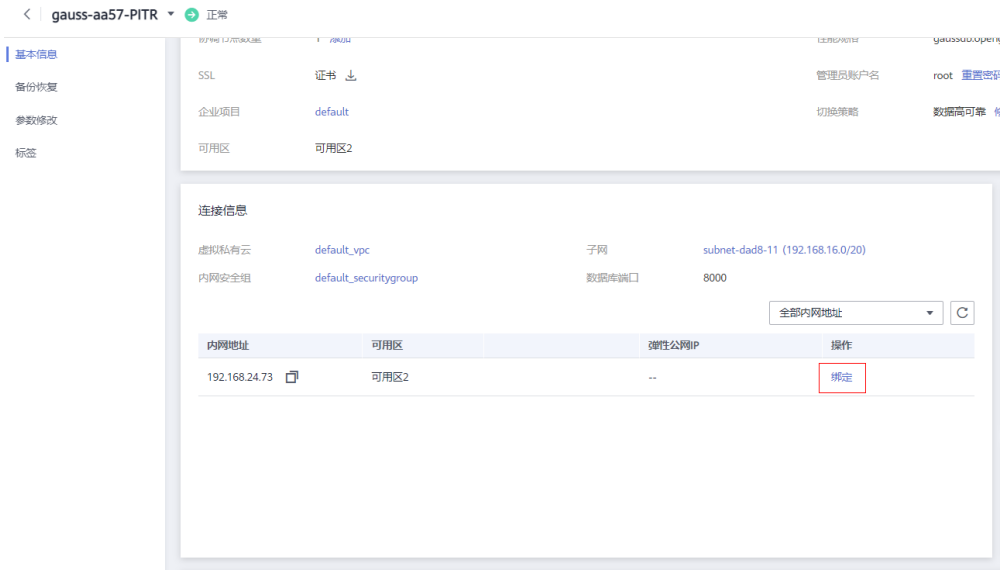
步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击实例名称，进入实例基本信息页面。

步骤5 在“连接信息”模块处，单击内网地址后的“绑定”。



步骤6 在弹出框的弹性公网IP地址列表中，显示“未绑定”状态的弹性公网IP，选择需要绑定的弹性公网IP，单击“确定”，提交绑定任务。如果没有可用的弹性公网IP，单击“查看弹性公网IP”，获取弹性公网IP。

步骤7 在“连接信息”模块的内网地址处，查看结果。

如需关闭，请参见**解绑弹性公网IP**。

说明

绑定成功后，您还可以单击内网IP前的 查看弹性公网IP详细信息。

----结束

解绑弹性公网 IP

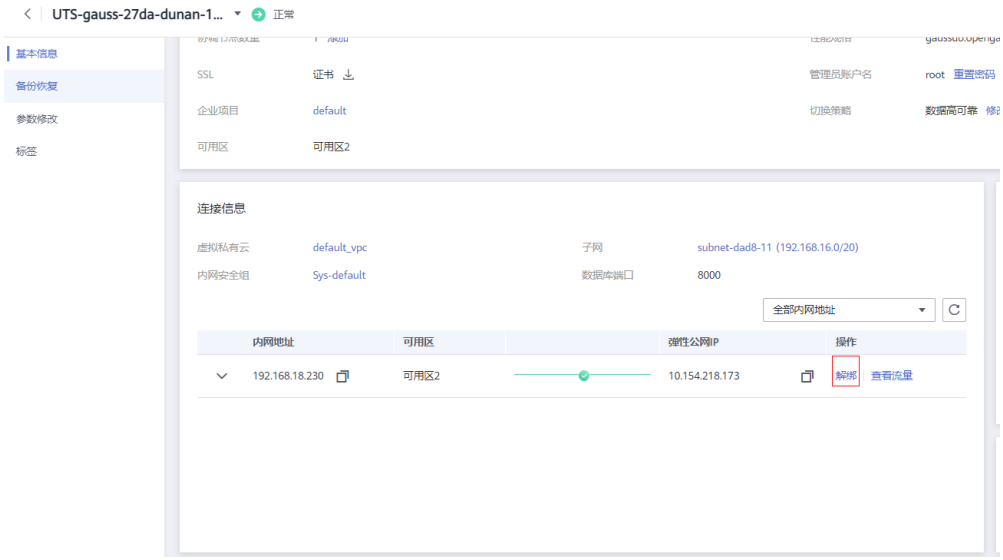
步骤1 **登录管理控制台**。

步骤2 单击管理控制台左上角的 ，选择区域和项目。

步骤3 在页面左上角单击☰，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 对于已绑定EIP的实例，在“实例管理”页面，选择指定实例，单击实例名称，进入实例基本信息页面。

步骤5 在“连接信息”模块，单击IP地址后面的“解绑”，在弹出框中单击“是”，解绑EIP。



步骤6 若您已开启高危操作保护，在“身份验证”弹出框中单击“获取验证码”，正确输入验证码并单击“确定”，页面自动关闭。

通过进行二次认证再次确认您的身份，进一步提高账号安全性，有效保护您安全使用云产品。关于如何开启操作保护，具体请参考**《统一身份认证服务用户指南》**的内容。

步骤7 在“连接信息”模块的内网地址处，查看结果。

如需重新绑定，请参见**绑定弹性公网IP**。

----结束

8.3 扩容实例

操作场景

随着实例部署时间及业务的增长，数据库在运行性能及存储上逐渐会达到瓶颈。此时，需要通过增加主机来提升实例的性能及存储能力。GaussDB独立部署形态支持扩容节点操作。

须知

- 扩容时长与业务数据量有关，默认扩容操作超时时间为7天，扩容中实例可正常使用，但不允许进行其他管理控制台操作，如需要进行操作，请及时联系客服处理。
- GaussDB支持灵活选择扩容CN或分片，建议扩容后实例中CN节点的数量小于或等于分片数量的两倍。
- 实例状态为“正常”时才能进行扩容，分片扩容过程中支持查询和插入、查询业务不中断，对用户的插入性能无影响，正在重分布的本地表跨节点组的关联查询业务，性能可能受到一定影响。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击实例的名称。

步骤5 在“基本信息”页面，进行扩容操作。

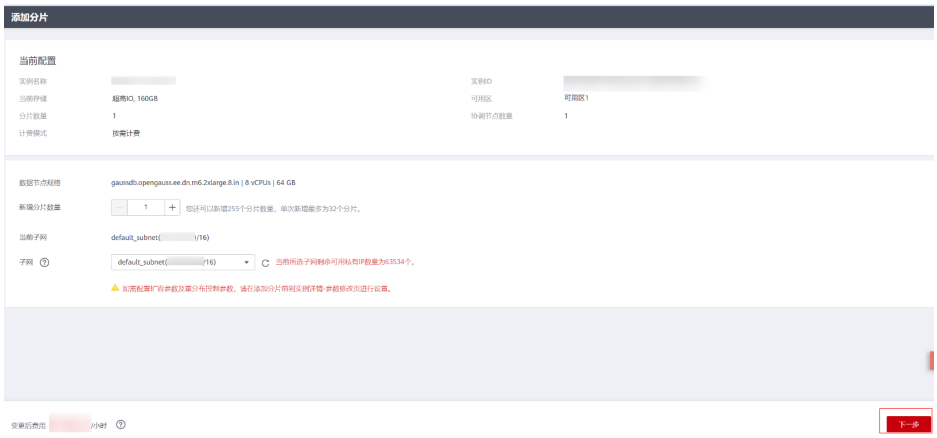
分片数量扩容

1. 单击“分片数量”后的“添加”。

图 8-1 分片扩容

2. 选择新增的分片数量。单击“下一步”。

图 8-2 分片扩容



3. 确认无误后，单击“提交”进行分片数量扩容。

说明

一个分片中默认包含三个DN副本，因此每增加一个分片会新增三个DN副本。

协调节点数量扩容

1. 单击“协调节点数量”后的“添加”。
2. 选择新增的协调节点数量以及可用区。

图 8-3 协调节点扩容

若创建实例时指定的可用区为1个，协调节点横向扩容需选择同一可用区。

3. 单击“下一步”，进入“规格确认”界面。
4. 确认无误后，单击“提交”进行协调节点扩容。

----结束

8.4 协调节点缩容

操作场景

随着业务下降，数据库协调节点利用率低，资源浪费严重。为提高资源利用率，需要对协调节点进行缩容。GaussDB支持协调节点缩容操作。

注意事项

- 协调节点缩容操作不会中断业务。
- 仅支持独立部署实例。
- 协调节点至少需要保留一个。
- 缩容前请确认该节点不在JDBC连接配置中，避免影响JDBC连接高可用性能。
- 缩容过程中执行的DDL操作将被回滚。
- 缩容过程中PITR备份暂停，缩容结束后自动恢复。
- 缩容完成后将自动进行一次全量备份。
- 缩容前须保证实例状态正常，所有协调节点状态正常，否则无法进行缩容。
- 当选择首节点缩容时，将随机替换为其他协调节点进行缩容。

操作步骤

步骤1 登录管理控制台。

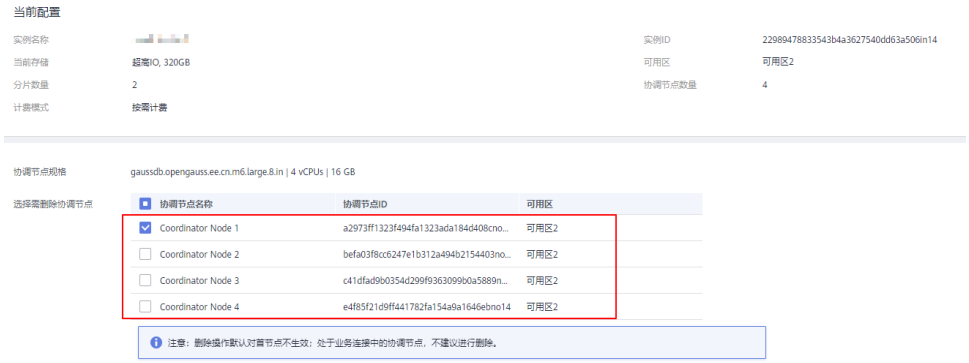
步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击实例的名称。

步骤5 在“基本信息”页面，进行缩容操作。

1. 单击“协调节点数量”后的“删除”。
2. 选择需要删除的协调节点。



3. 单击“下一步”，进入“删除协调节点”确认界面。
4. 确认要删除的协调节点正确无误后，单击“提交”进行协调节点缩容。

----结束

8.5 分片缩容

操作场景

实例进行读写分离或者业务冗余数据清理等操作后DN节点使用率会下降，此时可通过分片缩容避免成本浪费。GaussDB独立部署形态支持分片缩容操作。

说明

该功能仅针对特定用户开放，如需配置白名单权限，您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交开通白名单的申请。

注意事项

- 缩容时长与业务数据量有关，默认缩容操作超时时间为7天，缩容中实例可正常使用，但不允许进行其他管理控制台操作，如需要进行操作，请及时联系客服处理。
- 缩容期间会清理被缩减的DN分片上的原有会话，业务会部分受损，请合理选择业务低峰期进行操作。
- 集群缩容后需要至少保留一个分片，磁盘空间需满足（当前实例的磁盘使用量/缩容后DN个数 + 最大单表容量/缩容后DN个数）< 只读阈值（85%）* 实例磁盘总容量。
- 缩容过程中PITR备份暂停，缩容结束后自动恢复。
- 缩容完成后将自动进行一次全量备份。
- 实例状态为“正常”时才能进行缩容，分片缩容过程中支持查询和插入、查询业务不中断，对用户的插入性能无影响，正在重分布的本地表跨节点组的关联查询业务，性能可能受到一定影响。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

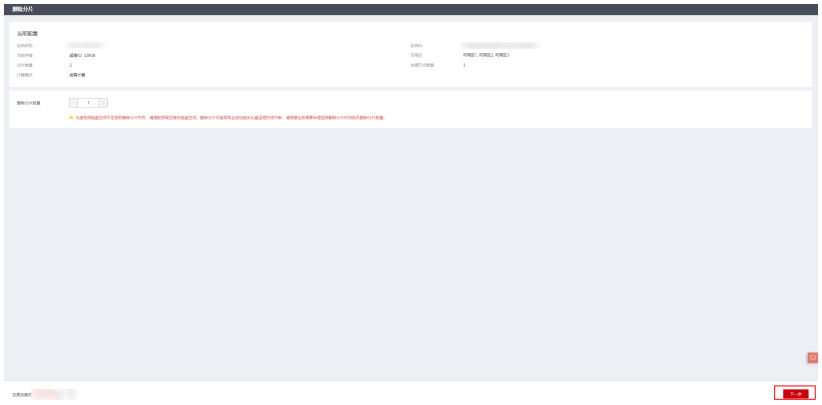
步骤4 在“实例管理”页面，选择指定的实例，单击实例的名称。

步骤5 在“基本信息”页面，“数据库信息”模块，单击“分片数量”后的“删除”。

图 8-4 分片缩容

步骤6 选择删除分片数量。单击“下一步”。

图 8-5 分片缩容



步骤7 确认无误后，单击“提交”进行分片数量缩容。

说明

一个分片中默认包含三个DN副本，因此每增加一个分片会缩减三个DN副本。

----结束

8.6 磁盘扩容

操作场景

GaussDB实例使用一段时间后业务攀升，原申请磁盘空间大小不足以支撑储存完整业务量。内核监测到磁盘使用量超过85%（默认值为85%，可以通过修改实例参数"cms:datastorage_threshold_value_check"进行配置），会将实例设置为只读，无法再写入数据，实例进入盘满只读状态。您可以通过磁盘扩容功能扩容数据库实例的磁盘。磁盘扩容期间，服务不中断。

注意事项

- 在最大值的允许范围内，扩容磁盘后磁盘使用率不能仍然高于85%。
- 节点状态需要为正常，否则应先联系运维人员修复节点。
- 扩容磁盘的大小必须是（40GB*分片数量）的整数倍。
- 一个分片磁盘容量最高24TB，每增加一个分片可以多增加24TB总容量。
- BMS的本地盘实例不支持磁盘扩容。

- 因磁盘空间占满而导致实例异常，支持磁盘扩容。

约束限制

- 账户余额大于等于0元，才可进行扩容。
- 默认最大可扩容至24TB，扩容次数没有限制。
- 磁盘扩容期间，实例状态为“扩容中”备份业务不受影响。
- 磁盘扩容的过程中，不需要重启数据库实例。
- 扩容过程中，该实例不可重启和删除。
- 磁盘容量变更只允许扩容，不能缩容。

操作步骤

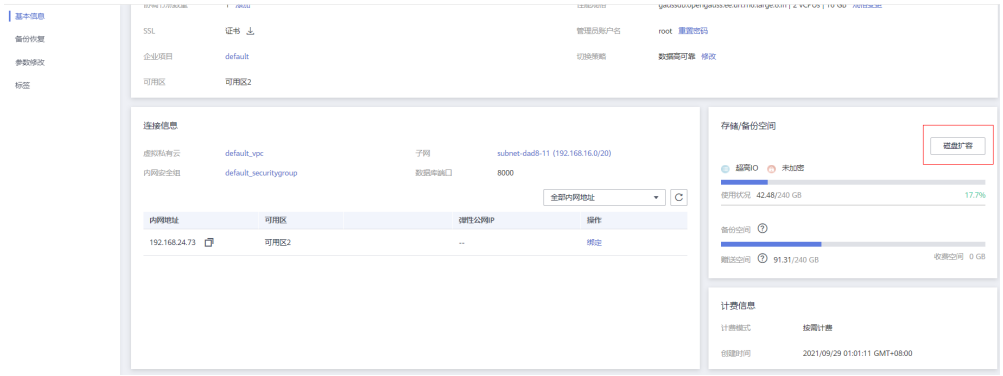
步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择目标实例，单击“操作”列的“磁盘扩容”，进入“磁盘扩容”页面。

您也可以通过单击目标实例名称，进入“基本信息”页面，在“存储/备份空间”模块的“存储空间”处，单击“磁盘扩容”，进入“磁盘扩容”页面。



步骤5 在“磁盘扩容”页面，选择空间大小，单击“下一步”。

选择空间大小时，需要考虑扩容后的磁盘使用率应该小于85%。只有低于85%时，才可以让实例从只读状态恢复为可读写状态。

步骤6 规格确认。

- 如果需要重新选择，单击“上一步”，回到上个页面，修改新增大小。
- 如果确认无误，单击“提交”，提交扩容。

步骤7 查看扩容结果。

在实例管理页面，可看到实例状态为“扩容中”，稍后单击实例名称，在“基本信息”页面，查看磁盘大小，检查扩容是否成功。此过程需要3～5分钟。

----结束

8.7 磁盘自动扩容

操作场景

GaussDB支持在磁盘达到一个阈值时，自动扩容磁盘，保障磁盘空间充足。

注意事项

- 仅支持实例下所有节点是正常可执行的状态。
- 磁盘自动扩容操作与磁盘扩容、节点扩容、磁盘类型变更、删除实例、快照检查、agent更新、磁盘自动扩容等互斥，即设置自动磁盘扩容策略时无法执行磁盘扩容等操作，磁盘扩容时也无法设置自动磁盘扩容策略操作。
- 主备版实例磁盘自动扩容属于实例级别扩容。
- 分布式版实例磁盘自动扩容属于分片级别扩容。
- 分布式版实例单分片磁盘扩容会导致每个分片磁盘容量不一致，将会影响分片数量添加和删除。
- 当可用磁盘存储空间率≤20%时，会触发自动扩容，增加当前分片存储空间的20%（非40倍数向上取整，账户余额不足/超出所设置存储自动扩容上限，会导致自动扩容失败）。
- 磁盘自动扩容失败时产生告警，当磁盘使用率低于阈值时告警会自消除。
- 包周期实例存在未完成订单时，不会自动扩容。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击实例的名称，进入“基本信息”页面。

步骤5 在“存储/备份空间”模块，单击“自动扩容”。

图 8-6 自动扩容

步骤6 在“存储空间自动扩容”弹框，设置如下参数：

图 8-7 存储空间自动扩容

表 8-1 参数说明

类别	说明
存储空间自动扩容	是否开启自动扩容，默认关闭。
可用存储空间率	空间使用率小于20%时会自动触发扩容，默认20%。

类别	说明
存储自动扩容上限	自动扩容上限，单位：GB。需要大于实例当前存储空间总大小。

步骤7 单击“确认”，则完成设置磁盘自动扩容策略。

----结束

8.8 修改实例参数

您可以实时修改GaussDB数据库实例参数，也可以通过该功能查看当前实例所使用的参数值。

修改参数

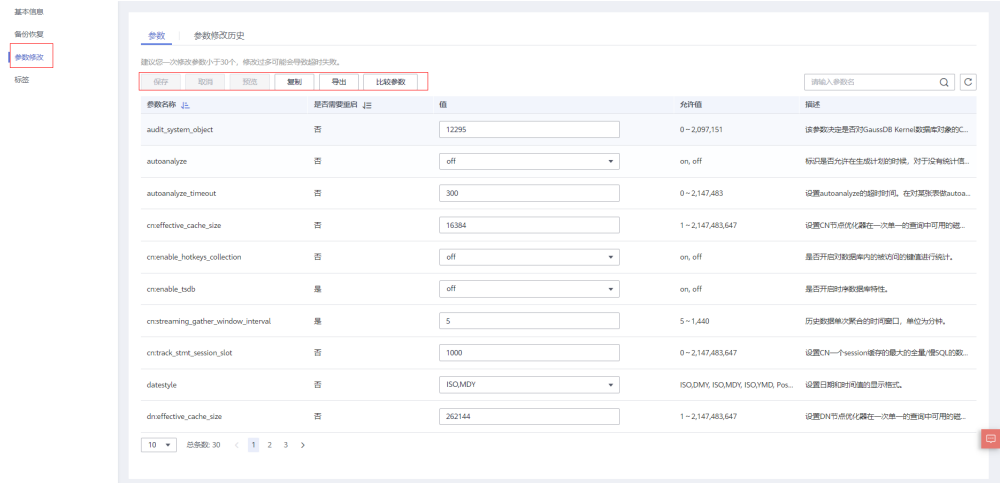
步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击实例名称，进入实例基本信息页面。

步骤5 在左侧导航栏单击“参数修改”，进入参数修改页面。



- 您可以在该页面对实例应用的参数进行修改和查询。修改参数后，可以预览修改结果或取消修改，确认修改无误后，保存修改内容。

说明

某些参数的修改需要重启才能生效，根据参数列表中“是否需要重启”提示，进行相应操作：

- 是：在实例列表中，查看“运行状态”，如果显示参数变更，等待重启，则需重启实例使之生效。
- 否：无需重启，立即生效。

- 您可以单击“复制”，将当前实例应用的参数另存为参数模板，可在**参数模板管理**的自定义页签中查看。
- 您还可以单击“导出”，将当前实例的参数下载到本地。
- 您可以单击“比较参数”，将已有的参数模板与当前实例应用的参数模板进行比较。

----结束

8.9 规格变更

操作场景

随着业务的不断增加，实例的CPU和内存资源可能成会为实例性能的瓶颈，无法满足业务要求时，GaussDB提供了规格变更功能来提升实例的CPU和内存。

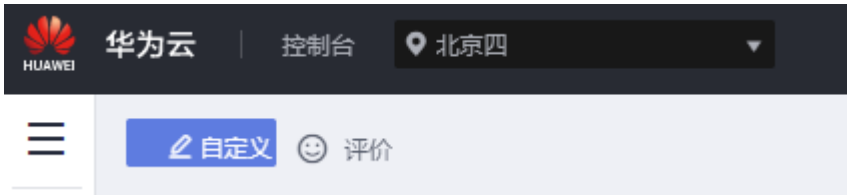
注意事项

- 默认不支持将规格参数变小，如需要将规格参数变小，您可以联系华为云客服，由华为云工程师给出分析评估后进行处理。
- 规格变更前，须确保实例状态正常。实例异常，节点异常，磁盘满均不允许进行规格变更。
- 高可用（1主2备）部署形态下，规格变更过程中会进行主备倒换，主备倒换过程中会有1min左右的业务中断。
- 单副本的部署形态下，规格变更过程中会进行中断重启，中断重启过程中会有5~10min的业务中断。
- 修改CPU/内存后，将会重启数据库实例。请选择业务低峰期，避免业务异常中断。重启后实例会自动释放内存中的缓存，请在业务低峰期进行重启，避免对高峰期业务造成影响。
- 高负载的情况下，规格变更时长会增加。

操作步骤

步骤1 **登录管理控制台**。

步骤2 单击管理控制台左上角的，选择区域和项目。



步骤3 在页面左上角单击☰，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择目标实例，单击“操作”列的“更多 > 规格变更”，进入“规格变更”页面。

您也可以通过单击目标实例名称，进入“基本信息”页面，在“数据库信息”模块的“性能规格”处，单击“规格变更”，进入“规格变更”页面。

步骤5 在“规格变更”页面，选择所需修改的性能规格，单击“下一步”。

步骤6 进行规格确认，单击“提交”。

步骤7 查看变更结果。

任务提交成功后，单击“返回实例列表”，在实例管理页面，可以看到实例状态为“规格变更中”。稍后在对应的“基本信息”页面，查看实例规格，检查修改是否成功。

----结束

8.10 修改切换策略

操作场景

GaussDB提供了数据高可靠和业务高可用两种故障切换策略，并支持在实例基本信息页面对切换策略进行修改。

说明

该功能仅针对特定用户开放，如需配置白名单权限，您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交开通白名单的申请。

注意事项

仅支持分布式版实例。

操作步骤

步骤1 **登录管理控制台**。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 单击指定实例的名称，进入实例基本信息页面。

步骤5 在实例基本信息页面的数据库信息模块，选择“切换策略”后的“修改”。

步骤6 在弹出框里选择切换策略。

- 数据高可靠：对数据一致性要求高的系统推荐选择数据高可靠，在故障切换的时候优先保障数据一致性。
- 业务高可用：对业务在线时间要求高的系统推荐使用业务高可用，在故障切换的时候优先保证数据库可用性。

须知

在业务高可用场景下需要谨慎修改如下数据库参数：

- recovery_time_target：不当修改该参数会导致实例频繁进行强制切换，请在技术人员指导下进行修改。
- audit_system_object：不当修改该参数会导致丢失DDL审计日志，请在技术人员指导下进行修改。

步骤7 单击“确认”，完成修改。

步骤8 修改该参数后需要手动重启实例才能生效，重启实例操作请参考**重启实例**。

重启实例前可以任意修改切换策略，实例重启中不允许修改。

----结束

8.11 设置安全组规则

操作场景

安全组是一个逻辑上的分组，为同一个虚拟私有云内具有相同安全保护需求，并相互信任的弹性云服务器和GaussDB实例提供访问策略。

如果账号已经申请创建时支持不指定安全组的白名单，则不需要执行本章节，而且在实例详情页也不会有内网安全组信息。

为了保障数据库的安全性和稳定性，在使用GaussDB实例之前，您需要设置安全组，开通需访问数据库的IP地址和端口。

- 内网连接GaussDB实例时，设置安全组分为以下两种情况：
  - ECS与GaussDB实例在相同安全组时，默认ECS与GaussDB实例互通，无需设置安全组规则。
  - ECS与GaussDB实例在不同安全组时，需要为GaussDB和ECS分别设置安全组规则。
    - 设置GaussDB安全组规则：为GaussDB所在安全组配置相应的**入方向规则**。
    - 设置ECS安全组规则：安全组默认规则为出方向上数据报文全部放行，此时，无需对ECS配置安全组规则。当在ECS所在安全组为非默认安全组且出方向规则**非全放通**时，需要为ECS所在安全组配置相应的**出方向**规则。
- 通过弹性公网IP连接实例时，需要为GaussDB所在安全组配置相应的**入方向规则**。

本节主要介绍如何为GaussDB实例设置相应的入方向规则。

关于添加安全组规则的详细要求，可参考《虚拟私有云用户指南》的“**添加安全组规则**”章节。

注意事项

因为安全组的默认规则是在出方向上的数据报文全部放行，同一个安全组内的弹性云服务器和GaussDB实例可互相访问。安全组创建后，您可以在安全组中定义各种访问规则，当GaussDB实例加入该安全组后，即受到这些访问规则的保护。

- 默认情况下，一个租户可以创建500条安全组规则。
- 建议一个安全组内的安全组规则不超过50条。
- 当需要从安全组外访问安全组内的GaussDB实例时，需要为安全组添加相应的**入方向规则**。
- 所有鲲鹏云服务器规格不支持配置不连续端口。

  如果您在鲲鹏云服务器中添加安全组规则时，使用了不连续端口号，那么除了该条规则不会生效，该规则后的其他规则也不会生效。比如：

  您先配置了安全组规则A（不连续端口号22，24），再配置了下一条安全规则B（独立端口号9096），则安全组规则A和B均不会生效。

- 出方向规则通常不适用于数据库实例。仅在数据库实例充当客户端时，出方向规则才适用。
- 数据库实例位于 VPC（虚拟私有云 Virtual Private Cloud）中但不可公开访问，则您还可以使用VPN连接。
- 创建分布式版实例时，如果需要修改内网安全组，请确保入方向规则TCP协议端口包含：40000-60480,20050,5000-5001,2379-2380,6000,6500，<database port> - (<database port> + 100)。（例如设置的数据库端口为8000，则安全组中需要包含8000-8100）。
- 创建主备版实例时，如果需要修改内网安全组，请确保入方向规则TCP协议端口包含：20050，5000-5001，2379-2380，6000，6500，<database port> - (<database port> + 100)。（例如设置的数据库端口为8000，则安全组中需要包含8000-8100）。

说明

为了保证数据及实例安全，请合理使用权限。建议使用最小权限访问，同时将可访问IP地址设置为远程主机地址或远程主机所在的最小子网地址，限制远程主机的访问范围。

源地址默认的IP地址0.0.0.0/0是指允许所有IP地址访问安全组内的GaussDB实例。

关于添加安全组规则的详细要求，可参考《虚拟私有云用户指南》的“**添加安全组规则**”章节。

操作步骤

步骤1 **登录管理控制台**。

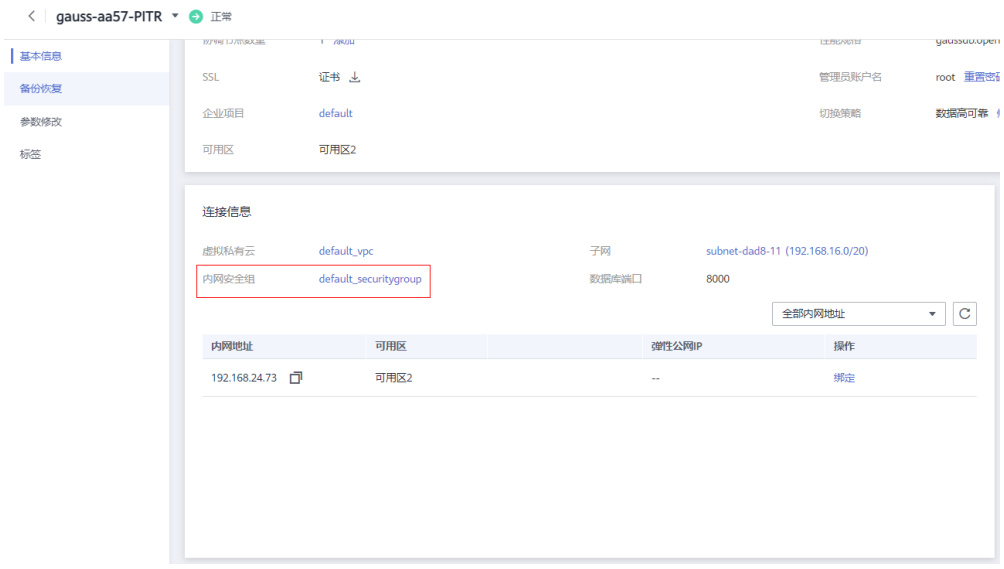
步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择目标实例，单击实例名称，进入实例的“基本信息”页面。

步骤5 设置安全组规则。

在“连接信息”模块的“内网安全组”处，单击安全组名称，进入安全组页面。



步骤6 在“入方向规则”子页签下单击“添加规则”，在“添加入方向规则”弹出框中填写安全组信息，单击“确定”。

单击“+”可以依次增加多条入方向规则。

表 8-2 入方向参数说明

参数	说明	取值样例
协议端口	网络协议。目前支持“All”、“TCP”、“UDP”、“ICMP”和“GRE”等协议。	自定义TCP
	端口：允许远端地址访问弹性云服务器指定端口，取值范围为：1～65535。常用端口请参见**弹性云服务器常用端口**。	**通过内网连接实例**时，输入已购买的弹性云服务器的目标实例的端口。

参数	说明	取值样例
类型	IP地址类型。 ● IPv4 ● IPv6	IPv4
源地址	源地址：可以是IP地址、安全组。例如： ● xxx.xxx.xxx.xxx/32（IPv4地址） ● xxx.xxx.xxx.0/24（子网） ● 0.0.0.0/0（任意地址）	0.0.0.0/0
描述	安全组规则的描述信息，非必填项。 描述信息内容不能超过255个字符，且不能包含“<”和“>”。	-

----结束

8.12 变更单副本实例部署形态

操作场景

GaussDB提供了将单副本实例部署形态变更为多副本实例部署形态的功能。

说明

该功能仅针对特定用户开放，如需配置白名单权限，您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交开通白名单的申请。

当前支持变更的部署形态：

表 8-3 部署形态

实例类型	部署形态	支持的底层资源类型	可以变更的目标部署形态
主备版	单副本	ECS	一主两备
		ECS	一主一备一日志

注意事项

- 目前支持的部署形态变更场景：
  - 主备版单副本实例变更为一主两备。
  - 主备版单副本实例变更为一主一备一日志。

- 变更部署形态前，确认实例状态为normal。
- 变更部署形态时，不支持磁盘扩容、规格变更、备份、重置密码、重启实例、删除实例等操作。
- 部署形态变更过程中，实例将出现业务中断，请在业务低峰期进行该操作。
- 变更部署形态后，新增节点的规格与原节点一致。日志节点规格使用日志节点配置规格。
- 主备版单副本实例变更为一主一备一日志形态后，副本一致性协议会变更为paxos。
- 变更部署形态后，会触发自动备份，开启关闭的归档日志。
- 8.0及以上版本实例支持单副本实例变更部署形态。
- 目前仅支持按需实例变更部署形态。

操作步骤

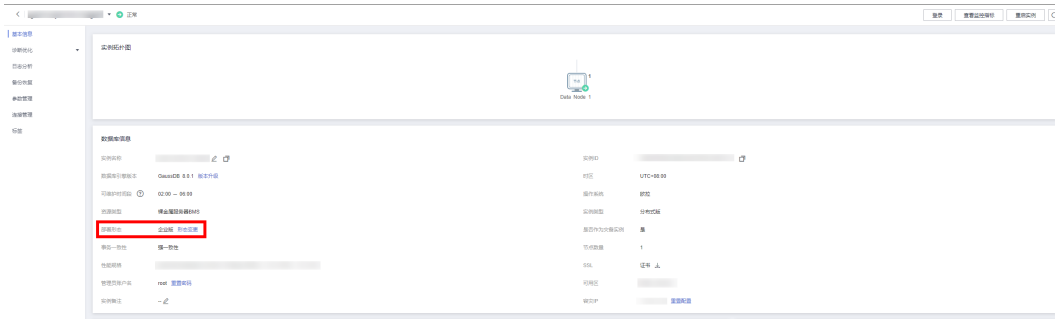
步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择目标实例，单击实例名称，进入实例的“基本信息”页面。

步骤5 在实例基本信息页面的“部署形态”处，单击“形态变更”。进入部署形态变更页面。



步骤6 在部署形态变更页面，选择变更后的部署形态和可用区，单击“下一步”。

图 8-8 变更部署形态页面

步骤7 确认详细信息。

- 如果需要重新选择，单击“上一步”，回到上个页面进行修改。
- 如果信息确认无误，单击“提交”，下发变更操作。

图 8-9 提交页面

任务提交成功后，在实例管理页面，可以看到实例状态为“形态变更中”。

----结束

变更后验证

变更完成后需要检查升级后的实例状态、备份创建、连接实例是否正常，能否进行正常的增加、删除、修改、查询操作。

步骤1 在“实例管理”页面，查看实例的运行状态是否为“正常”。

步骤2 在“实例管理”页面单击实例名称，进入基本信息页面，查看“节点列表”模块的“节点状态”是否为“正常”。

步骤3 检查备份创建是否正常。变更完成后系统会进行一次自动备份，检查备份创建是否正常。

1. 在“实例管理”页面，选择指定的实例，单击“实例名称”。
2. 在左侧导航栏中选择“备份恢复”，查看备份是否创建，且备份状态是否为“备份完成”。

步骤4 检查实例连接是否正常，是否能进行正常的增加、删除、修改、查询操作。

1. 参考**通过数据管理服务DAS连接实例**，登录到数据库。
2. 进入SQL查询页面。

图 8-10 SQL 查询

3. 创建数据库。
   CREATE DATABASE *数据库名*;
   以创建一个库名为db_tpcds的数据库为例：
   CREATE DATABASE db_tpcds;
   创建完db_tpcds数据库后，可以在左上方切换到新创建的库中。

图 8-11 切换数据库

4. 创建表，并进行增加、删除、修改、查询操作。
   a. 创建一个schema。
      CREATE SCHEMA *myschema*;
   b. 创建一个名称为mytable，只有一列的表。字段名为firstcol，字段类型为integer。
      CREATE TABLE myschema.mytable *(firstcol int)*;
   c. 向表中插入数据：
      INSERT INTO myschema.mytable values (100);
   d. 查看表中数据：
      SELECT * FROM myschema.mytable;

```
 | firstcol |
---+----------+
1 |   100    |
```

e. 修改表中数据：

UPDATE myschema.mytable SET firstcol = 200;

f. 再次查看表中数据：

SELECT * FROM myschema.mytable;

```
 | firstcol |
---+----------+
1 |   200    |
```

g. 删除表：

DROP TABLE myschema.mytable;

----结束

8.13 DN 主备倒换

操作场景

GaussDB实例状态正常时支持对分片内的DN做主备切换，可以选择同一分片内的备DN进行升主操作。

注意事项

- DN主备倒换可能会造成几秒或几分钟的服务闪断，请选择在业务低峰期进行操作。
- 实例节点状态为异常时不能执行该操作。
- 同一分片内只能指定一个备节点升主。
- 单节点实例不支持DN分片主备倒换操作。
- 主备倒换过程中，以下操作不可进行：
  - 实例重启
  - AZ切换
  - 规格变更
  - 节点修复
  - 节点替换
  - 节点扩容
  - 备份恢复

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击☰，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 单击目标实例名称，进入实例的“基本信息”页面。

步骤5 在“节点列表”模块，单击“DN主备倒换”。

可以选择可用区，筛选出主DN组件在该可用区上的DN分片。

- 如果该可用区没有主DN组件，则不显示分片信息。
- DN主备倒换可能会造成几秒或几分钟的服务闪断，请选择在业务低峰期进行操作。
- 一次最多支持对30个分片做主备切换。

步骤6 选择要升主的备DN组件，单击“确定”下发。

----结束

8.14 数据库引擎及操作系统更新

当前GaussDB服务数据库引擎及OS暂不支持租户侧维护窗口自助升级，如果需要升级，您可以联系华为云客服，由华为云工程师在给出升级分析评估后进行升级。

华为云仍然会通过热补丁方式及时修复对数据库引擎及操作系统影响重大的漏洞。

9 节点管理

9.1 重启节点

操作场景

GaussDB实例节点状态为非正常状态时，可通过执行重启节点操作尝试将节点状态恢复正常。当实例节点状态为正常状态时，也可执行重启节点操作。

注意事项

- 当数据库在以下操作过程中，可以执行重启节点操作：
  - 备份恢复失败
  - 按需转包周期中
  - 流容灾主实例容灾中
  - 流容灾主实例日志保持中
  - 流容灾灾备实例演练中
  - 流容灾灾备实例容灾中
  - 流灾备实例升主完成
- 重启过程中，节点不可用。
- 重启后节点会自动释放内存中的缓存，请在业务低峰期进行重启，避免对高峰期业务造成影响。
- 重启节点操作目前只支持主备版实例。
- 重启主节点后，会发生主备节点倒换。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的 ，选择区域和项目。

步骤3 在页面左上角单击 ☰，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 单击目标实例名称，进入实例的“基本信息”页面。

步骤5 在“节点列表”模块，单击“重启节点”，确认重启节点信息后，单击“确定”。

“重启节点”操作下发成功后，节点“运行状态”为“节点重启中”。

步骤6 稍后刷新实例基本信息，查看节点重启结果。如果节点运行状态为“正常”，说明节点重启成功。

----结束

10 数据备份

10.1 备份概述

GaussDB支持数据库实例的备份和恢复，以保证数据可靠性。备份目前将以未加密的方式存储。

备份存放在OBS桶，OBS备份恢复规格如下：

在华为云标准环境下全量备份恢复的性能规格为2T数据在8小时以内完成全量备份或全量恢复。

注意事项

备份期间xlog不回收。

备份的作用

当数据库或表被恶意或误删除，虽然GaussDB支持高可用，但备机数据库会被同步删除且无法还原。因此，数据被删除后只能依赖于实例的备份保障数据安全。

全量备份

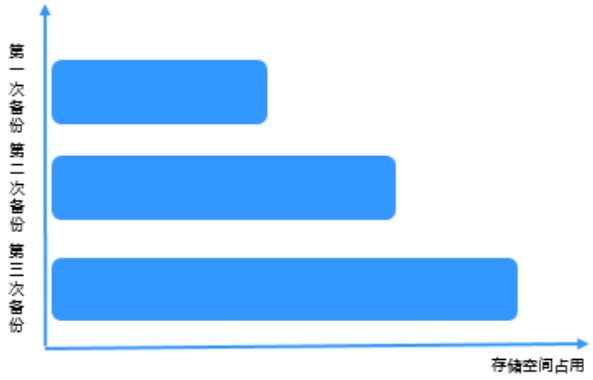
全量备份（Full Backup）表示对所有目标数据进行备份，包含备份时刻点上数据库的全量数据，耗时时间长（和数据库数据总量成正比），自身即可恢复出完整的数据库。全量备份总是备份所有选择的目标，即使从上次备份后数据没有变化。

差量备份

差量备份（Differential Backup）只包含从指定时刻点之后的增量修改数据，耗时时间短（和增量数据成正比，和数据总量无关），但是必须要和全量备份数据一起才能恢复出完整的数据库。GaussDB默认自动每30分钟对上一次自动备份后更新的数据进行备份，支持修改备份周期为最小15分钟，最大1440分钟。

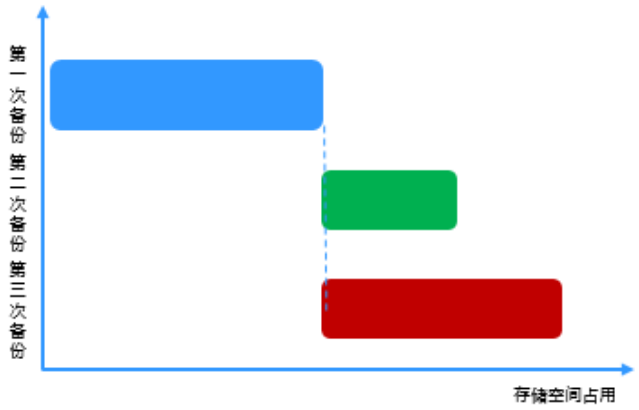
备份原理

图 10-1 全量备份示意图



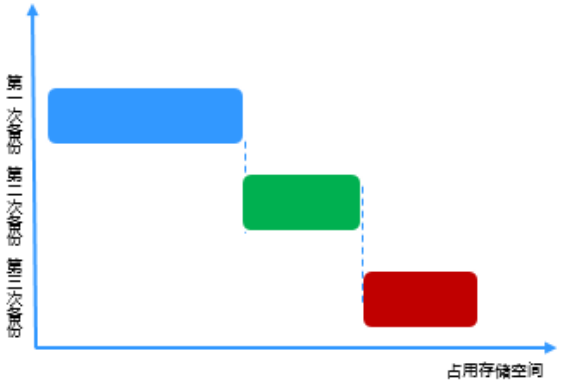
全量备份：第一次的全量备份后，无论数据是否变化，第二次备份和第三次备份都会将所有的数据全部进行备份。

图 10-2 差量备份示意图



差量备份：第一次的全量备份之后，第二次备份只会备份数据变化的数据，第三次备份会备份第一次全量备份后数据变化的数据。

图 10-3 增量备份示意图



增量备份：第一次的全量备份之后，第二次备份只会备份数据变化的数据，第三次备份只会备份第二次备份后数据变化的数据。

自动备份

GaussDB会在数据库实例的备份时段中创建数据库实例的自动备份。系统根据您指定的备份保留期保存数据库实例的自动备份。扩容实例CN或者分片后，系统会进行一次自动备份。

手动备份

用户还可以创建手动备份对数据库进行备份，手动备份是由用户启动的数据库实例的全量备份，会一直保存，直到用户手动删除。

10.2 设置实例级自动备份策略

操作场景

创建GaussDB实例时，系统默认开启实例级自动备份策略。实例创建成功后，您可根据业务需要修改实例级自动备份策略。GaussDB按照用户设置的自动备份策略对数据库进行备份。

GaussDB默认开启的自动备份策略设置如下：

- 保留天数：默认为7天。保留天数范围为1～732天。需要延长保留时间请联系客服人员申请，自动备份最长可以申请保留2562天。
- 备份时间段：默认为24小时中，间隔一小时的随机的一个时间段 ，例如01:00～02:00，12:00～13:00等。备份时间段以UTC时区保存。如果碰到夏令时/冬令时切换，备份时间段会因时区变化而改变。
- 备份周期：默认为一周内的每一天。
- 差量备份策略：默认每30分钟保存一次。
- 备份流控：默认75MB/s。
- 差量预取页面个数：默认64。

说明

为了满足时间点恢复的需求，超出备份保留天数最近的一次全量备份不会被立即删除。示例：设置自动备份策略为每天备份1次，保留天数为1天，即11.1号生成备份1，11.2号生成备份2并保留备份1；11.3号生成备份3，并保留备份2及删除备份1。

约束限制

- 全量备份时不允许重启数据库，请谨慎选择备份时间段。
- GaussDB单副本实例3.0以下版本不支持设置实例级自动备份策略。

计费说明

备份都是以压缩包的形式存储在对象存储服务上。具体收费规则请参见**GaussDB的备份是如何收费的**。

修改自动备份策略

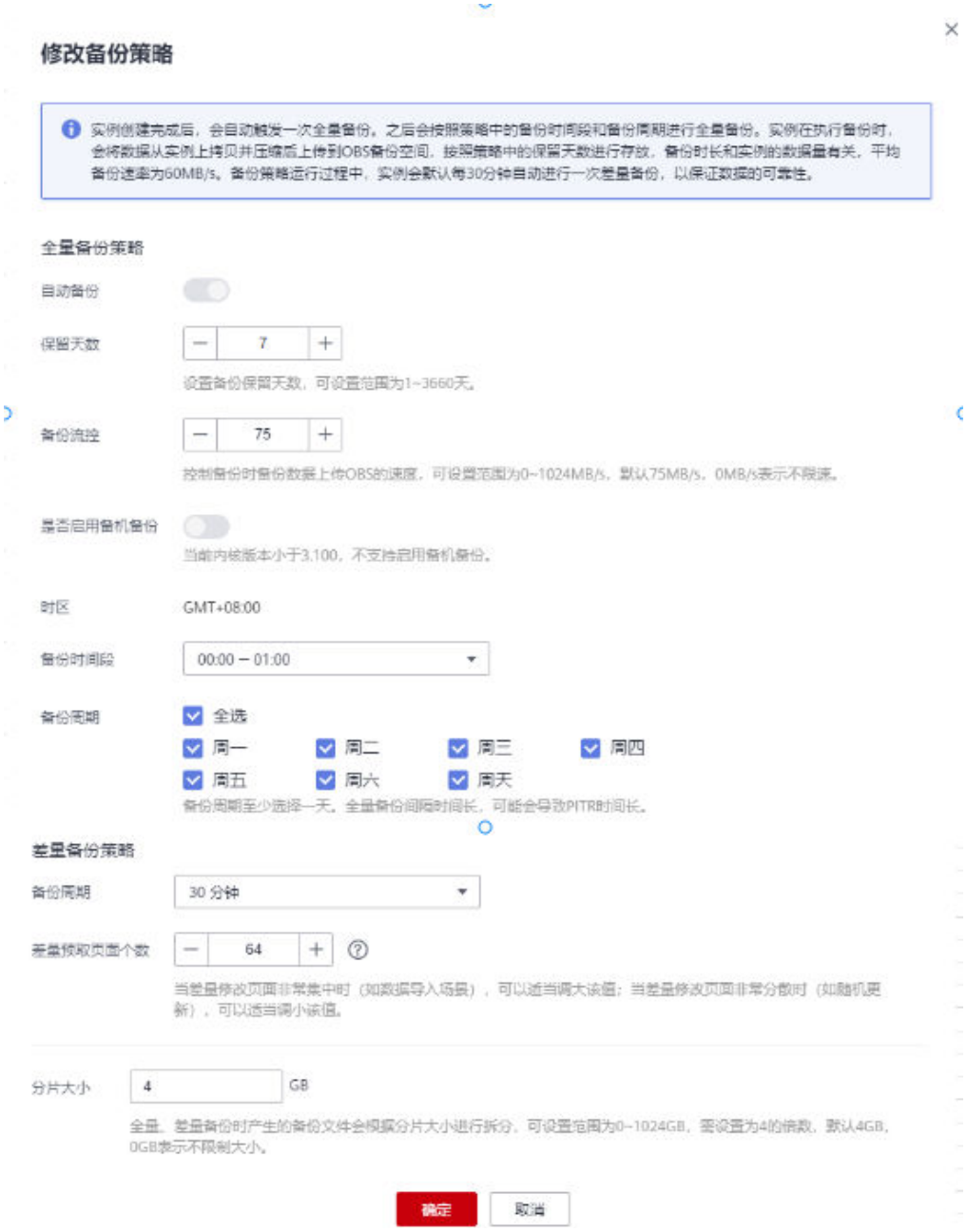
步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击实例名称。

步骤5 在左侧导航栏，选择“备份恢复”，单击“修改备份策略”。您可以查看到已设置的备份策略，如需修改备份策略，请调整以下参数的值。



步骤6 按照界面提示修改实例级备份策略。

- 实例级全量备份策略
  - 保留天数：保留天数是指自动备份可保留的时间，增加保留天数可提升数据可靠性，请根据需要设置，默认7天。对于系统中最近一个全量备份文件，如果在新的全量备份未超过保留天数前系统会一直保留，直至新的全量备份超过保留天数后才会删除。
    - 增加保留天数，可提升数据可靠性，请根据需要设置。
    - 减少保留天数，会针对已有的备份文件生效，即超出备份保留天数的已有备份文件（包括全量备份和增量备份）会被自动删除，但手动备份不会自动删除，请您谨慎选择。

全量备份文件自动删除策略：

已有备份文件超出备份天数后会自动删除，考虑到数据完整性，自动删除时仍然会保留最近的一次超过保留天数的全量备份，保证在保留天数内的数据可正常恢复。

假如备份周期选择“周一”、“周二”，保留天数设置为“2”，备份文件的删除策略如下：

- 本周一产生的全量备份，会在本周四当天自动删除。原因如下：

  本周二的全量备份在本周四当天超过保留天数，按照全量备份文件自动删除策略，会保留最近的一个超过保留天数的全量备份（即本周二的备份会被保留），因此周四当天删除本周一产生的全量备份文件。

- 本周二产生的全量备份，会在下周三当天自动删除。原因如下：

  下周一产生的全量备份在下周三超过保留天数，按照全量备份文件自动删除策略，会保留最近的一个超过保留天数的全量备份（即下周一的备份会被保留），因此下周三当天删除本周二产生的全量备份。

- 备份流控：控制备份时备份数据上传OBS的速度，默认75M/s，0表示不限速。
- 是否启用备机备份：如果启动备机备份，实例全量备份、差量备份在备DN所在主机进行备份。
- 备份时间段：默认为24小时中，间隔一小时的随机的一个时间段 ，例如01:00～02:00，12:00～13:00等。备份时间段以UTC时区保存。如果碰到夏令时/冬令时切换，备份时间段会因时区变化而改变。

图 10-4 修改备份策略

图 10-5 自定义备份时间段

  - 备份周期：请根据需要进行选择，并且最少需要选择一天。

    说明

    保留天数范围为1～732天。需要延长保留时间请联系客服人员申请，自动备份最长可以申请保留2562天。

    备份时间段为间隔1小时，建议根据业务情况，选择业务低峰时段，备份周期默认全选，可修改，且至少选择一周中的1天。

    实例创建完成后，会立即触发一次全量备份，之后会按照策略中的备份时间段和备份周期进行全量备份和差量备份策略。备份时间段请选择为业务峰值较低的时间段。全量备份会在此时间段进行。

- 实例级差量备份策略
  - 备份周期：需要选择差量备份的周期，即每隔多长时间进行一次差量备份。默认30分钟一次。
  - 差量预取页面个数：控制差量备份时读取磁盘上表文件增量修改页面的预取页面个数，默认64。当差量修改页面非常集中时（如数据导入场景），可以适当调大该值；当差量修改页面非常分散时（如随机更新），可以适当调小该值。当调大差备预取页面个数时，差备在读取磁盘上表文件的预取页面会变多，所占用的IO变大，此时会影响其他业务，导致数据库性能有一定的下降。
  - 分片大小：全量、差量备份时产生的备份文件会根据分片大小进行拆分，可设置范围为0~1024GB，需设置为4的倍数，默认4GB，0GB表示不限制大小。

步骤7 单击“确定”，确认修改。

----结束

10.3 创建实例级手动备份

操作场景

GaussDB支持对运行正常的实例创建实例级手动备份，用户可以通过手动备份恢复数据，从而保证数据可靠性。

注意事项

- 手动备份是由用户启动的数据库实例的全量备份，会一直保存，直到用户手动删除。
- 备份操作需要在实例状态为正常时才可以进行。
- 同一用户在一个实例上，同一时间只能进行一次实例级备份操作。
- 账户余额大于等于0元，才可创建实例级手动备份。
- GaussDB单副本实例3.0以下版本不支持创建实例级手动备份。

计费说明

备份都是以压缩包的形式存储在对象存储服务上。具体收费规则请参见**GaussDB的备份是如何收费的**。

当数据库实例被删除后，实例赠送的备份空间会自动取消，此时手动备份会按照占用空间大小按需收费，详见**产品价格详情**。

方式一

步骤1 **登录管理控制台**。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定的实例，在操作列选择“更多 > 创建备份”。

步骤5 在创建备份弹出框中，命名该备份，并添加描述，单击“确定”，提交备份创建，单击“取消”，取消创建。

- 备份名称的长度在4~64个字符之间，必须以字母开头，区分大小写，可以包含字母、数字、中划线或者下划线，不能包含其他特殊字符。
- 备份描述不能超过256字符，且不能包含回车和>!<"&'=特殊字符。

- 手动备份创建过程中，状态显示为“备份中”，此过程所需时间由数据量大小决定。

页面长时间未刷新，可单击页面右上  刷新页面，查看实例是否备份完成。若实例状态为正常，备份完成，执行**步骤6**。

步骤6 手动备份创建成功后，用户可在“备份恢复管理”页面，对其进行查看并管理。

也可在“实例管理”页面，单击实例名称，在左侧导航栏，单击“备份恢复”，对其进行查看并管理。

----结束

方式二

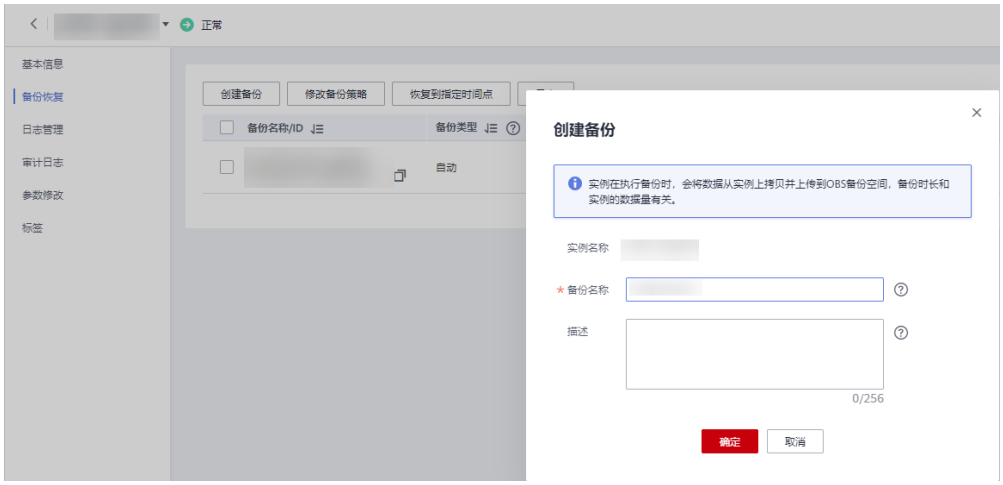
步骤1 **登录管理控制台**。

步骤2 单击管理控制台左上角的  ，选择区域和项目。

步骤3 在页面左上角单击  ，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击实例名称。

步骤5 在左侧导航栏中选择“备份恢复”，单击“创建备份”。



步骤6 命名该备份，并添加描述，单击“确定”，提交备份创建，单击“取消”，取消创建。

- 备份名称的长度在4~64个字符之间，必须以字母开头，区分大小写，可以包含字母、数字、中划线或者下划线，不能包含其他特殊字符。

- 备份描述不能超过256字符，且不能包含回车和>!<"&'=特殊字符。
- 手动备份创建过程中，状态显示为“备份中”，此过程所需时间由数据量大小决定。

图 10-6 创建备份

步骤7 手动实例级备份创建成功后，用户可在“实例管理”页面，单击实例名称，在左侧导航栏中选择“备份恢复”，对其进行查看并管理。

也可在“备份恢复管理”页面，对其进行查看并管理。

----结束

10.4 导出备份信息

操作场景

GaussDB支持导出备份，用户可以通过导出备份功能将备份信息（ID，备份名称，实例名称，实例ID，引擎，备份类型，备份开始时间，备份结束时间，备份状态，备份大小，备份描述）导出到csv并下载，方便用户查看并分析备份信息。

注意事项

GaussDB单副本实例3.0以下版本不支持导出备份信息。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在左侧导航栏，单击“备份恢复管理”，在“备份恢复管理”页面，勾选需要导出的备份，单击，导出备份信息。

说明

只可导出当前页面的备份，不可跨页面导出。

您也可以在“实例管理”页面，单击实例名称，进入“基本信息”页面，在左侧导航栏，单击“备份恢复”，勾选需要导出的备份，单击“导出”，导出备份信息。

导出的备份信息列表为csv文件，您可以对其进行分析，以满足业务需求。

步骤5 查看导出的数据库备份。

----结束

10.5 停止备份

操作场景

GaussDB支持对实例级的备份任务进行停止，包括自动全量备份、手动全量备份和差量备份。

说明

该功能仅针对特定用户开放，如需配置白名单权限，您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交开通白名单的申请。

注意事项

停止备份操作会停止该实例所有进行中的全量备份和差量备份。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定实例，单击实例名称，进入“基本信息”页面。

步骤5 在左侧导航栏，选择“备份恢复”，然后单击“停止备份”。

步骤6 单击“确定”，停止备份。

步骤7 在“任务中心”页面，选择目标任务，查看任务信息。

须知

只有2.8及以上版本的实例执行停止备份操作后可通过“任务中心”查看停止备份任务信息。

----结束

10.6 删除手动备份

操作场景

GaussDB支持对实例级和表级手动备份进行删除，从而释放相关存储空间。

须知

- 手动备份删除后，不可恢复。
- 自动备份的文件不可手动删除。
- 恢复中的备份不允许删除。
- 要删除备份，您必须登录到拥有备份的账户。
- GaussDB单副本实例3.0以下版本不支持删除手动备份。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击 ☰，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在左侧导航栏，单击“备份恢复管理”，在“备份恢复管理”页面，选择目标备份，单击操作列中的“删除”。

您也可以在“实例管理”页面，单击实例名称，进入“基本信息”页面，在左侧导航栏，单击“备份恢复”，勾选需要删除的备份，单击操作列的“删除”。

步骤5 单击“是”，删除手动备份。

备份删除后，将不会在备份恢复管理界面展示。

步骤6 若您已开启高危操作保护，在“身份验证”弹出框中单击“获取验证码”，正确输入验证码并单击“确定”，页面自动关闭。

通过进行二次认证再次确认您的身份，进一步提高账号安全性，有效保护您安全使用云产品。关于如何开启操作保护，具体请参考**《统一身份认证服务用户指南》**的内容。

----结束

11 数据恢复

11.1 通过备份文件恢复实例

操作场景

GaussDB支持使用已有的实例级自动备份和手动备份，将实例数据恢复到备份被创建时的状态。该操作恢复的为整个实例的数据。

当前支持恢复到新实例、已有实例和当前实例。

限制条件

- 账户余额大于等于0元，才可恢复到新实例。
- 恢复时目标实例异常、实例磁盘满将会导致恢复失败。
- 不支持跨大版本恢复。例如：1.4.x的实例仅可以恢复到1.4.y版本的实例。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在左侧导航栏单击“备份恢复管理”，选择需要恢复的备份，单击操作列的“恢复”。

您也可在“实例管理”页面，单击指定的实例名称，在左侧导航栏单击“备份恢复”，在“全量备份”页签下选择实例级备份，单击目标备份对应的操作列中的“恢复”。

步骤5 单击“确定”，恢复实例。

说明

- 如果打开并行恢复功能，那么恢复过程中，所有主、备副本会同时从OBS服务器下载备份数据，与默认的串行恢复相比，OBS带宽消耗量增加到N倍（N等于每个分片的副本个数）。因此，为了防止OBS带宽达到上限导致恢复速度反而下降的情况，当待恢复集群的分片个数大于5个时，建议先咨询运维当前OBS服务器空闲带宽，然后再决定是否开启并行恢复功能。
- 主备版实例只支持并行恢复。
- 数据库内核版本小于1.4时，不支持开启并行恢复。
- 全量备份和增量备份除了备份数据文件之外，也会备份这个过程中的增量日志文件，用于保证该备份集恢复以后数据的一致性。由于增量日志文件的备份和上传需要一定时间（受网络、OBS存储介质流控等影响），因此，需要注意的是，备份结束时间并不代表该备份集恢复后的数据一致性时间点（该恢复一致性点一般在备份结束时刻之前的几分钟以内）。如果用户对于恢复后数据的一致性时刻点有严格要求，请使用指定时间点恢复。

- 恢复到新实例：
  - 数据库大版本与原实例相同。例如：1.4.x的实例仅可以恢复到1.4.y版本的实例。
  - 存储空间大小默认和备份时实例磁盘空间相同，且必须大于或等于备份时实例存储空间大小。
  - 数据库密码需重新设置。
  - 新实例的规格默认和原实例相同，如果需要修改规格，新实例的规格必须大于或等于原实例的规格。
  - 新实例的节点配置需要与备份时保持一致。

  填写完新实例的基本信息后，单击“立即申请”。
- 恢复到当前实例：

  - 恢复时的实例要与备份时实例版本号和节点配置相同。
  - 选择该选项时，会将原实例上的数据全部覆盖，且恢复过程中数据库不可用。
  - 建议先进行手动备份后再进行恢复操作。
  - 如果使用开启高级压缩特性之前的备份恢复到当前实例，需要重新开启高级压缩特性。
- 恢复到已有实例：
  - 恢复时的实例要与备份时实例版本号和节点配置相同。
  - 选择该选项时，会将目标实例上的数据全部覆盖，且恢复过程中数据库不可用。
  - 建议先对目标实例进行手动备份后再进行恢复操作。

步骤6 查看恢复结果。

- 恢复到新实例

  为用户重新创建一个和该备份数据相同的实例。可看到实例由“创建中”变为“正常”，说明恢复成功。

  恢复成功的新实例是一个独立的实例，与原有实例没有关联。
- 恢复到当前实例

  在“实例管理”页面，可查看目标实例状态为“恢复中”，恢复完成后，实例状态由“恢复中”变为“正常”。恢复完成后系统会自动进行一次实例级全量备份。

  恢复完成后，检查恢复数据与要恢复到的时间点一致，在实例备份恢复界面，单击“数据确认”。在单击数据确认前，可多次进行恢复。单击数据确认后会删除本次恢复时间点后的归档日志，并重新开启日志归档。
- 恢复到已有实例

  在“实例管理”页面，可查看目标实例状态为“恢复中”，恢复完成后，实例状态由“恢复中”变为“正常”。恢复完成后系统会自动进行一次实例级全量备份。

----结束

11.2 恢复实例到指定时间点

操作场景

GaussDB支持使用已有的实例级自动备份，恢复实例数据到指定时间点。

GaussDB支持将备份恢复到新实例、当前实例和已有实例。

注意事项

- 恢复到任意时间点仅支持2.1版本以上实例，单副本实例暂不支持。
- 节点扩容，版本升级，恢复自身期间，对应时间点无法恢复。
- 实例故障，发生CN剔除等场景无法产生归档日志，对应时间点无法恢复。
- 如果您要将数据库备份恢复到新实例：
  - 数据库引擎、数据库大版本，与原实例相同，不可修改。

  - 数据库密码需重新设置。
- 恢复到当前实例会将当前实例上的数据全部覆盖，并且恢复过程中数据库不可用，且立即停止归档。恢复完成后会出现数据确认按钮，在单击数据确认前，可多次进行恢复。数据确认后会删除本次恢复时间点后的归档日志，并重新开启日志归档。
- 删除实例会默认删除所有归档日志，不支持选择保留。重建后不支持恢复任意时间点。
- 账户余额大于等于0元，才可恢复到新实例。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击实例名称。

步骤5 在左侧导航栏中选择“备份恢复”页签，单击“恢复到指定时间点”。

步骤6 单击“确定”，恢复实例。

说明

- 如果打开并行恢复功能，那么恢复过程中，所有主、备副本会同时从OBS服务器下载备份数据，与默认的串行恢复相比，OBS带宽消耗量增加到N倍（N等于每个分片的副本个数）。因此，为了防止OBS带宽达到上限导致恢复速度反而下降的情况，当待恢复集群的分片个数大于5个时，建议先咨询运维当前OBS服务器空闲带宽，然后再决定是否开启并行恢复功能。
- 主备版实例只支持并行恢复。
- 数据库内核版本小于1.4时，不支持开启并行恢复。

- 恢复到新实例：
  - 数据库大版本与原实例相同。例如：1.4.x的实例仅可以恢复到1.4.y版本的实例。
  - 存储空间大小默认和备份时实例磁盘空间相同，且必须大于或等于备份时实例存储空间大小。
  - 数据库密码需重新设置。
  - 新实例的规格默认和原实例相同，如果需要修改规格，新实例的规格必须大于或等于原实例的规格。
  - 新实例的节点配置需要与备份时保持一致。

  填写完新实例的基本信息后，单击“立即申请”。
- 恢复到当前实例：
  - 恢复时的实例要与备份时实例版本号和节点配置相同。
  - 选择该选项时，会将原实例上的数据全部覆盖，且恢复过程中数据库不可用。
  - 建议先进行手动备份后再进行恢复操作。
  - 如果使用开启高级压缩特性之前的备份恢复到当前实例，需要重新开启高级压缩特性。
- 恢复到已有实例：
  - 恢复时的实例要与备份时实例版本号和节点配置相同。
  - 选择该选项时，会将目标实例上的数据全部覆盖，且恢复过程中数据库不可用。
  - 建议先对目标实例进行手动备份后再进行恢复操作。

步骤7 查看恢复结果。

- 恢复到新实例

  为用户重新创建一个和该备份数据相同的实例。可看到实例由“创建中”变为“正常”，说明恢复成功。

  恢复成功的新实例是一个独立的实例，与原有实例没有关联。
- 恢复到当前实例

  在“实例管理”页面，可查看目标实例状态为“恢复中”，恢复完成后，实例状态由“恢复中”变为“正常”。恢复完成后系统会自动进行一次全量备份。

  恢复完成后，检查恢复数据与要恢复到的时间点一致，在实例备份恢复界面，单击“数据确认”。在单击数据确认前，可多次进行恢复。单击数据确认后会删除本次恢复时间点后的归档日志，并重新开启日志归档。
- 恢复到已有实例

在“实例管理”页面，可查看目标实例状态为“恢复中”，恢复完成后，实例状态由“恢复中”变为“正常”。恢复完成后系统会自动进行一次全量备份。

----结束

12 账号和网络安全

12.1 重置管理员密码

操作场景

在使用GaussDB过程中，如果忘记数据库root账号密码，可以重新设置密码。

注意事项

- 如果您提供的密码被系统视为弱密码，您将收到错误提示，请提供更高强度的密码。
- 如果数据库实例处于“异常”状态，则无法重置管理员密码。
- 重置密码生效时间取决于该实例当前执行的业务数据量。
- 请定期修改用户密码，以提高系统安全性，防止出现密码被暴力破解等安全风险。
- 账号被冻结时不可重置密码。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击“更多 > 重置密码”。

您也可以在“基本信息”页签，在“数据库信息”模块的“管理员账户名”处，单击“重置密码”。

步骤5 在“重置密码”弹框，输入新密码及确认密码。

图 12-1 重置密码

须知

重置的密码需满足以下几个条件：

- 8到32个字符。
- 至少包含大写字母（A-Z），小写字母（a-z），数字（0-9），非字母数字字符（限定为~!@#%^*-_=+?,）四类字符中的三类字符。
- 新密码不能与旧密码相同或相反。

- 单击“确定”，提交重置。
- 单击“取消”，取消本次重置。

步骤6 若您已开启高危操作保护，在弹出框单击“去验证”，跳转至验证页面，单击“免费获取验证码”，正确输入验证码并单击“认证”，页面自动关闭。

通过进行二次认证再次确认您的身份，进一步提高账号安全性，有效保护您安全使用云产品。关于如何开启操作保护，具体请参考**《统一身份认证服务用户指南》**的内容。

----结束

12.2 企业项目配额管理

GaussDB在华为云的管理控制台提供配额管理的功能，可以对租户下的企业项目进行配额管理。

只有账号为企业账号，并配置白名单后，才可以使用配额管理功能。如需配置白名单权限，您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交开通白名单的申请。

管理配额

步骤1 **登录管理控制台**。

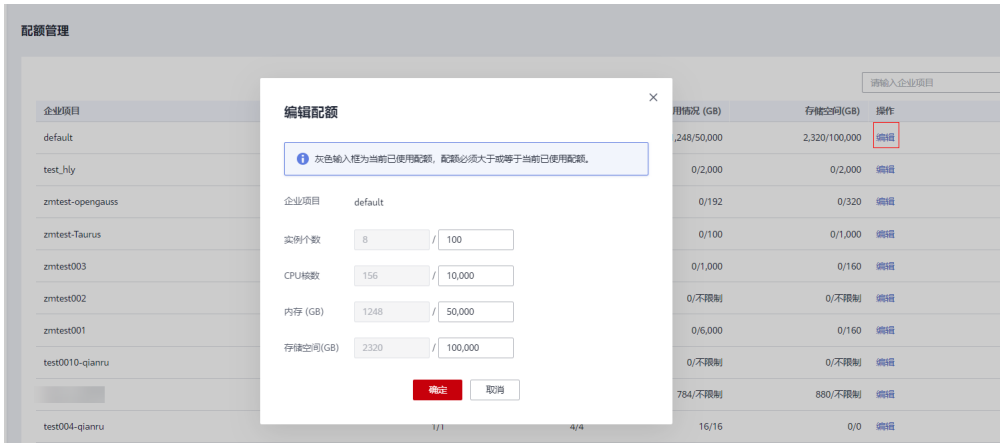
步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在左侧导航栏，单击“配额管理”，进入“配额管理”界面。

可以在此界面查看每个项目的实例使用情况、CPU使用情况、内存使用情况、存储空间信息。

步骤5 选择需要管理的企业项目，单击“操作”列的“编辑”。



说明

首次进入配额管理页面时，此处显示为“设置”。

步骤6 在弹出的对话框中填写需要变更的配额数量。单击“确认”。

----结束

13 参数模板管理

13.1 创建参数模板

您可以使用数据库参数模板中的参数来管理数据库引擎配置，数据库参数模板可应用于一个或多个数据库实例。

如果您在创建数据库实例时未指定客户创建的数据库参数模板，系统将会为您的数据库实例适配默认的数据库参数模板。该默认组包含数据库引擎默认值和系统默认值，具体根据引擎、计算规格及实例的分配存储空间而定。您无法修改默认数据库参数模板的参数设置，您必须创建自己的数据库参数模板才能更改参数设置的默认值。

须知

并非所有数据库引擎参数都可在客户创建的数据库参数模板中进行更改。

如果您想使用您自己的数据库参数模板，只需创建一个新的数据库参数模板，创建实例的时候选择该参数模板，如果是在创建实例后有这个需求，可以重新应用该参数模板，请参见**应用参数模板**。

若您已成功创建数据库参数模板，并且想在新的数据库参数模板中包含该组中的大部分自定义参数和值时，复制参数模板是一个方便的解决方案，请参见**复制参数模板**。

以下是您在使用数据库参数模板中的参数时应了解的几个要点：

- 当您修改当前实例的参数模板并保存后，仅应用于当前实例，不会对其他实例造成影响。
- 某些参数需要重启才能生效，这些参数更改将在您手动重启数据库实例后生效。
- 在数据库参数模板内设置参数不恰当可能会产生意外的不利影响，包括性能降低和系统不稳定。修改数据库参数时应始终保持谨慎，禁止对参数组进行边界测试，否则会导致实例异常，且修改数据库参数模板前要备份数据。将参数模板更改应用于生产数据库实例前，您应当在测试数据库实例上试用这些参数模板设置更改。

说明

GaussDB和文档数据库服务不共享参数模板配额。
每个项目最多可以创建100个GaussDB数据库参数模板，各GaussDB引擎共享该配额。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 单击左侧导航栏的“参数模板管理”。

步骤5 在“参数模板管理”页面，单击“创建参数模板”。

步骤6 选择数据库引擎版本，命名并添加对该参数模板的描述，单击“确定”，创建参数模板。

- 选择该数据库引擎参数模板所需应用的参数模板类型。
- 参数模板名称长度在1~64个字符之间，区分大小写，可包含字母、数字、中划线、下划线或句点，不能包含其他特殊字符。
- 参数模板的描述长度不能超过256个字符，且不能包含回车和! < " = ' > &特殊字符。

----结束

13.2 修改 GaussDB 实例参数

为确保GaussDB发挥出最佳性能，用户可根据业务需求对用户创建的参数模板里边的参数进行调整。

您可以修改用户创建的数据库参数模板中的参数值，但不能更改默认数据库参数模板中的参数值。

以下是您在使用数据库参数模板中的参数时应了解的几个要点：

- 如果您单击实例名称，在“参数修改”页面修改当前实例的参数模板，更改动态参数并保存数据库参数模板时，系统将立即应用更改，而不管“应用”设置如何。当您更改静态参数并保存数据库参数模板时，参数更改将在您手动重启该数据库实例后生效。
- 当您在“参数模板管理”页面，批量修改参数模板时，需执行“应用”操作，才会对实例生效。当您更改静态参数并保存数据库参数模板时，参数更改将在您应用到实例后，手动重启与数据库参数模板关联的数据库实例后生效。应用参数模板到数据库实例，请参见应用参数模板。

如果您更改一个参数值，则所做更改的应用时间将由该参数的类型决定，可以根据界面提示确认参数是否重启生效。

说明

系统提供的默认参数模板不允许修改，只可单击参数模板名进行查看。当用户参数设置不合理导致数据库无法启动时，可参考默认参数模板重新配置。

批量修改参数

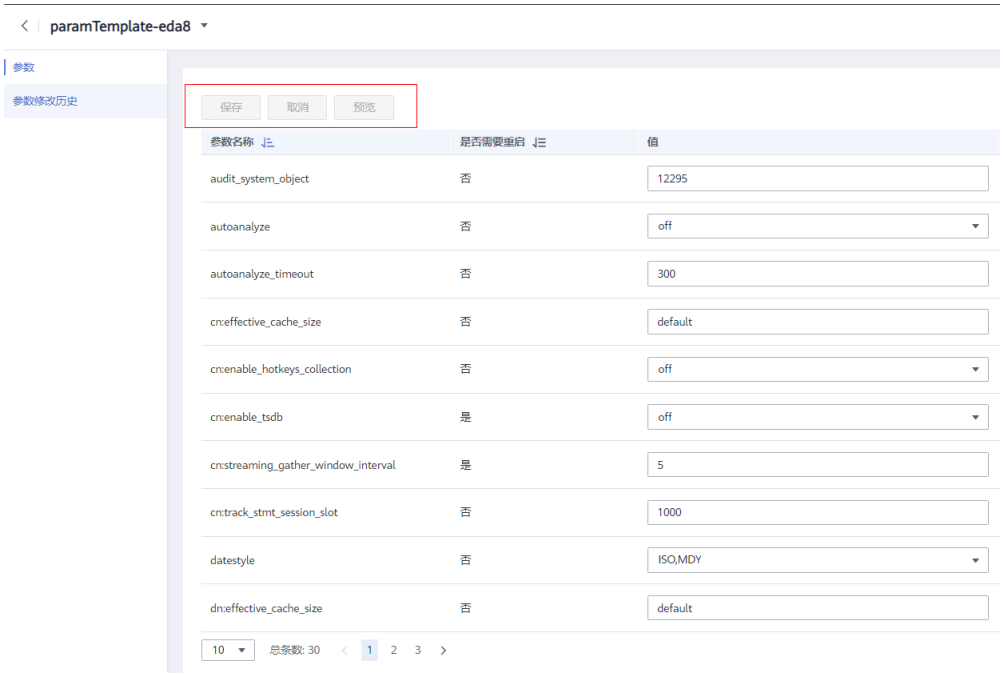
步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“参数模板管理”页面的“自定义”页签，选择目标参数模板，单击参数模板名称。

步骤5 根据需要修改相关参数值。



- 单击“保存”，在弹出框中单击“是”，保存修改。
- 单击“取消”，放弃本次设置。
- 单击“预览”，可对比参数修改前和修改后的值。

步骤6 参数修改完成后，您可在“参数模板管理”页面单击目标参数模板名称，然后在左侧导航栏中，单击“参数修改历史”查看参数的修改详情。

须知

参数模板修改后，不会立即应用到当前使用的实例，您需要进行应用操作才可生效，具体操作请参见**应用参数模板**。

----结束

13.3 导出参数

操作场景

您可以将该实例对应的参数模板信息（参数名称，值，描述）导出到CSV中，方便查看并分析。

操作步骤

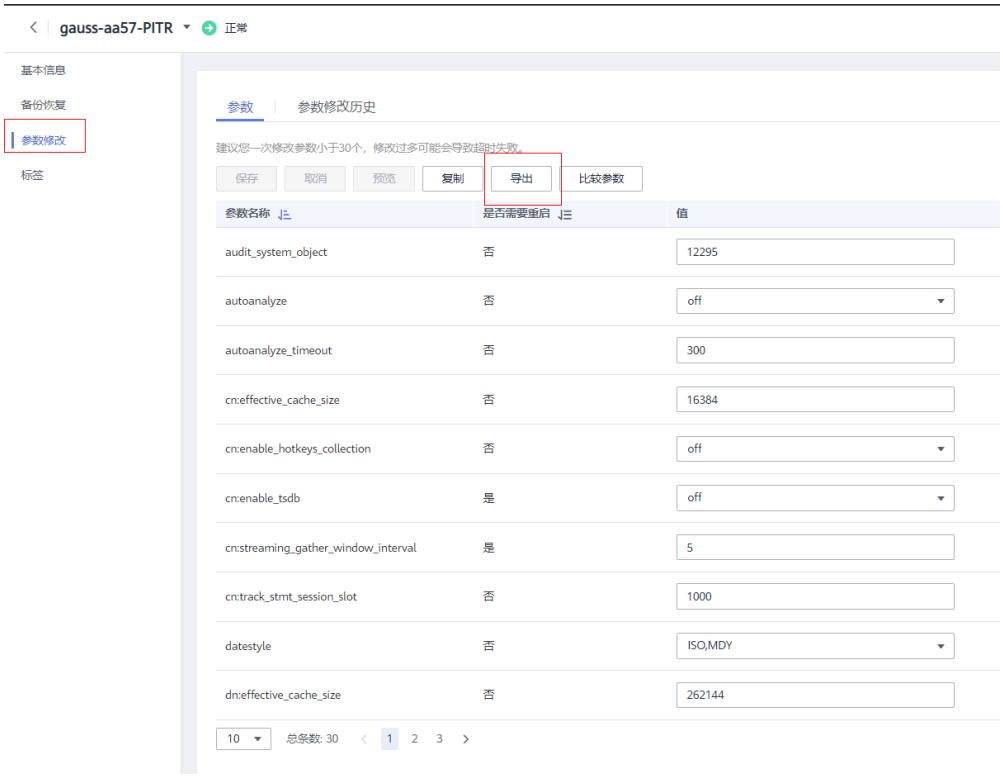
步骤1 **登录管理控制台**。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击实例名称，进入实例的基本信息页面。

步骤5 在左侧导航栏中选择“参数修改”，在“参数”页签单击“导出”。



导出到文件。将该实例对应的参数模板信息（参数名称，值，描述）导出到CSV表中，方便用户查看并分析。

步骤6 在弹出框中，填写文件名称，单击“确定”。

说明

文件名称在4位到81位之间，必须以字母开头，可以包含字母、数字、中划线或下划线，不能包含其他特殊字符。

----结束

13.4 比较参数模板

操作场景

- 您可以将当前实例应用的参数与参数模板进行比较，以了解当前实例参数的差异项。
- 您可以比较GaussDB默认参数模板，以了解默认参数模板与其他参数模板的配置差异。
- 你还可以比较自定义的参数模板，以区分不同参数模板之间的差别。

比较当前实例参数模板

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击☰，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，单击实例名称，进入实例的“基本信息”页签。

步骤5 在左侧导航栏中选择“参数修改”。

步骤6 在“参数”子页签中单击“比较参数”，比较当前实例参数。

步骤7 在弹出框中选择与当前实例同数据库类型的参数模板，单击“确定”，比较两个参数的差异项。

- 有差异项，则会显示差异参数的如下信息：参数名称、当前实例参数模板的参数值和被比较参数模板的参数值。
- 无差异项，则不显示。

----结束

比较目标参数模板

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击☰，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“参数模板管理”页面的“系统默认”或者“自定义”页签，选择一个参数模板，单击“比较”。

步骤5 选择同一数据库引擎的不同参数模板，单击“确定”，比较两个参数模板之间的配置参数差异项。

- 有差异项，则会显示差异参数模板的如下信息：参数名称、当前参数模板的参数值和被比较参数模板的参数值。
- 无差异项，则不显示。

----结束

13.5 查看参数修改历史

操作场景

您可以查看当前实例所使用参数模板的修改历史，以满足业务需要。

您也可以查看自定义参数模板的修改历史，以满足业务需要。

说明

用户创建或复制的新参数模板，在未进行参数修改前，无修改历史。

当前仅显示7天之内的参数修改历史。

查看当前实例的参数修改历史

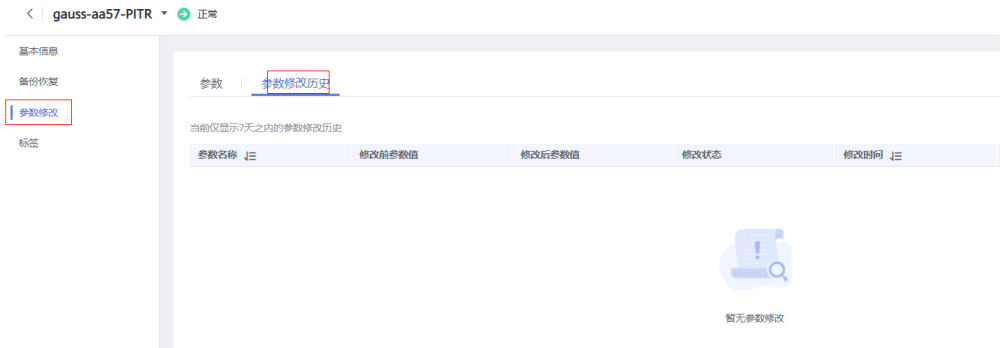
步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击实例名称，进入实例的基本信息页面。

步骤5 在左侧导航栏，单击“参数修改”。



步骤6 在弹出的页签中，单击“参数修改历史”。

您可查看参数对应的参数名称、修改前参数值、修改后参数值、修改状态、修改时间、是否应用以及应用时间。

如修改后参数模板未应用，请根据业务需要，参考**应用参数模板**，将其应用到对应实例。

----结束

查看目标参数模板的参数修改历史

步骤1 **登录管理控制台**。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“参数模板管理”页面的“自定义”页签，单击目标参数模板名称。

步骤5 单击“参数修改历史”。

您可查看参数对应的参数名称、修改前参数值、修改后参数值、修改状态和修改时间。

----结束

13.6 复制参数模板

操作场景

您可以复制您创建的自定义数据库参数模板。当您已创建一个数据库参数模板，并且想在新的数据库参数模板中包含该组中的大部分自定义参数和值时，复制参数模板是一个方便的解决方案。您还可以复制某数据库实例应用的参数列表，生成一个新的参数模板，供您后期使用。

复制数据库参数模板之后，新参数模板可能不会立即显示，建议您等待至少5分钟再使用。

您无法复制默认参数模板。不过，您可以创建基于默认参数模板的新参数模板。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“参数模板管理”页面的“自定义”页签，选择需要复制的参数模板，单击“复制”。

您还可以在页面，单击实例名称，在左侧导航栏，单击“参数修改”，单击“复制”，将该实例对应参数列表生成一个参数模板，供您后期使用。

步骤5 在弹出框中，填写新参数模板名称和描述，单击“确定”。

- 参数模板名称长度在1~64个字符之间，区分大小写，可包含字母、数字、中划线、下划线或句点，不能包含其他特殊字符。
- 参数模板的描述长度不能超过256字符，且不能包含回车和>!<"&'=特殊字符。

创建完成后，会生成一个新的参数模板，您可在参数模板列表中对其进行管理。

----结束

13.7 重置参数模板

操作场景

您可根据自己的业务需求，重置自己创建的参数模板对应的所有参数，使其恢复到默认值。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“参数模板管理”页面的“自定义”页签，选择需要设置的参数模板，单击“更多 > 重置”。

步骤5 单击“是”，重置所有参数为其默认值。

说明

有关参数模板状态，请参见**状态**中的参数模板状态内容。

对于某些参数模板重置后，您需在实例列表中，查看状态，如果显示参数模板变更，等待重启，则需重启关联的实例使之生效。

----结束

13.8 应用参数模板

操作场景

参数模板编辑修改后，不会立即应用到实例，您可以根据业务需要应用到实例中，参数模板只能应用于相同版本的实例中。

操作步骤

步骤1 **登录管理控制台**。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“参数模板管理”页面，根据参数模板类型不同进行如下操作。

- 若需要将默认参数模板应用到实例，在“系统默认”页签的目标参数模板操作列，单击“应用”。
- 若需要将用户自己创建的参数模板应用到实例，在“自定义”页签的目标参数模板操作列，单击“更多 > 应用”。

一个参数模板可被应用到一个或多个实例。

步骤5 在弹出框中，选择或输入所需应用的实例，单击“确定”。

参数模板应用成功后，您可参考**查看参数模板应用记录**，查看应用状态是否为“成功”。如果应用状态为“应用中”，则再次应用该模板时，将无法选择正在应用参数模板的实例。如果需要再次应用参数模板到同一实例，请确保应用状态为“成功”。

说明

参数模板应用成功后，如果重启生效的参数有修改，实例将会变为“等待重启”状态，此时需要重启实例参数修改才能生效。如果所有重启生效的参数都没有修改，则实例状态不会发生改变。

----结束

13.9 查看参数模板应用记录

操作场景

参数模板编辑修改后，您可根据业务需要将其应用到对应实例中，GaussDB支持查看参数模板所应用到实例的记录。

操作步骤

步骤1 **登录管理控制台**。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“参数模板管理”页面，“系统默认”页签或“自定义”页签，选择目标参数模板，单击操作列的“应用记录”，查看应用记录。

步骤5 您可查看参数模板所应用到的实例名称/ID、应用状态、应用时间、失败原因。

----结束

13.10 修改参数模板描述

操作场景

参数模板创建成功后，用户可根据需要对自己创建的参数模板描述进行修改。

说明

默认参数模板的描述不可修改。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“参数模板管理”页面的“自定义”页签，选择一个用户创建的参数模板，单击“描述”列。

步骤5 输入新的描述信息，单击，提交修改，单击，取消修改。

- 修改成功后，新的新描述信息，可在参数模板列表的“描述”列查看。
- 参数模板的描述长度不能超过256字符，且不能包含>!<"&'=特殊字符。

----结束

13.11 删除参数模板

操作场景

每个用户最多可以创建100个数据库参数模板，如果参数模板已满或者一些参数模板已经没有使用价值，您可以参考本章内容删除已有的参数模板。

须知

- 参数模板删除后，不可恢复，请谨慎操作。
- 默认参数模板不可被删除。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“参数模板管理”页面的“自定义”页签，选择需要删除的参数模板，单击“更多 > 删除”。

步骤5 单击“是”，删除。

----结束

14 监控与告警

14.1 监控指标一览表

功能说明

本节定义了GaussDB上报云监控的监控指标的命名空间，监控指标列表和维度定义，用户可以通过云监控提供的API接口来检索GaussDB产生的监控指标和告警信息。

命名空间

SYS.GAUSSDBV5

指标采集约束

- 分布式备DN：3.100.0及以上版本实例开始支持采集，且需要事务一致性为"最终一致性"。
- 主备版备DN：2.0.10及以上版本实例开始支持采集。

支持的监控指标

GaussDB数据库性能监控指标，如下表所示。

表 14-1 GaussDB 支持的监控指标

指标ID	指标名称	指标含义	展示对象	指标单位	测量对象	监控周期(原始指标)
rds001_cpu_util	CPU使用率	该指标用于统计测量对象的CPU使用率。	当前节点	%	节点	60秒
rds002_mem_util	内存使用率	该指标用于统计测量对象的内存使用率。	当前节点	%	节点	60秒

指标ID	指标名称	指标含义	展示对象	指标单位	测量对象	监控周期(原始指标)
rds003_bytes_in	数据写入量	该指标用于统计测量对象对应VM的网络发送字节数，取时间段的平均值。	当前节点	Byte/s	节点	60秒
rds004_bytes_out	数据传出量	该指标用于统计测量对象对应VM的网络接受字节数，取时间段的平均值。	当前节点	Byte/s	节点	60秒
rds014_iops	数据磁盘每秒读写次数	该指标用于统计测量对象的节点数据磁盘每秒读写次数，该值为实时值。	当前节点	Count/s	节点	60秒
rds016_disk_write_throughput	数据磁盘写吞吐量	该指标用于统计测量对象的节点数据磁盘每秒写吞吐量，该值为实时值。	当前节点	Byte/s	节点	60秒
rds017_disk_read_throughput	数据磁盘读吞吐量	该指标用于统计测量对象的节点数据磁盘每秒读吞吐量，该值为实时值。	当前节点	Byte/s	节点	60秒
rds020_avg_disk_ms_per_write	数据磁盘单次写入花费的时间	该指标用于统计测量对象的节点数据磁盘单次写入花费的时间，取时间段的平均值。	当前节点	ms	节点	60秒
rds021_avg_disk_ms_per_read	数据磁盘单次读取花费的时间	该指标用于统计测量对象的节点数据磁盘单次读取花费的时间，取时间段的平均值。	当前节点	ms	节点	60秒

指标ID	指标名称	指标含义	展示对象	指标单位	测量对象	监控周期(原始指标)
io_bandwidth_usage	磁盘io带宽占用率	当前磁盘io带宽与磁盘最大带宽比值。	当前节点	%	节点	60秒
iops_usage	IOPS使用率	当前IOPS与磁盘最大IOPS比值。	当前节点	%	节点	60秒
rds005_instance_disk_used_size	实例数据磁盘已使用大小	该指标用于统计测量对象的实例数据磁盘已使用大小，该值为实时值。	实例	GB	实例	60秒
rds006_instance_disk_total_size	实例数据磁盘总大小	该指标用于统计测量对象的实例数据磁盘总大小，该值为实时值。	实例	GB	实例	60秒
rds007_instance_disk_usage	实例数据磁盘已使用百分比	该指标用于统计测量对象的实例数据磁盘使用率，该值为实时值。	实例	%	实例	60秒
rds035_buffer_hit_ratio	buffer命中率	该指标用于统计数据库buffer命中率。	实例	%	实例	60秒
rds036_deadlocks	死锁次数	该指标用于统计数据库发生事务死锁的次数，取该时间段的增量值。	实例	Count	实例	60秒
rds048_P80	80% SQL的响应时间	该指标用于统计数据库80% SQL的响应时间，该值为实时值。	实例	us	实例	60秒
rds049_P95	95% SQL的响应时间	该指标用于统计数据库95% SQL的响应时间，该值为实时值。	实例	us	实例	60秒

指标ID	指标名称	指标含义	展示对象	指标单位	测量对象	监控周期(原始指标)
rds008_disk_used_size	磁盘已使用大小	该指标用于统计测量对象的节点数据磁盘使用值，该值为实时值。	当前节点	GB	组件	60秒
rds009_disk_total_size	磁盘总大小	该指标用于统计测量对象的节点数据磁盘总大小，该值为实时值。	当前节点	GB	组件	60秒
rds010_disk_usage	磁盘已使用百分比	该指标用于统计测量对象的节点数据磁盘使用率，该值为实时值。	当前节点	%	组件	60秒
rds024_current_sleep_time	主机流控时间	该指标用于统计测量对象的主机流控时间，该值为实时值。	分布式：备DN 主备版：备DN	s	组件	60秒
rds025_current_rto	备机RTO时间	该指标用于统计测量对象的主备复制的RTO，该值为实时值。	分布式：备DN 主备版：备DN	s	组件	60秒
rds026_login_counter	用户登入次数/秒	该指标用于统计每秒的登入次数，取时间段的平均值。	分布式：所有CN 主备版：主DN	Count/s	组件	60秒
rds027_logout_counter	用户登出次数/秒	该指标用于统计每秒的登出次数，取时间段的平均值。	分布式：所有CN 主备版：主DN	Count/s	组件	60秒

指标ID	指标名称	指标含义	展示对象	指标单位	测量对象	监控周期(原始指标)
rds028_standby_delay	备机redo进度	该指标用于统计分片内备机redo进度，表示备机和主机的差距，该值为实时值。	分布式：备DN 主备版：主DN	Byte	组件	60秒
rds030_wait_ratio	锁等待状态会话比率	该指标用于统计当前处于锁等待状态会话占活跃工作状态下会话比率，该值为实时值。	分布式：所有CN+主DN 主备版：所有DN	%	组件	60秒
rds031_active_ratio	活跃会话率	该指标用于统计当前处于活跃工作状态会话占总会话数比率，该值为实时值。	分布式：所有CN+主DN 主备版：所有DN	%	组件	60秒
rds034_inuse_counter	CN连接数	该指标用于统计CN连接池中正在使用的连接数，该值为实时值。	分布式：所有CN 主备版：不采集	Count	组件	60秒
rds037_commit_counter	用户提交事务数/秒	该指标用于统计用户每秒提交的事务数，取时间段的平均值。	分布式：所有CN 主备版：主DN	Count/s	组件	60秒
rds038_rollback_counter	用户回滚事务数/秒	该指标用于统计用户每秒回滚的事务数，取时间段的平均值。	分布式：所有CN 主备版：主DN	Count/s	组件	60秒

指标ID	指标名称	指标含义	展示对象	指标单位	测量对象	监控周期(原始指标)
rds039_bg_commit_counter	后台提交事务数/秒	该指标用于统计后台每秒提交的事务数，取时间段的平均值。	分布式：所有CN 主备版：主DN	Count/s	组件	60秒
rds040_bg_rollback_counter	后台回滚事务数/秒	该指标用于统计后台每秒回滚的事务数，取时间段的平均值。	分布式：所有CN 主备版：主DN	Count/s	组件	60秒
rds041_resp_avg	用户事务平均响应时间	该指标用于统计用户事务的平均响应时间。	分布式：所有CN 主备版：主DN	us	组件	60秒
rds042_rollback_ratio	用户事务回滚率	该指标用于统计用户事务回滚事务占用户提交、回滚事务之和的比率，取时间段的平均值。	分布式：所有CN 主备版：主DN	%	组件	60秒
rds043_bg_rollback_ratio	后台事务回滚率	该指标用于统计后台事务回滚事务占用户提交、回滚事务之和的比率，取时间段的平均值。	分布式：所有CN 主备版：主DN	%	组件	60秒
rds044_ddl_count	Data Definition Language/s	该指标用于统计用户负载在query层的DDL数量，取时间段的平均值。	分布式：所有CN+所有DN 主备版：所有DN	Count/s	组件	60秒

指标ID	指标名称	指标含义	展示对象	指标单位	测量对象	监控周期(原始指标)
rds045_dml_count	Data Manipulation Language/s	该指标用于统计用户负载在query层的DML数量，取时间段的平均值。	分布式：所有CN+所有DN 主备版：所有DN	Count/s	组件	60秒
rds046_dcl_count	Data Control Language/s	该指标用于统计用户负载在query层的DCL数量，取时间段的平均值。	分布式：所有CN+所有DN 主备版：所有DN	Count/s	组件	60秒
rds047_ddl_dcl_ratio	DDL+DCL比率	该指标用于统计用户负载在query层的DDL+DCL占DDL+DCL+DML的比率，取时间段的平均值。	分布式：所有CN+所有DN 主备版：所有DN	%	组件	60秒
rds050_ckpt_delay	待落盘的数据量	该指标用于统计信息同步到磁盘过程中待落盘的数据量，该值为实时值。	分布式：所有CN+主DN 主备版：主DN	Byte	组件	60秒
rds051_phyrds	读物理文件的IO次数/秒	该指标用于统计数据库每秒读物理物件的IO次数，取时间段的平均值。	分布式：所有CN+主DN 主备版：所有DN	Count/s	组件	60秒

指标ID	指标名称	指标含义	展示对象	指标单位	测量对象	监控周期(原始指标)
rds052_phywrts	写物理文件的IO次数/秒	该指标用于统计数据库每秒写物理物件的IO次数，取时间段的平均值。	分布式：所有CN+主DN 主备版：所有DN	Count/s	组件	60秒
rds053_online_session	在线会话数量	该指标用于统计当前在线的session个数，该值为实时值。	分布式：所有CN+所有DN 主备版：所有DN	Count	组件	60秒
rds054_active_session	活跃会话数量	该指标用于统计当前所有活跃工作状态下会话个数，该值为实时值。	分布式：所有CN+主DN 主备版：主DN	Count	组件	60秒
rds055_online_ratio	在线会话率	该指标用于统计CN（分布式）/主DN（主备版）上的在线会话比例，该值为实时值。	分布式：所有CN+主DN 主备版：所有DN	%	组件	60秒
rds060_long_running_transaction_exectime	数据库最长事务的执行时长	该指标用于统计测量对象的数据库最长事务的执行时长,该值为实时值。	分布式：所有CN+主DN 主备版：所有DN	s	组件	60秒
rds066_replication_slot_wal_log_size	复制槽保留的WAL日志大小	该指标用于统计主DN上复制槽中保留的WAL日志的大小,该值为实时值。	分布式：主DN 主备版：所有DN	Byte	组件	60秒

指标ID	指标名称	指标含义	展示对象	指标单位	测量对象	监控周期(原始指标)
rds067_xlog_lsn	xlog速率	该指标用于统计CN或者主DN上xlog的速率, 该值为实时值。	分布式：所有CN+主DN 主备版：主DN	Byte/s	组件	60秒
rds068_swap_used_ratio	交换内存使用率	该指标用于描述操作系统交换内存使用率，该值为实时值。	当前节点	%	节点	60秒
rds069_swap_total_size	交换内存总大小	该指标用于描述操作系统交换内存总大小，该值为实时值。	当前节点	MB	节点	60秒
rds070_thread_pool	线程池使用率	该指标用于统计CN和DN的线程池使用率，该值为实时值。	分布式：所有CN+主DN 主备版：所有DN	%	组件	60秒
rds071_locks_session	等锁会话数	该指标用于统计CN/主DN的等锁会话数，该值为实时值。	分布式：所有CN+主DN 主备版：所有DN	Count	组件	60秒
rds072_streaming_dr_xlog_gap	灾备集群分片日志差距	该指标用于统计流式容灾特性开启情况下，灾备集群中各个分片相对于生产集群的日志差距。	分布式：所有CN+主DN 主备版：主DN	Byte	组件	60秒

指标ID	指标名称	指标含义	展示对象	指标单位	测量对象	监控周期(原始指标)
rds073_streaming_dr_xlog_to_be_replayed	灾备集群分片待回放日志量	该指标用于统计流式容灾特性开启情况下，灾备集群中各个分片待回放日志量。	分布式：所有CN+主DN 主备版：主DN	Byte	组件	60秒
rds074_streaming_dr_xlog_flushing_rate	灾备集群分片日志落盘速率	该指标用于统计流式容灾特性开启情况下，灾备集群中各个分片日志落盘速率。	分布式：所有CN+主DN 主备版：主DN	Byte/s	组件	60秒
rds075_streaming_dr_xlog_replay_rate	灾备集群分片日志回放速率	该指标用于统计流式容灾特性开启情况下，灾备集群中各个分片日志回放速率。	分布式：所有CN+主DN 主备版：主DN	Byte/s	组件	60秒
rds076_streaming_dr_rpo	分片RPO	该指标用于统计流式容灾特性开启情况下，各个分片的实时RPO。	分布式：所有CN+主DN 主备版：主DN	s	组件	60秒
rds077_streaming_dr_rto	分片RTO	该指标用于统计流式容灾特性开启情况下，各个分片的实时RTO。	分布式：所有CN+主DN 主备版：主DN	s	组件	60秒
rds078_inactive_replication_slot	非活跃的复制槽数	该指标用于统计非活跃的复制槽数(物理加逻辑)。	分布式：所有CN+主DN 主备版：所有DN	Count	组件	60秒

指标ID	指标名称	指标含义	展示对象	指标单位	测量对象	监控周期(原始指标)
rds079_standy_not_replayed_log	只读节点未回放日志量	该指标用于查询只读节点日志回放与接收量差距。	分布式：备DN 主备版：备DN	Byte	组件	60秒
rds080_xlog_num	xlog数量	该指标用于统计CN和DN数据目录下xlog数量，该值为实时值。	分布式：所有CN+所有DN 主备版：所有DN	Count	组件	60秒
rds081_xlog_size	xlog大小	该指标用于统计CN和DN数据目录下xlog大小，该值为实时值。	分布式：所有CN+所有DN 主备版：所有DN	MB	组件	60秒
rds064_dynamic_used_memory	已使用动态内存	该指标用于统计测量对象的动态内存已使用大小，该值为实时值。	分布式：所有CN+所有DN 主备版：所有DN	MB	组件	60秒
rds065_dynamic_used_memory_usage	动态内存使用率	该指标用于统计测量对象的动态内存使用率，该值为实时值。	分布式：所有CN+所有DN 主备版：所有DN	%	组件	60秒

指标ID	指标名称	指标含义	展示对象	指标单位	测量对象	监控周期(原始指标)
rds061_idle_in_transaction_num	空闲事务个数	该指标用于统计测量对象的空闲事务连接的数量，该值为实时值。	分布式：所有CN+所有DN 主备版：所有DN	Count	组件	60秒
rds062_slowquery_sys	系统库慢SQL数量	该指标用于统计指定周期内主DN/CN上系统数据库慢SQL数量，该值为实时值。	分布式：所有CN 主备版：主DN	Count	组件	60秒
rds063_slowquery_user	用户库慢SQL数量	该指标用于统计指定周期内主DN/CN上用户库慢SQL数量，该值为实时值。	分布式：所有CN 主备版：主DN	Count	组件	60秒
rds082_gaussv5_wait_session	等待会话数量	该指标用于统计当前等待会话数量，该值为实时值。	分布式：所有CN + 备DN 主备版：所有DN	Count	组件	60秒
rds083_cn_temp_dir_size	CN临时目录大小	该指标用于统计CN数据目录下临时目录大小，该值为实时值。	分布式：所有CN + 备DN 主备版：所有DN	MB	组件	60秒
rds084_sys_database_size	系统数据库大小	该值用于统计实例的postgres数据库大小，该值为实时值。	当前节点	Byte	节点	60秒
rds085_user_database_size	用户数据库总大小	该值用于统计实例的所有用户数据库总大小，该值为实时值。	当前节点	Byte	节点	60秒

指标ID	指标名称	指标含义	展示对象	指标单位	测量对象	监控周期(原始指标)
rds086_select_distribution	select分布	该值用于统计select语句的比例，该值为实时值。	分布式：所有CN + 所有DN 主备版：所有DN	%	组件	60秒
rds087_update_distribution	update分布	该值用于统计update语句的比例，该值为实时值。	分布式：所有CN + 所有DN 主备版：所有DN	%	组件	60秒
rds088_insert_distribution	insert分布	该值用于统计insert语句的比例，该值为实时值。	分布式：所有CN + 所有DN 主备版：所有DN	%	组件	60秒
rds089_delete_distribution	delete分布	该值用于统计delete语句的比例，该值为实时值。	分布式：所有CN + 所有DN 主备版：所有DN	%	组件	60秒
rds090_gaussdbv5_statement	statement数量	该值用于统计唯一SQL数量，该值为实时值。	分布式：所有CN + 所有DN 主备版：所有DN	Count	组件	60秒
rds091_gaussv5_qps	读请求量	该值用于统计租户的每秒读请求数量，取时间段内的平均值。	分布式：所有CN 主备版：所有DN	Count	组件	60秒

指标ID	指标名称	指标含义	展示对象	指标单位	测量对象	监控周期(原始指标)
rds092_gaussv5_tps_rt_insert	insert写请求响应时间	该值用于统计租户的insert写请求平均响应时间，取时间段内的平均值。	分布式：所有CN 主备版：所有DN	ms	组件	60秒
rds093_gaussv5_tps_rt_update	update写请求响应时间	该值用于统计租户的update写请求平均响应时间，取时间段内的平均值。	分布式：所有CN 主备版：所有DN	ms	组件	60秒
rds094_gaussv5_tps_rt_delete	delete写请求响应时间	该值用于统计租户的delete写请求平均响应时间，取时间段内的平均值。	分布式：所有CN 主备版：所有DN	ms	组件	60秒
rds095_gaussv5_qps_rt	读请求响应时间	该值用于统计租户的读请求平均响应时间，取时间段内的平均值。	分布式：所有CN 主备版：所有DN	ms	组件	60秒
retrans_rate	重传比例	该值用于统计TCP 包重传率，该值为实时值。	当前节点	%	节点	60秒
rds096_process_used_memory	进程已使用内存	该指标用于统计CN或者DN上已经使用内存，该值为实时值。	分布式：所有CN + 所有DN 主备版：所有DN	MB	组件	60秒
rds097_2pc_transaction_prepare	最长未决事务存活时长	该指标用于统计2pc未提交事务最长时间。	主备版：主DN	s	组件	60秒

指标ID	指标名称	指标含义	展示对象	指标单位	测量对象	监控周期(原始指标)
rds098_dn_instance_status	DN组件状态	该指标用于表示DN组件状态，该值为实时值，1代表正常(Primary)，2代表正常(Standby)，3代表正常(Main Standby)，4代表正常(Cascade Standby)，10代表Catchup(备机追主机xlog)，20代表备机：连接正常，复制异常，21代表连接异常。	主备版：所有DN	无	组件	60秒
rds099_replication_slot_dir_size	复制插槽目录大小	该值用于统计复制插槽目录大小，该值为实时值。	主备版：所有DN	KB	组件	300秒
rds100_standby_diff_redo_and_receive	备机redo位置和接收位置差距	该指标用于查询备机redo位置与备机接收位置差距，以判断主备差异是因为备机回放慢还是主机未发送。	分布式：备DN 主备版：备DN	Byte	组件	60秒
rds101_online_distinct_client_addr_count	在线客户端数量	该指标用于查看每个CN上的在线客户端数量。	分布式：所有CN	Count	组件	60秒

指标ID	指标名称	指标含义	展示对象	指标单位	测量对象	监控周期(原始指标)
rds102_working_distinct_client_addr_count	活跃客户端数量	该指标用于查看每个CN上的活跃客户端连接数。	分布式：所有CN	Count	组件	60秒

维度

表 14-2 GaussDB 涉及的维度

Key	Value
gaussdbv5_instance_id	GaussDB实例
gaussdbv5_node_id	GaussDB节点
gaussdbv5_component_id	GaussDB组件

14.2 支持的事件列表

表 14-3 云数据库 GaussDB

事件来源	事件名称	事件ID	事件级别	事件说明	处理建议	事件影响
GaussDB	进程状态告警	ProcessStatusAlarm	重要	GaussDB关键进程退出，包括：CMS/CMA、ETCD、GTM、CN、DN。	等待进程自动恢复或者自动主备切换，观察业务是否恢复。如果业务未恢复，您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交工单。	主机进程故障，在主机上进行的业务将中断回滚。备机进程故障不影响业务。

事件来源	事件名称	事件ID	事件级别	事件说明	处理建议	事件影响
	组件状态告警	ComponentStatusAlarm	重要	GaussDB关键组件无响应，包括：CMA、ETCD、GTM、CN、DN。	等待进程自动恢复或者自动主备切换，观察业务是否恢复。如果业务未恢复，您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交工单。	主机进程无响应，在主机上进行的业务将无响应。备机进程故障不影响业务。
	集群状态告警	ClusterStatusAlarm	重要	集群状态异常，包括： 集群只读、ETCD多数派故障、集群分布不均衡。	您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交工单。	集群只读： 业务只读。 ETCD多数派故障：集群不可用。 集群分布不均衡：集群性能/可靠性降低。
	硬件资源告警	HardwareResourceAlarm	重要	集群中出现严重的硬件故障，包括：磁盘损坏、GTM网络通信故障。	您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交工单。	业务部分/全部受损。
	状态转换告警	StateTransitionAlarm	重要	集群出现如下重要事件：DN build/build失败、DN强切、DN主备切换/failover、GTM主备切换/failover。	等待自动恢复，观察业务是否恢复。如果业务未恢复，您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交工单。	部分业务受损。
	其他异常告警	OtherAbnormalAlarm	重要	磁盘使用阈值告警等。	关注业务变化，及时计划扩容。	超过使用阈值，将无法扩容。

事件来源	事件名称	事件ID	事件级别	事件说明	处理建议	事件影响
	实例运行状态异常	TaurusInstanceRunningStatusAbnormal	重要	由于灾难或者物理机故障导致实例故障时，会上报该事件，属于关键告警事件。	您可以在管理控制台右上角，选择“工单 > 新建工单”，提交工单。	可能导致数据库服务不可用。
	实例运行状态异常已恢复	TaurusInstanceRunningStatusRecovered	重要	针对灾难性的故障，GaussDB有高可用工具会自动进行恢复或者手动恢复，执行完成后会上报该事件。	不需要处理。	无
	节点运行状态异常	TaurusNodeRunningStatusAbnormal	重要	由于灾难或者物理机故障导致数据库节点故障时，会上报该事件，属于关键告警事件。	检查数据库服务是否可以正常使用，并在管理控制台右上角，选择“工单 > 新建工单”，提交工单。	可能导致数据库服务不可用。
	节点运行状态异常已恢复	TaurusNodeRunningStatusRecovered	重要	针对灾难性的故障，GaussDB有高可用工具会自动进行恢复或者手动恢复，执行完成后会上报该事件。	不需要处理。	无
	创建实例业务失败	GaussDBV5CreateInstanceFailed	重要	创建实例失败产生的事件，一般是配额大小不足，底层资源耗尽导致。	先释放不再使用的实例再尝试重新发放，或者您可以在管理控制台右上角，选择“工单 > 新建工单”，提交工单调整配额上限。	无法创建数据库实例。

事件来源	事件名称	事件ID	事件级别	事件说明	处理建议	事件影响
	添加节点失败	GaussDBV5ExpandClusterFailed	重要	一般是由于底层资源不足等原因导致。	您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交工单协调资源，删除添加失败的节点，重新尝试添加新节点。	无
	存储扩容失败	GaussDBV5EnlargeVolumeFailed	重要	一般是由于底层资源不足等原因导致。	您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交工单协调资源再重试扩容操作。	如果磁盘满，会导致业务中断。
	重启失败	GaussDBV5RestartInstanceFailed	重要	一般是由于网络问题等原因导致。	重试重启操作或在管理控制台右上角，选择“**工单 > 新建工单**”，提交工单。	可能导致数据库服务不可用。
	全量备份失败	GaussDBV5FullBackupFailed	重要	一般是备份文件导出失败或上传失败等原因导致。	您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交工单。	无法备份数据。
	差量备份失败	GaussDBV5DifferentialBackupFailed	重要	一般是备份文件导出失败或上传失败等原因导致。	您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交工单。	无法备份数据。

事件来源	事件名称	事件ID	事件级别	事件说明	处理建议	事件影响
	删除备份失败	GaussDBV5DeleteBackupFailed	重要	一般是清理备份文件失败导致。	您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交工单。	可能导致OBS文件残留。
	绑定EIP失败	GaussDBV5BindEIPFailed	重要	弹性公网IP已被占用或IP资源不足等原因导致。	您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交工单。	导致实例无法使用公网链接或访问。
	解绑EIP失败	GaussDBV5UnbindEIPFailed	重要	网络故障或公网EIP服务故障等原因导致。	重新解绑IP或在管理控制台右上角，选择“**工单 > 新建工单**”，提交工单。	可能导致IP资源残留。
	参数组应用失败	GaussDBV5ApplyParamFailed	重要	一般是由于修改参数组命令超时导致。	重新尝试修改参数组操作。	无
	参数修改失败	GaussDBV5UpdateInstanceParamGroupFailed	重要	一般是由于修改参数组命令超时导致。	重新尝试修改参数组操作。	无
	备份恢复失败	GaussDBV5RestoreFromBcakupFailed	重要	一般是由底层资源不足或备份文件下载失败等原因导致。	您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交工单。	可能导致在恢复失败期间数据库服务不可用。

14.3 创建告警规则

操作场景

通过设置数据库告警规则，用户可自定义监控目标与通知策略，及时了解数据库运行状况，从而起到预警作用。

设置的告警规则包括设置告警规则名称、资源类型、维度、监控对象、监控指标、告警阈值、监控周期和是否发送通知等参数。

操作步骤

步骤1 登录管理控制台。

步骤2 在“服务列表”中选择“管理与监管 > 云监控服务CES”，进入“云服务监控”信息页面。

步骤3 在左侧导航栏选择“云服务监控 > 云数据库 GaussDB”。

步骤4 选择需要添加告警规则的实例，单击操作列的“创建告警规则”。

步骤5 在“创建告警规则”页面，填选相关信息。

- “触发规则”建议选择“导入已有模板”。默认告警模板中已经包含实例数据磁盘使用率告警策略。
- 输入告警“名称”和“描述”。
- 单击开启“发送通知”，生效时间默认为全天，若没有您想要选择的主题，可以单击下一行的“创建主题”进行添加，“触发条件”勾选“出现告警”和“恢复正常”。

  说明

  该告警规则仅在生效时间段内发送通知消息。

步骤6 单击“立即创建”，告警规则创建完成。

关于告警参数的配置，请参见《云监控用户指南》。

----结束

14.4 查看监控指标

操作场景

云服务平台提供的云监控，可以对数据库的运行状态进行日常监控。您可以通过管理控制台，直观地查看数据库的各项监控指标。您可以查看实例监控。

由于监控数据的获取与传输会花费一定时间，因此，云监控显示的是当前时间5～10分钟前的数据库状态。如果您的数据库刚创建完成，请等待5～10分钟后查看监控数据。

前提条件

- 数据库正常运行。

故障、删除状态的数据库，无法在云监控中查看其监控指标。当数据库再次启动或恢复后，即可正常查看。

说明

故障24小时的数据库，云监控将默认该数据库不存在，并在监控列表中删除，不再对其进行监控，但告警规则需要用户手动清理。

- 数据库已正常运行一段时间（约10分钟）。
  对于新创建的数据库，需要等待一段时间，才能查看上报的监控数据和监控视图。

查看实例监控

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在“所有服务”或“服务列表”中选择“管理与监管 > 云监控服务CES”，进入“云监控服务”信息页面。

步骤4 在左侧导航栏选择“云服务监控 > 云数据库 GaussDB”。

步骤5 在云监控页面，单击需要查看监控的实例，可以查看指定实例的监控信息。

云监控支持的性能指标监控时间窗包括：近1小时、近3小时、近12小时、近24小时、7天。

您也可以单击页面左上角的，选择“数据库 > 云数据库 GaussDB，在“实例管理”页面，选择目标实例，单击操作列中的“查看监控指标”，跳转到云监控服务页面查看实例监控信息。您还可以在“实例管理”页面，单击目标实例名称，在页面右上角，单击“查看监控指标”，跳转到云监控服务页面查看实例监控信息。

----结束

14.5 创建事件监控的告警规则

操作场景

本章节指导用户创建事件监控的告警规则。

操作步骤

步骤1 登录管理控制台。

步骤2 在页面左上角单击，在“服务列表”中选择“管理与监管 > 云监控服务CES”，进入“云监控服务”信息页面。

步骤3 在左侧导航栏选择“事件监控”，在“事件监控”页面单击“创建告警规则”。

步骤4 在“创建告警规则”界面，配置参数。

表 14-4 告警内容参数说明

参数	参数说明
名称	系统会随机产生一个名称，用户也可以进行修改。
描述	告警规则描述（此参数非必填项）。
告警类型	用于指定告警规则对应的告警类型。
事件类型	用于指定告警规则对应指标的事件类型。
事件来源	事件来源的云服务名称。 选择“云数据库 GaussDB”。
监控范围	创建事件监控针对的资源范围。
选择类型	选择自定义创建。
事件名称	用户操作系统资源的动作，如用户登录，用户登出，为一个瞬间的操作动作。
触发方式	用户可根据该操作的严重程度选择立即触发或累计触发。
告警策略	触发告警的告警策略。例如：监控周期为5分钟，累计达到3次。 **说明** 当触发方式为累计触发时需配置该参数。
告警级别	根据告警的严重程度不同等级，可选择紧急、重要、次要、提示。
操作	可选择“删除”，删除该条告警策略。

单击开启“发送通知”，生效时间默认为全天，若没有您想要选择的主题，可以单击下一行的“创建主题”进行添加，“触发条件”勾选“出现告警”和“恢复正常”。

说明

该告警规则仅在生效时间段内发送通知消息。

表 14-5 发送通知

参数	参数说明
发送通知	配置是否发送邮件、短信、HTTP和HTTPS通知用户。
通知方式	根据需要可选择通知组或主题订阅两种方式。
通知组	需要发送告警通知的通知组。创建通知组请参见**创建通知对象/通知组**。

参数	参数说明
通知对象	需要发送告警通知的对象，可选择“云账号联系人”或主题。 • 云账号联系人：注册账号时的手机和邮箱。 • 主题：消息发布或客户端订阅通知的特定事件类型，若此处没有需要的主题，需先创建主题并订阅该主题。 详细操作请参见**创建主题**和**添加订阅**。
生效时间	该告警规则仅在生效时间内发送通知消息。 如生效时间为08:00-20:00，则该告警规则仅在08:00-20:00发送通知消息。
触发条件	可以选择“出现告警”、“恢复正常”两种状态，作为触发告警通知的条件。

步骤5 单击“立即创建”，告警规则创建完成。

关于告警参数的配置，请参见《**云监控用户指南**》。

----结束

14.6 事件监控简介

事件监控提供了事件类型数据上报、查询和告警的功能。方便您将业务中的各类重要事件或对云资源的操作事件收集到云监控服务，并在事件发生时进行告警。

事件即云监控服务保存并监控的数据库资源的关键操作，您可以通过“事件”了解到谁在什么时间对系统哪些资源做了什么操作，如规格变更等。

事件监控为您提供上报自定义事件的接口，方便您将业务产生的异常事件或重要变更事件采集上报到云监控服务。

事件监控默认开通，您可以在事件监控中查看系统事件和自定义事件的监控详情，目前支持的系统事件请参见**支持的事件列表**。

15 监控巡检

15.1 监控大盘

操作场景

GaussDB支持同时查看实例多个实时性能指标数据以及对比实例各组件性能指标历史趋势。

说明

该功能仅针对特定用户开放，如需配置白名单权限，您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交开通白名单的申请。

注意事项

- 实例和节点必须选择。
- 每次最多只能选择9个节点 。
- 每次最多只能选择9个组件。

操作步骤

步骤1 **登录管理控制台**。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 单击监控巡检 > 监控大盘，进入监控大盘页面。

步骤5 在监控大盘页面选择指定实例、节点和组件。

图 15-1 选择实例

图 15-2 选择节点

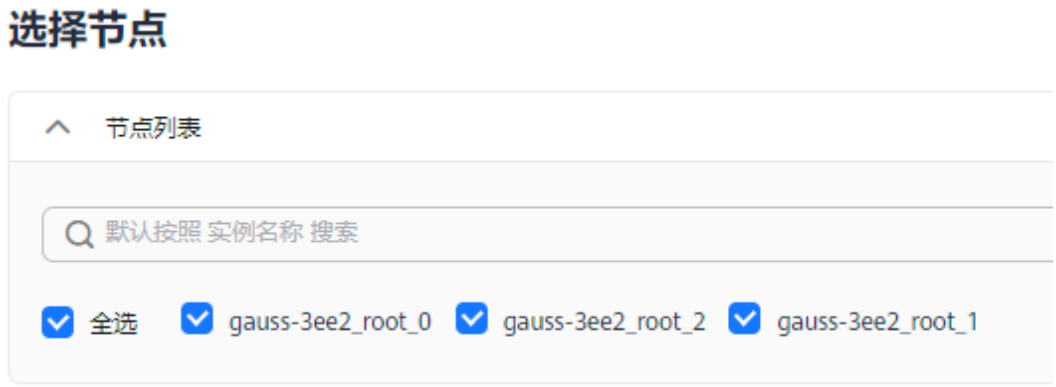
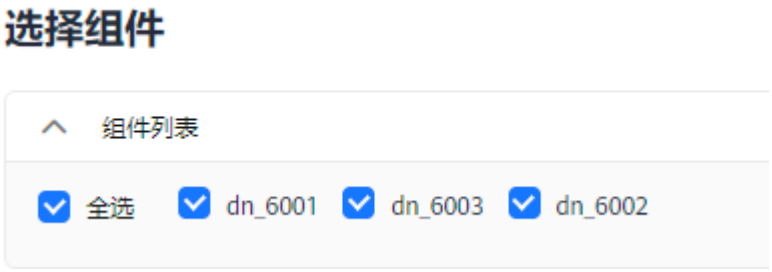
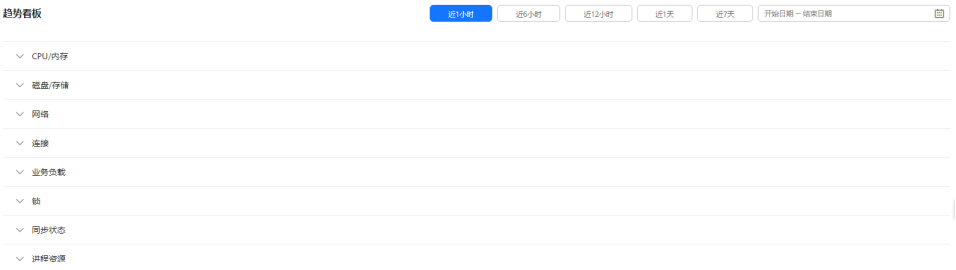


图 15-3 选择组件



步骤6 单击“查看”。

步骤7 选择需要查看的时间段，单击 查看对应指标数据。



----结束

16 CTS 审计

16.1 支持审计的关键操作列表

通过云审计服务，您可以记录与GaussDB实例相关的操作事件，便于日后的查询、审计和回溯。

表 16-1 云审计服务支持的操作列表

操作名称	资源类型	事件名称
创建实例、恢复到新实例	instance	createInstance
删除实例	instance	deleteInstance
数据库实例规格变更	instance	resizeFlavor
实例版本升级	instance	upgradeVersion
密码重置	instance	resetPassword
实例重启	instance	instanceRestart
绑定公共IP	instance	setOrResetPublicIP
解绑公共IP	instance	setOrResetPublicIP
修改资源标签	instance	modifyTag
删除资源标签	instance	deleteTag
添加资源标签	instance	createTag
重命名实例	instance	instanceRename
实例扩容	instance	instanceAction
删除任务记录	instance	deleteTaskRecord4OpenGauss

操作名称	资源类型	事件名称
减少副本	instance	reduceReplica
协调节点缩容	instance	reduceCoordinatorNode
设置回收站策略	backup	setRecyclePolicy
创建手动备份	backup	createManualSnapshot
删除手动备份	backup	deleteManualSnapshot
修改备份策略	backup	setBackupPolicy
实例还原	backup	restoreInstance
备份恢复实例	instance	restoreInstance
修改/更新自动备份保留时长	instance	setBackupPolicy
创建参数组	parameterGroup	createParameterGroup
应用参数组	parameterGroup	applyParameterGroup
复制参数组	parameterGroup	copyParameterGroup
删除参数组	parameterGroup	deleteParameterGroup
重置参数组	parameterGroup	resetParameterGroup
更新参数组	parameterGroup	updateParameterGroup
修改端口号	instance	modifyPort

16.2 查看追踪事件

操作场景

在您开通了云审计服务后，系统开始记录云服务资源的操作。云审计服务管理控制台保存最近7天的操作记录。

本节介绍如何在云审计服务管理控制台查看最近7天的操作记录。

说明

使用云审计服务前需要先开通云审计服务，请参见**开通云审计服务**。

操作步骤

步骤1 登录管理控制台。

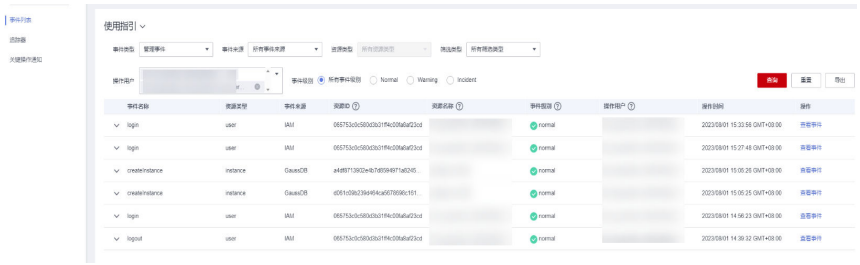
步骤2 在页面左上角单击☰，选择“管理与监管 > 云审计服务 CTS”，进入云审计服务信息页面。

步骤3 单击左侧导航树的“事件列表”，进入事件列表信息页面。

步骤4 事件列表支持通过筛选来查询对应的操作事件。详细信息如下：

- 事件类型、事件来源、资源类型和筛选类型：在下拉框中选择查询条件。
  其中筛选类型选择资源ID时，还需选择或者手动输入某个具体的资源ID。
- 操作用户：在下拉框中选择某一具体的操作用户。
- 事件级别：可选项为“所有事件级别”、“normal”、“warning”、“incident”，只可选择其中一项。
- 时间范围：可通过选择时间段查询操作事件。

步骤5 选择查询条件后，单击“查询”。



步骤6 在需要查看的记录左侧，单击∨展开该记录的详细信息。

步骤7 在需要查看的记录右侧，单击“查看事件”，在弹出框中显示该操作事件结构的详细信息。

步骤8 单击右侧的“导出”，将查询结果以CSV格式的文件导出，该CSV文件包含了云审计服务记录的七天以内的操作事件的所有信息。

关于事件结构的关键字段详解，请参见《云审计服务用户指南》的“事件结构”和“事件样例”章节。

----结束

17 日志管理

17.1 日志分析

GaussDB提供慢日志和错误日志的下载功能，慢日志帮助您定位SQL语句执行慢的问题，错误日志可查看集群的日志信息。

注意事项

- 实例CN与DN节点没有异常。
- 实例所有节点与OBS网络连接正常。
- 底层网络正常。

慢日志

步骤1 登录管理控制台。

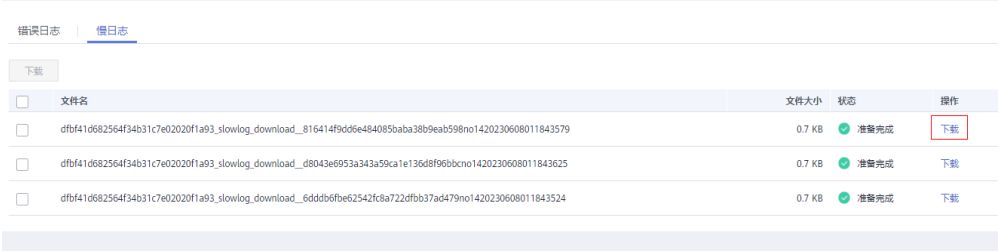
步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击实例的名称，进入“基本信息”页面。

步骤5 单击左侧导航栏中的“日志管理”，进入“日志管理”页面。

步骤6 在日志管理页面，单击“慢日志”，选中状态为“准备完成”的日志，单击操作列的“下载”，即可下载慢日志文件。



下载完成后，可在本地进行分析，调整慢SQL。

----结束

错误日志

步骤1 登录管理控制台。

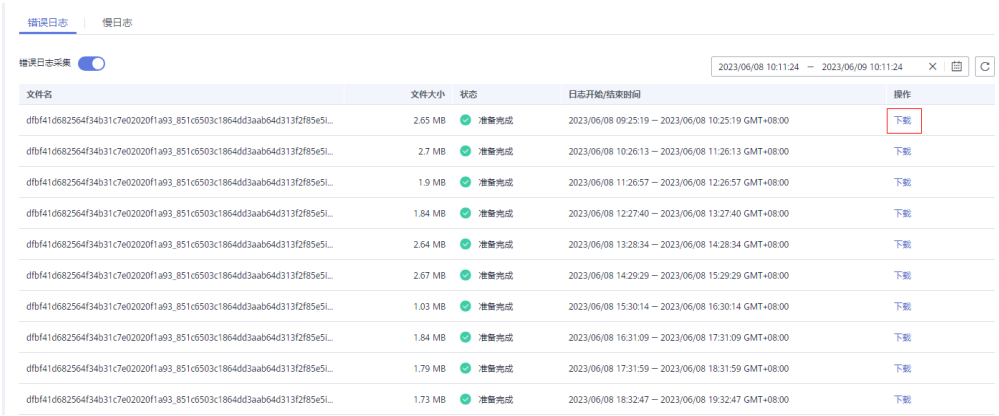
步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击实例的名称，进入“基本信息”页面。

步骤5 单击左侧导航栏中的“日志管理”，进入“日志管理”页面。

步骤6 在日志管理页面，单击“错误日志”，开启“错误日志采集”，单击状态为“准备完成”的操作列的“下载”，即可下载错误日志文件。



下载完成后，可在本地进行分析。

----结束

17.2 LTS 审计日志

操作场景

配置访问日志后，GaussDB实例新生成的审计日志记录会上传到云日志服务（Log Tank Service，简称LTS）进行管理。您可以查看GaussDB实例审计日志的详细信息，包括搜索日志、日志可视化、下载日志和查看实时日志等功能。

- **配置单个实例访问日志**：添加单个实例的LTS配置。
- **解除单个实例访问日志**：解除单个实例的LTS配置。

使用须知

- LTS属于白名单特性，如需开通配置访问日志权限，您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交开通配置访问日志的申请。
- 当前只提供主备版实例， 并且数据库引擎需要为2.1.0及以上版本。
- 访问日志提供了实例所请求的所有详细日志，日志存在LTS云日志服务中。
- 配置完成后，日志不会立即上传，需要等待10分钟左右才可以在LTS服务上查询审计日志。
- 控制审计日志的总开关参考参数**audit_enabled**。
- 控制审计日志的具体参数配置参考**审计项**。
- 配置完成后，会产生一定费用，费用情况请参考LTS的**定价详情**。
- 在您进行LTS审计日志配置后，会默认上传当前实例的所有审计策略到LTS服务。

配置单个实例访问日志

步骤1 **登录管理控制台**。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在左侧导航树，单击“实例管理”。

步骤5 在右边列表单击实例名称，进入实例详情。

步骤6 在实例详情的左侧导航树里，单击“审计日志”。

步骤7 在右边页面中找到“开启上传审计日志到LTS”，单击相邻右侧 。

步骤8 在弹框中，选择“日志组”和“日志流”。

图 17-1 配置 LTS 参数

说明

首次配置LTS时，需要单击“查看日志组”，登录LTS服务配置日志组和日志流，详情请参见日志组和日志流。

步骤9 单击“确定”。

----结束

解除单个实例访问日志

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在左侧导航树，单击“实例管理”。

步骤5 在右边列表单击实例名称，进入实例详情。

步骤6 在实例详情的左侧导航树里，单击“审计日志”。

步骤7 在右边页面中找到“开启上传审计日志到LTS”，单击相邻右侧。

步骤8 在弹框中，确认解除LTS的实例信息。

步骤9 在弹框中，单击“是”。

----结束

18 任务中心

18.1 查看任务

您可以通过“任务中心”查看用户在控制台上提交的任务的执行进度和状态。

说明

GaussDB支持查看和管理以下任务：

- 创建GaussDB实例
- 手动创建备份
- 恢复到新实例
- 磁盘扩容
- 分片扩容
- 协调节点扩容
- 恢复到当前实例
- 规格变更
- 删除GaussDB实例
- 磁盘类型变更
- 停止备份

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“任务中心”页面，选择目标任务，查看任务信息。

- 通过任务名称/任务ID、实例名称/ID确定目标任务，或通过右上角的搜索框选择任务类型来确定目标任务。
- 单击页面右上角的，查看某一时间段内的任务执行进度和状态，默认时长为一周。

  任务保留时长最多为30天。
- 系统支持查看以下状态的任务：
  - 执行中
  - 完成
  - 失败
- 查看任务创建时间和结束时间。

----结束

18.2 删除任务

对于不再需要展示的任务，您可以通过“任务中心”进行任务记录的删除。删除任务仅删除记录，不会删除数据库实例或者停止正在执行中的任务。

须知

删除任务将无法恢复，请谨慎操作。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“任务中心”页面，选择目标任务，单击操作列的“删除”，在弹出框中单击“是”，删除任务。

GaussDB服务支持删除以下状态的任务：

- 完成
- 失败

----结束

19 标签

操作场景

标签管理服务（Tag Management Service，TMS）用于用户在云平台，通过统一的标签管理各种资源。TMS服务与各服务共同实现标签管理能力，TMS提供全局标签管理能力，各服务维护自身标签管理 。

- 建议您先在TMS系统中设置预定义标签。
- 标签由“键”和“值”组成，每个标签中的一个“键”只能对应一个“值”。
- 每个实例最多支持20个标签配额。

添加/编辑标签

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击实例名称，进入实例的“基本信息”页签。

步骤5 在左侧导航栏，单击“标签”，单击“添加/编辑标签”，在弹出框中，输入标签的键和值，单击“添加”，将标签加入上方输入框，单击“确定”。

图 19-1 添加/编辑标签

- 输入标签的键和值时，系统会自动联想当前用户的所有实例（除当前实例外）的所有关联的预定义标签。
- 标签的键不能为空且必须唯一，长度为1~36个字符，只能包含中文、字母、数字、中划线、下划线和@符号。
- 标签的值可以为空字符串，长度为0~43个字符，只能包含中文、字母、数字、中划线、下划线、 英文句点和@符号。

步骤6 添加成功后，您可在当前实例的所有关联的标签集合中，查询并管理自己的标签。

----结束

删除标签

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的 ，选择区域和项目。

步骤3 在页面左上角单击 ，选择“数据库 > 云数据库 GaussDB”。进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击实例名称。

步骤5 在左侧导航栏，单击“标签”，选择需要删除的标签，单击“删除”。

图 19-2 删除标签

步骤6 在“删除标签”弹出框中单击“是”。

步骤7 删除成功后，该标签将不再显示在实例的所有关联的标签集合中。

----结束

20 配额

什么是配额?

为防止资源滥用，平台限定了各服务资源的配额，对用户的资源数量和容量做了限制。如您最多可以创建多少个云数据库GaussDB实例。

如果当前资源配额限制无法满足使用需要，您可以申请扩大配额。

怎样查看我的配额?

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的 ，选择区域和项目。

步骤3 在页面右上角，选择“资源 > 我的配额”，进入“服务配额”页面。

图 20-1 我的配额

步骤4 您可以在“服务配额”页面，查看各项资源的总配额及使用情况。

----结束

如何申请扩大配额?

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面右上角，选择“资源 > 我的配额”，进入“服务配额”页面。

步骤4 在页面右上角，单击“申请扩大配额”。

图 20-2 申请扩大配额

步骤5 在“新建工单”页面，根据您的需求，填写相关参数。

其中，“问题描述”项请填写需要调整的内容和申请原因。

步骤6 填写完毕后，勾选协议并单击“提交”。

----结束