译者序

背景简介

近几年，随着 Greenplum(GP)作为一种数据仓库产品在中国的大范围推广，越来越多的技术人员开始接触这个陌生的数据仓库产品。其基于开源的数据库 PostgreSQL(PG)，虽然 PG 是一个被全世界广泛使用的开源数据库，但在中国，使用者相对较少。再加上 Greenplum 具备独特的分布式 Share-Nothing 特征，在使用初期很难准确深刻的理解很多细节。在调优方面和故障处理方面更是无从入手。译者在接触了 Greenplum 两年之后决定将主要内容翻译为中文，并将译者使用中遇到的问题解决方案适当的添加进来。该书与官方文档不等价。本书基于 Greenplum4.2.2.0 版本，由于 GP 的多个版本之间存在较大的功能差异，本书中涉及的内容未必适用其他版本，尤其是早期的版本(4.0 之前)。本书中译者经验的叙述不代表官方观点，对于一些风险较高的操作，最好获得 EMC 官方的支持。建议慎重对待任何建议，从自己对 GP 深入理解的角度来看待问题才是最合理的。

致读者

本书虽基于官方文档翻译，但灌注了译者诸多的心血，请尊重译者的辛勤工作，如其中内容不符合你的期望，请不要攻击诽谤译者，未经译者授权不得用于任何商业用途，译者保留追究的权利。如有任何的建议，可邮件联系译者，谢绝恶意攻击，你的建议译者会尽力考虑，但未必如你所愿。

译者不承诺增加调优排错等超出官方文档内容的知识，比如为不同性能的机器配置非平衡集群，非对称镜像，无法启动事故处理，日常的管理维护，查询语句调优，系统最优化安装方案等等诸多方面，这些内容可能会在书中提及，但译者不承诺做深入讲解。如需更多支持可联系 EMC 技术支持或者与译者联系(建议在相信译者具备这个能力且对其他支持不是特别满意的情况下考虑)。望读者理解。

译者：陈淼
电邮：miaochen@mail.ustc.edu.cn
手机：+86 18616691889(021)
：18616691889

版本说明

译者今后将通过微信订阅号 greenplum 发布技术分享信息。
关注微信订阅号可搜索 greenplum 公众号或者直接扫描右上方的二维码。
对于该文档的更新将不再有任何承诺。

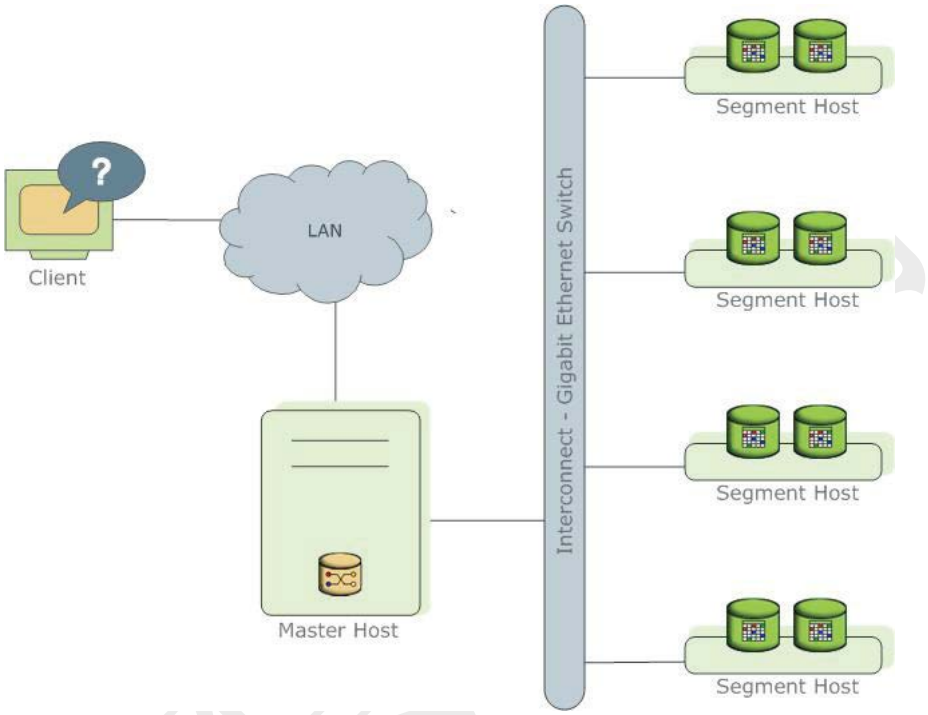
致谢

感谢我的家人对我工作的理解与支持！
感谢我的老婆长期以来对我工作的理解和支持！

目录

第一章：GPDB 架构简介

GPDB 是一个分布式数据库软件，其可以管理和处理分布在多个不同主机上的海量数据。对于 GPDB 来说，一个 DB 实例实际上是由多个独立的 PostgreSQL 实例组成的，它们分布在不同的物理主机上，协同工作，呈现给用户的是一个 DB 的效果。Master 是 GPDB 系统的访问入口，其负责处理客户端的连接及 SQL 命令、协调系统中的其他 Instance(Segment)工作，Segment 负责管理和处理用户数据。



这一章节阐述组成 GPDB 系统的组件及如何协同工作：

- 管理节点 Master
- 计算节点 Segment
- 网络
- 冗余与故障切换
- 并行数据装载
- 管理与监控

管理节点 Master

Master 作为 GPDB 系统的访问入口，其处理客户端连接的访问以及用户提交的 SQL 语句。

GPDB 是基于 PostgreSQL 发展而来，终端用户可以像 PostgreSQL 那样与 GPDB 进行交互。并同样可以通过客户端程序(如 psql)和应用程序接口(APIs(如 JDBC、ODBC))连接 GPDB。

Master 上存储着全局系统表(Global System Catalog)(包含 GPDB 系统自身元数据的系统表)，但不存储任何用户数据，用户数据只存储在 Segment 上。Master 负责客户端认证、处理 SQL 命令入口、在 Segment 之间分配工作负载、整合 Segment 处理结果、将最终

结果呈现给客户端程序。

计算节点 Segment

在 GPDB 系统中，Segment 才真正是数据存储和查询处理的地方。用户 Table 和相应的 Index 都分布在 GPDB 系统中各 Segment 上，每个 Segment 存储着一部分不同的数据。Segment 实例(Instance)才是真正的数据处理进程。用户不能够直接跳过 Master 访问 Segment，而只能通过 Master 来访问整个系统。

在 GPDB 推荐的硬件配置环境下，每个有效的 CPU 核对应一个 Segment Instance，比如一个 Segment 主机配备了 2 个双核的 CPU，那么可以选择每个 Segment 主机 4 配置个主实例(Primary Instance)。

网络

网络层是 GPDB 系统的重要组件，在用户执行查询时，每个 Instance 都需要执行相应的处理，网络层涉及到 Instance 之间的通信，网络层可以使用标准的以太网络协议。

在默认情况下，网络层使用 UDP 协议。GPDB 自己会为 UDP 协议做数据包校验，其可靠性与 TCP 协议相同，但其性能和扩展性远好于 TCP 协议。在使用 TCP 协议的情况下，GPDB 的 Instance 数量被限制在 1000 个。为了去除这个限制，UDP 协议被作为默认的网络层协议。

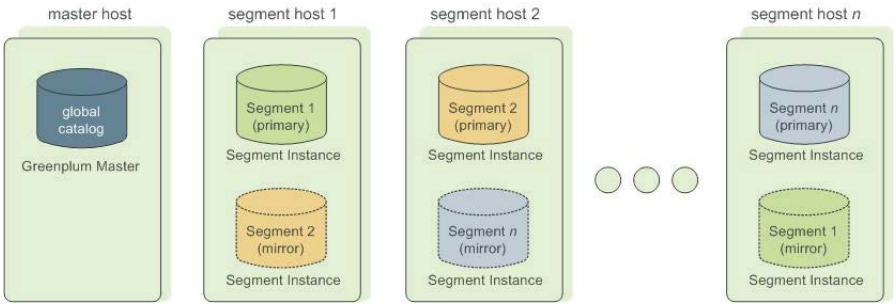
冗余与故障切换

GPDB 提供了避免单点故障的部署选项。本节阐述 GPDB 的冗余组件。

- Segment 镜像
- Master 镜像
- 网络层冗余

Segment 镜像

在部署 GPDB 系统时，可以选择配置 Mirror Instance。Mirror Instance 使得数据库查询在 Primary Instance 不可用时切换到备份的 Instance 上。为了配置 Mirror，GPDB 系统需要有足够多的 Host，从而可以保证冗余的 Segment Instance 总是在与 Primary Instance 不同的 Host 主机上。下图展示了在配置了 Mirror 的情况下数据如何分布在不同的 Host 节点上。Mirror Instance 总是位于不同于 Primary Instance 的 Host 主机上。



Segment 故障切换与恢复

在 GPDB 系统 Mirror 启用的置情况下，当 Primary Instance 不可访问时，系统会自动切换到其对应的 Mirror Instance 上，此时，Mirror Instance 取代 Primary Instance 的作用。只要剩余的可用 Segment Instance 能够保证数据完整性，在 Segment Instance 或者 Host 主机宕机时，GPDB 系统仍可保持可用状态。

每当 Master 无法连接到 Primary Instance 时，其都会在 GPDB 的系统日志表中被标记为失败状态，并激活/唤醒对应的 Mirror Instance 取代原有的 Primary Instance。在采取相应的措施将其恢复到联机(Online)状态之前，失败的 Primary Instance 一直保持未运行状态。失败的 Primary Instance 可以在系统处于运行状态下被恢复回来。恢复进程仅仅复制失败期间发生变化的那部分数据。

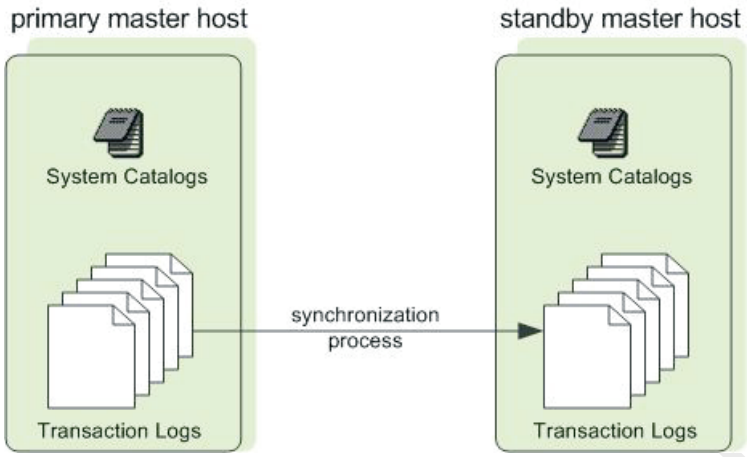
在未启用 Mirror 的情况下，任何的 Segment Instance 失败都会导致系统自动停止服务。在继续使用系统之前，必须恢复所有失败的 Segment Instance。

Master 镜像

同样，可以为 Master 部署一个备份/镜像到一个不同于 Master 节点的主机上。在 Master 不可用时，Standby 就成为了热备 Master。Standby 与 Master 之间保持事务日志的同步，其保证 Standby 与 Master 之间的一致性。

在 Master 失效时，复制进程会自动停止，同时，Standby 可以被激活。在 Standby 上，冗余的日志被用来将状态恢复到最后成功提交(commit)的状态。激活的 Standby 实际上会成为 GPDB 的 Master，通过 Master Port(该端口需要设置和 Master 的相同)接受客户端的链接访问。

由于 Master 不存储用户数据，在 Master 和 Standby 之间仅仅是系统信息表需要被同步。这些表很少发生变化，一旦发生变化，就会自动同步到 Standby 从而保证与 Master 的一致性。



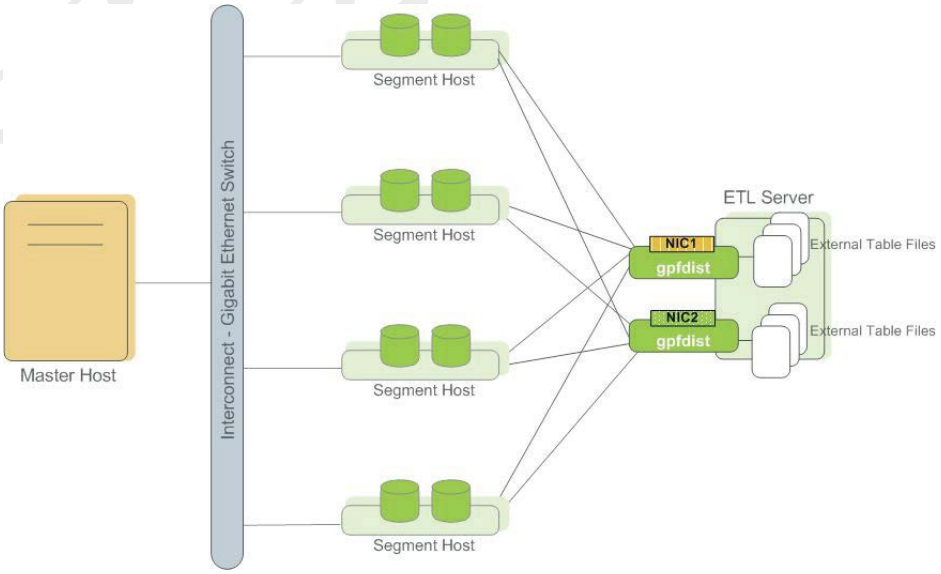
网络层冗余

网络层关系到 Segment Instance 之间的通信，其依靠基础网络设施，高可用网络层可以通过部署双重以太网络实现。不过如果在配置 Mirror 的情况下，通过不同网段间的 Primary 与 Mirror 之间的对应关系也可以达到网络保障的效果。

并行数据装载

海量数据仓库的一个重大挑战就是在一个给定的时间窗口内完成大量数据的装载。GP 通过外部表(External Table)支持高速并行数据装载。外部表可以使用‘单条记录出错隔离’模式，从而允许管理员在装载数据时将出错的数据记录剥离到一个单独的错误记录表中。管理员还可以控制错误容忍阀值，以实现对数据装载质量的控制。

结合使用外部表和 GPDB 的并行文件服务(gpfdist)，管理员可以实现最大化利用网络带宽资源以实现高速并行装载。

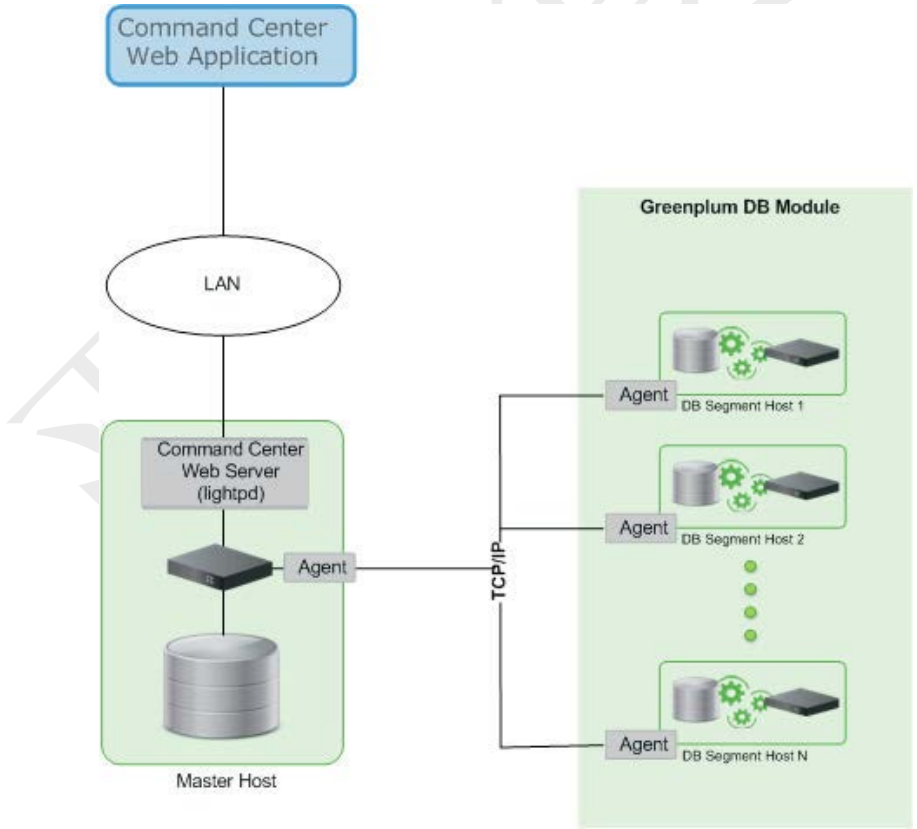


管理与监控

对 GPDB 系统的管理通过一系列的命令行来实现，它们都放置在$GPHOME/bin 目录下。GPDB 提供的命令可以实现如下的管理任务：

- 批量安装 GPDB 软件
- 初始化 GPDB 系统
- 启动关闭 GPDB 系统
- 添加或移除 Host 主机
- 扩展 Segment Instance 以及在新节点间重新分布 Table
- 监控和恢复失败的 Segment Instance
- 监控和恢复失败的 Master Instance
- 备份和恢复数据库(并行)
- 并行装载数据
- 系统状态报告

GP 还提供了一个可选的监控管理工具，管理员可以选择与 GPDB 一起安装和启用。GPCC(Greenplum Command Center)使用数据收集代理程序(Agent)在各个 Segment Host 收集数据库的指标。Agent 会定期(比如 15 秒)主动将 Segment Instance 上收集的数据发送给 Master。用户可以直接查询 CC 数据库查看系统指标。GPCC 还有一个基于 WEB 的图形化用户交互界面服务，其可以独立于 GPDB 安装。

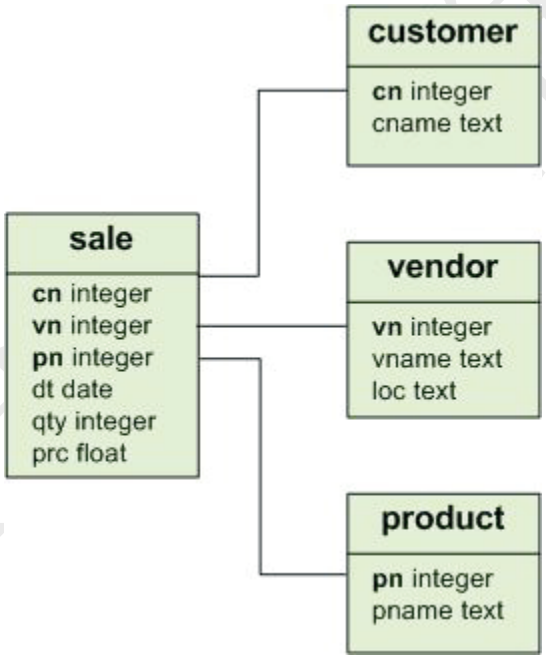


第二章：分布式数据库概念

GP 是一个分布式数据库系统。这就意味着在物理上，数据是存储在多个数据库服务上的(在 GP 中称为 Segment Instance)。这些独立的数据库服务通过网络进行通信(在 GP 中称为网络层)。分布式数据库的一个基本特征是，用户和客户端程序如同访问一个单机数据库(在 GP，这个入口数据库称为 Master)。数据库分布在不同的机器上，但对于用户来说，如同使用一个单机数据库一样。

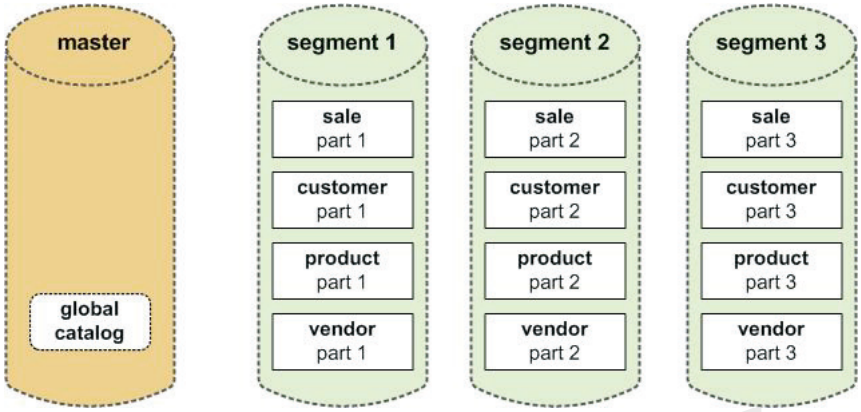
数据是如何存储的

要理解 GPDB 是如何在不同的 Segment Instance 之间存储数据，如下图所示的简单逻辑数据库，主键(Primary Key)被使用黑体标记，外键(Foreign Key)关系通过连线标明。
用数据仓库的术语来说，这种数据模型称为星型模型。在这种数据库模型下，sale 表通常被称为事实表(Fact Table)，其他表(Customer、Vendor、Product)被成为维表(Dimension Table)。



GPDB 系统中所有的用户表都是分布的，这意味着数据被拆分成无重叠的记录集合。每部分存储在一个 Segment Instance 中。数据通过复杂的 HASH 算法分布到所有 Segment Instance。HASH KEY(一个或者多个)由管理员在定义 Table 时指定。

GPDB 从底层上来说，通过一系列相关的独立 Database Instance 实现——1 个 Master Instance 和数个 Segment Instance。Master Instance 不存储用户数据。Segment Instance 存储每张表无重叠的部分数据(记录集合/Collection Of Rows)。



解读 GP 分布策略

在 GPDB 中创建(Create)或者修改(Alter)表时，有一个额外的 DISTRIBUTED 子句用以定义表的分布策略(Distribution Policy)。分布策略决定了表中的数据记录如何被打散到 GP 的 Segment Instance。GPDB 提供了 2 种分布策略：

- **HASH 分布**

  使用 HASH 分布时，1 个或数个 Table Column 被用作 Distribution Key(简称 DK)。DK 被 HASH 算法用来决定每行记录对应特定的 Segment Instance。相同 Key 值的记录会 HASH 到相同的 Segment Instance。选择一个唯一键(unique key)作为 DK，比如主键(Primary Key)，可以确保尽可能的平坦分布数据。

- **随机(Random)分布**

  使用随机分布，数据记录被循环的分布到 Segment Instance。相同值的记录可能会落在不同的 Segment Instance。随机分布可以确保数据分布的平坦性，但为了确保性能优势应该尽可能的使用 HASH 分布。

第三章：GPDB 特性摘要

本章概述 GPDB 的系统要求及相关特征。包含以下论题：

- **GP SQL 标准一致性**
- **GP 与 PostgreSQL 兼容性**

GP SQL 标准一致性

SQL 语言在 1986 年首次由美国国家标准协会(ANSI)标准化。随后的版本由 ANSI 发布并由国际标准化机构(ISO)标准化：SQL1989，SQL1992，SQL2003，SQL2006 及目前的标准 SQL2008。官方标准化名称为 ISO/IEC 9075-14:2008。通常新版的标准会增加一些特性，偶尔也会有特性被建议废弃或者剔除。

值得一提的是，目前还没有一款商业数据库系统完全符合 SQL 标准。GPDB 是几乎完全符合 SQL1992 标准，符合大部分的 SQL1999 标准。数个来自 SQL2003 标准的特性被实现(尤其是大部分的 SQL OLAP 特性)。

本章重点阐述 GPDB 与 SQL 标准间的关系。GP 对 SQL 标准一对一的特性支持列表可参见相关附录，”SQL2008 标准可选特性支持”。

核心 SQL 一致性

由于 GPDB 采用的是无共享(Shared-Nothing)架构，GPDB 的查询优化器对于一些 SQL 结构目前还没有实现。下面这些 SQL 结构是不被支持的：

1. 一些在 EXISTS 或 NOT EXISTS 子句中有返回值的子查询，其无法被 GP 的查询优化器重写为 JOIN。
2. 表 JOIN 且有子查询时的 UNION ALL 语法。
3. 在 FROM 作用的子查询中有记录集返回的 FUNCTION。
4. 反向滚动游标(CURSOR)，包括 FETCH PRIOR、FETCH FIRST、FETCH ABOLUTE、和 FETCH RELATIVE。
5. 在使用 CREATE TABLE 语句时，UNION 或者 PRIMARY KEY 子句必须包含所有 DK 列(若有 DK)。出于这个限制，仅有一个 UNION 或者 PRIMARY KEY 子句可被使用在 CREATE TABLE 语句中。而对于 DISTRIBUTED RANDOMLY 的表来说，不允许使用 UNION 和 PRIMARY KEY 子句。
6. CREATE UNIQUE INDEX 子句未包含全部 DK 列。CREATE UNIQUE INDEX 不可用在 DISTRIBUTED RANDOMLY 表。
7. VOLATILE 或者 STABLE 的 FUNCTION 不能在 Segment Instance 得到执行，因此只能将字面值作为参数传递给这些函数。
8. 触发器(Trigger)不被支持，虽然其依赖于 VOLATILE FUNCTION。
9. 外键(Foreign Key)的参考约束在 GPDB 中不生效。用户仍可以定义外键，并且这些信息也会保存在系统信息表中。
10. 序列(Sequence)操作函数 CURRVAL 和 LASTVAL。
11. DELETE WHERE CURRENT OF 和 UPDATE WHERE CURRENT OF(指定游标删除和更新操作)。

SQL1992 一致性

以下的 SQL1992 标准特性在 GPDB 中不被支持：

1. NATIONAL CHARACTER(NCHAR)和 NATIONAL CHARACTER VARYING (NVARCHAR)。用户可以使用 NCHAR 和 NVARCHAR 类型，不过在 GPDB 中他们只是作为 CHAR 和 VARCHAR 的同义词。
2. CREATE ASSERTION 子句。
3. INTERVAL 在 GPDB 中的支持与 SQL 标准中不一致。
4. GET DIAGNOSTICS 子句。
5. GRANT INSERT 或 UPDATE 对列进行授权。在 GPDB 中只能对 TABLE 对象进行权限授予，而不能对其 COLUMN 进行权限操作。
6. GLOBAL TEMPORARY TABLE 和 LOCAL TEMPORARY TABLE。GP 的 TEMPORARY TABLE 与 SQL 标准不一致，但与多数的商业数据库相同。GP 的 TEMPORARY TABLE 与 Teradata 的 VOLATILE TABLE 是不同的。
7. UNIQUE 谓语(集合操作)。
8. 针对参考约束校验的 MATCH PARTIAL(可能以后也不会被实现)。

SQL1999 一致性

以下的 SQL1999 标准特性在 GPDB 中不被支持：

1. 大数据对象:BLOB，CLOB，NCLOB。不过在 GP 中可以使用 BYTEA 和 TEXT 列来存储特别大的对象。
2. 模块化(SQL 端的模块化，类似 Oracle 的 PLSQL)。
3. 创建存储过程(PROCEDURE)。不过在 GPDB 中可以通过创建返回值为 void 的函数(FUNCTION)来实现同样的效果，调用的时候通过这样的方式即可：
   SELECT myfunc(args);
4. PostgreSQL/GP 的函数(FUNCTION)定义语言(PL/PGSQL)是 Oracle PL/SQL 的子集。更像是 SQL/PSM 定义语言。GPDB 仍然支持通过 Python，Perl，Java 和 R 语言来定义函数。
5. BIT 和 BIT VARYING 数据类型(故意遗漏)。其在 SQL2003 中被弃用，且在 SQL2008 中被取代。
6. GP 支持 63 个字符长度的对象标识符。SQL 标准要求支持 128 个字符长度的对象标识符。
7. Prepared Transaction(PREPARE TRANSACTION, COMMIT PREPARED, ROLLBACK PREPARED)。这意味着 GP 不支持分布式事务(XA Transaction)(数据库事务与外部事务分两阶段提交)。
8. CHAR 和 VARCHAR 数据列上的字符集选项(CHARACTER SET)。
9. 在 CHAR 或者 VARCHAR 数据列上的 CHARACTERS 或 OCTETS (BYTES)方式的长度说明。比如：VARCHAR(15 CHARACTERS)或者 VARCHAR(15 OCTETS)或者 VARCHAR(15 BYTES)。
10. CURRENT_SCHEMA 函数。
11. CREATE DISTINCT TYPE 语句。在 GP 中可以使用 CREATE DOMAIN 达到相同的效果。

12. 显式表(没明白原文为 explicit table)。

SQL2003 一致性

以下的 SQL2003 标准特性在 GPDB 中不被支持：

1. MERGE 子句(Oracle 的典型 UPDATE INSERT 操作)。
2. 身份列(IDENTITY columns)和默认生成子句 GENERATED ALWAYS/GENERATED BY DEFAULT。序列(SERIAL)和大序列(BIGSERIAL)类似通过默认身份列(DEFAULT AS IDENTITY)生成的 INT 和 BIGINT。
3. 多重结果上的数据类型修改手段(MULTISET modifiers on data types)。
4. ROW 数据类型。
5. GPDB 使用非标准语法操作序列(SEQUENCE)。例如：在 SQL 标准中使用 NEXT VALUE FOR seq，而在 GPDB 中使用 nextval('seq')。
6. GENERATED ALWAYS AS columns。VIEW 可以作为变通方案。
7. 在 SELECT 语句中的样本子句(TABLESAMPLE)。Random()函数可以作为从 table 获取随机样本函数的变通方案。比如：select * from pg_class order by random() limit 10。
8. 在 SELECT 语句中的 NULLS FIRST 或者 NULLS LAST 子句(确保空值前置或后置)。
9. 分区 JOIN(在 JOIN 时使用 PARTITION BY)。
10. 对列进行 SELECT 授权。GPDB 中仅仅可以精确到对 Table 进行授权(GRANT)。可以考虑使用视图(VIEW)作为变通方案。
11. 对于 CREATE TABLE x (LIKE(y))语句，GPDB 不支持以下子句：[INCLUDING|EXCLUDING] [DEFAULTS|CONSTRAINTS|INDEXES]。
12. GP 的数据类型近乎 SQL 标准相一致。通常不会存在任何的使用问题。

SQL2008 一致性

以下的 SQL2008 标准特性在 GPDB 中不被支持：

1. BINARY 和 VARBINARY 数据类型。
2. 在使用 SELECT 语句时 FETCH FIRST 或 FETCH NEXT 子句，例如：
   SELECT id, name FROM tab1 ORDER BY id OFFSET 20 ROWS FETCH NEXT 10 ROWS ONLY;
   GP 使用 LIMIT 和 LIMIT OFFSET 子句代替这种分页功能。
3. 在视图(VIEW)或者子查询(SUBQUERY)中 ORDER BY 子句是被忽略的，除非有 LIMIT 子句存在。这在 GPDB 中是被故意忽略的，理由是：GP 的优化器不知道何时避免排序是安全的，因为排序可能会导致无法预期的资源开销。作为一种变通方案，可以使用很大的 LIMIT 来强制 ORDER BY。比如：SELECT * FROM mytable ORDER BY 1 LIMIT 9999999999。
4. 不支持行子查询(ROW SUBQUERY)。
5. TRUNCATE TABLE 操作不支持 CONTINUE IDENTITY 和 RESTART IDENTITY 子句。

GP 与 PostgreSQL 兼容性

GPDB 是基于 PostgreSQL8.2 开发的，增加了一些 8.3 版本的新特性。为了支撑 GPDB 系

统的分布式特征和典型的工作负载特征，一些 SQL 命令会被增加或者修改，且少量 PostgreSQL 特性会不被支持。同时 GP 也增加了一些 PostgreSQL 中没有的特性，比如物理数据分布，并行查询优化，外部表，资源队列，表分区等。完全的 SQL 语法参考可查阅”SQL 命令参考”相关章节。

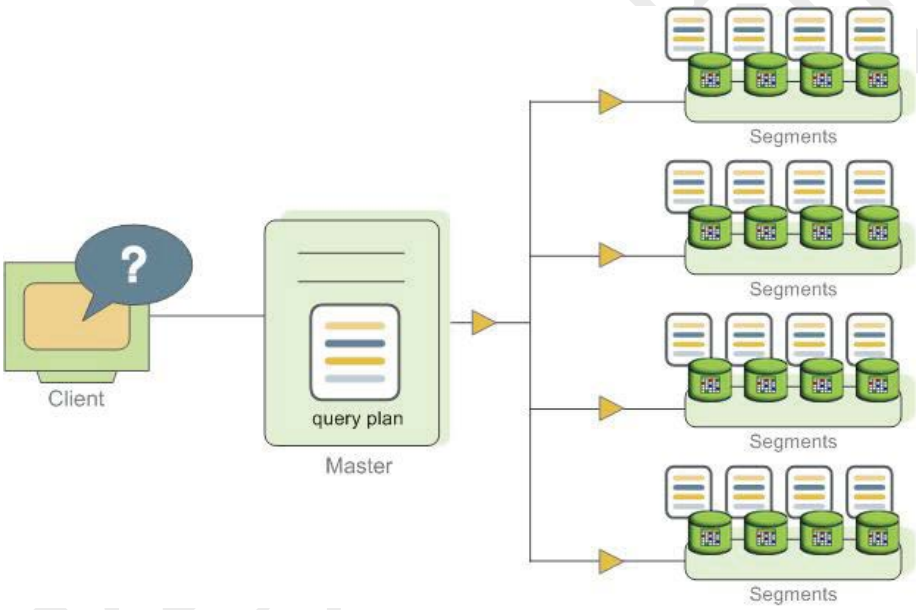
第四章：GPDB 查询处理

用户可以像使用其他的 DBMS 一样向 GPDB 提交查询。直接通过客户端程序(例如 psql)连接到 GP 的 Master 主机并提交查询语句。

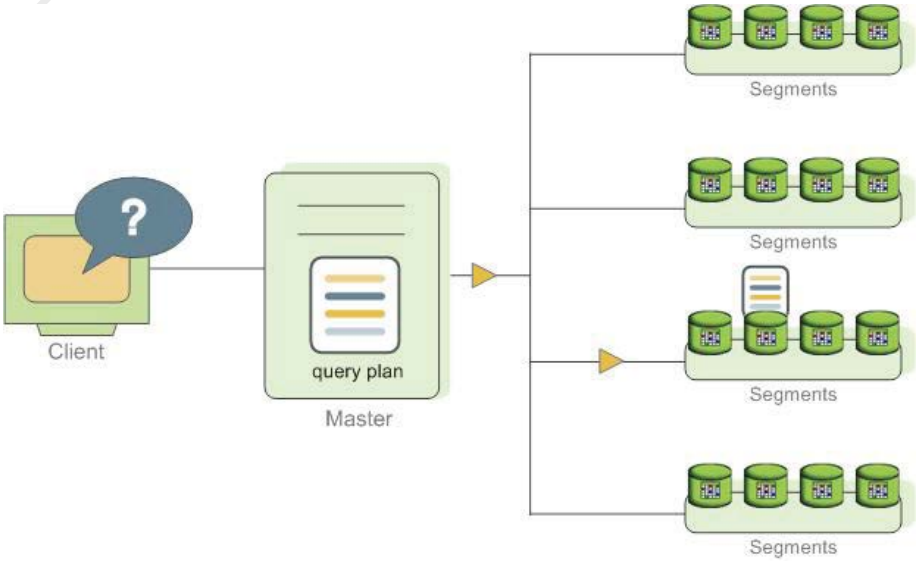
理解查询规划与分发

查询被 Master 接收、处理、优化、创建一个并行的或者定向的查询计划(根据查询语句决定)。之后 Master 将查询计划分发到相关的 Instance 去执行，每个 Instance 只负责处理自己本地的那部分数据。

大部分的操作—比如扫表、关联、聚合、排序都是同时在 Instance 上并行被执行。每个操作都独立于其他 Instance 执行。



一些特定的语句可能只使用一个 Instance，比如单行的 INSERT、UPDATE、DELETE 或者 SELECT 操作，还有就是一些直接过滤 DK 列的查询。这些语句不会被分发到全部 Instance，而是定向到包含该 DK 列的 Instance。



理解查询计划

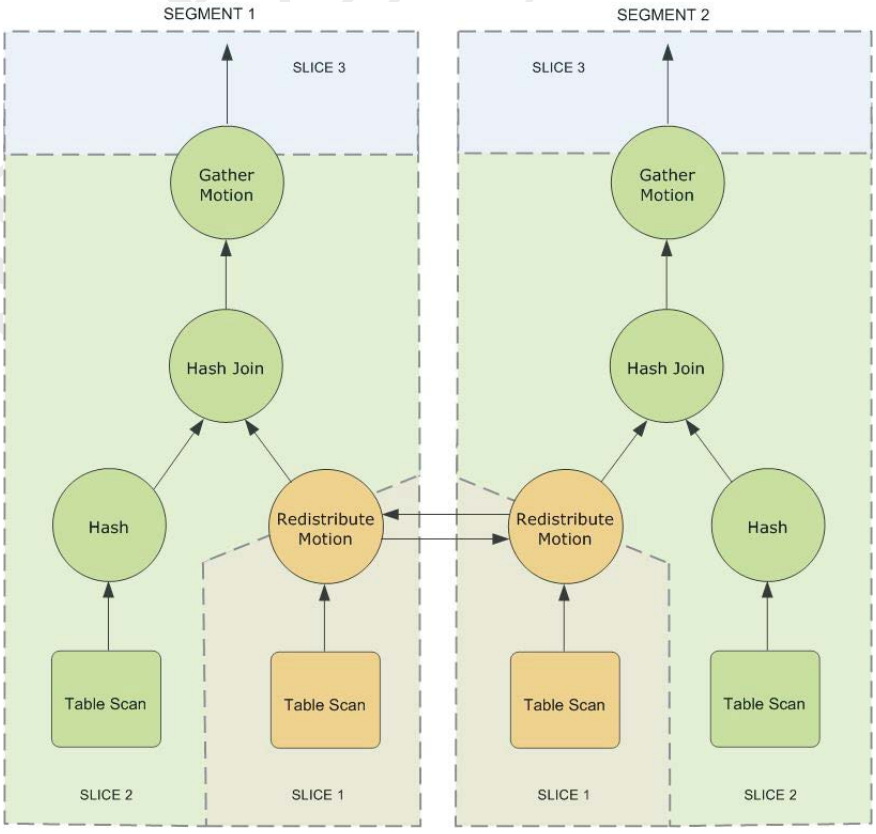
查询计划是 GPDB 为了特定查询语句生成相应结果的一系列操作。查询计划的每一步代表着特定的 DB 操作，比如：扫表、关联、聚合、排序等。查询计划被从下向上执行。与典型的 DB 操作不同的是，GPDB 有一个特有的操作：移动(motion)。移动操作(motion)涉及到查询处理期间在 Instance 之间移动数据。不过并非所有的查询都需要移动数据。比如针对系统日志表(在 Master 上)的查询不会涉及通过网络层移动数据。

为了最大限度的实现并行化处理，GP 会将查询计划分割为多个步骤。步骤是查询计划的一部分，其可以在独立的 Instance 上执行。当查询计划需要移动数据时，查询计划会被分割开，两个操作分处在数据移动步骤两侧，即：先执行一步操作，然后执行数据移动，再执行下一步操作。

例如，下面两个 table 的关联查询：

```
SELECT customer, amount FROM sales JOIN customer USING (cust_id)
WHERE dateCol = '04-30-2008';
```

下图解释了查询计划。每个 Instance 获取到查询计划的副本且并行执行。对于该查询计划来说，有一个重分布操作(Redistribute motion)，这是为了完成连接(join)而执行的数据移动操作。查询计划被重分布操作分割为两步(slice 1 与 slice 2)。该查询计划还有另外一种数据移动称为汇总移动(Gather motion)。汇总移动是 Instance 将计算结果反馈到 Master 从而可以反馈给客户端的一种操作。由于在移动动作发生时查询计划总是会被分割，该查询计划还存在一个隐含的操作(slice 3)。不是所有的查询都有汇总操作，比如：CREATE TABLE…AS SELECT…语句就不需要汇总操作。



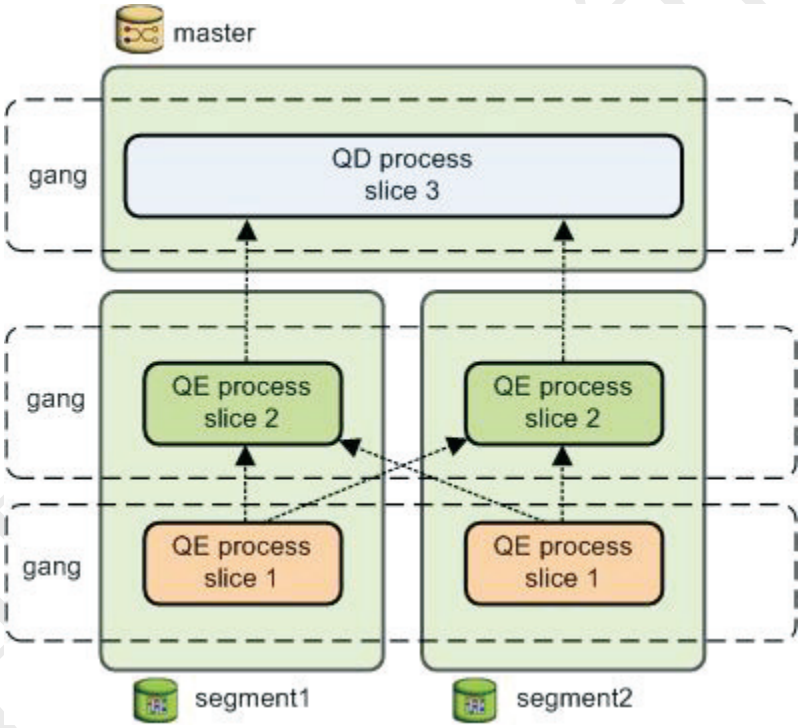
理解并行执行

GP 会创建多个 DB 进程来处理查询。在 Master 上的称为执行分发器(Query Dispatcher/QD)。QD 负责创建、分发查询计划，汇总呈现最终结果。在 Instance 上，处理进程被称为查询执行器(Query executor/QE)。QE 负责完成自身部分的处理工作以及与其他处理进程之间交换中间结果。

查询计划的每个处理部分都至少涉及一个处理工作。执行进程只处理属于自己部分的工作。在查询执行期间，每个 Instance 会并行的执行一系列的处理工作。

同一部分相关的处理工作称为簇(原文为 gangs)。在一部分处理完成后，数据将从当前处理向上传递，直到查询计划完成。Instance 之间的通信涉及到 GPDB 的网络层组件。

下图显示查询处理如何在 Master 和 2 个 Instance 之间被逐步执行的。



第五章：角色权限管理

GPDB 通过角色(Role)的概念来管理数据库的访问权限。Role 的概念包含两个子概念用户(User)和组(Group)。一个 Role 可以是一个 DB User 或者一个 Group 或者两者兼备。Role 可以拥有 DB 对象(比如 Table)并可以分配该对象的权限给其他 Role 从而实现对该对象的控制。Role 还可以成为其他 Role 的成员，因此 Role 可以继承其父级 Role 的对象权限。

每个 GPDB 系统都包含一系列的 Role(User 或 Group)。这些 Role 与运行在 OS 上的 Role 是没有关系的。但出于便利考虑，可以选择使用与 OS Role 相关联的 GPDB Role，这样对于一些默认使用 OS User 名称作为 DB User 的应用来说会比较方便。

在 GPDB 中 User 通过 Master 登录和认证。而对于 Instance 的访问是 Master 通过后台操作实现，与当前登录的 User 信息无关。

Role 是定义在 GPDB 系统级别的，这就意味着，在其所在系统的全部 DB 实例中都是有效的。

为了能够启动 GPDB 系统，在初始化系统时会自动包含一个 SUPERUSER，该 Role 与执行该初始化操作的 OS User 相关。该 Role 与该 OS User 具有相同的 Name。习惯上这个 Role 使用 gpadmin。为了能够创建其他 Role，一开始需要使用该初始化 Role。

角色与权限安全的最佳实现

- **保护系统 User gpadmin**。GP 需要使用一个 UNIX 的 User ID 来安装和初始化 GPDB 系统。在 GP 文档中该系统 User ID 被称为 gpadmin。gpadmin 用户作为 GPDB 系统的默认 SUPERUSER，同时是 GP 安装目录及相关数据文件的 Owner。默认管理员账户是 GPDB 的基础，如果没有该账户将无法运行，且没有办法限制 gpadmin 用户的访问。gpadmin 用户可以绕过 GPDB 的所有权限限制。任何人通过 gpadmin 登录到 GP 主机后，都可以 Read、Alter、Delete 任何数据，包括系统日志表的访问和 DB 操作。因此，保护好 gpadmin 用户账号是很重要的。超级用户(gpadmin)只应该用以执行特定的系统管理任务(比如升级、扩展等)。DB User 不应该使用 gpadmin 账号，ETL 和生产系统也不应该使用 gpadmin 账号。
- **为每个登录的 User 分配不同的 Role。**出于登录和审计的目的，每个被允许登录到 GPDB 的 User 都应该分配属于自己的 Role。对于应用程序(APP)或者 Web Service 来说，应该考虑为每个 APP 或者 Service 建立独立的 Role。
- **使用 Group 来管理访问权限。**
- **控制具备 SUPERUSER 属性的 User 数量。**具有 SUPERUSER 属性的 Role 将可以像 gpadmin 那样绕过 GPDB 的所有权限限制，包括资源队列(Resource Queue/RQ)。应该仅为系统管理员分配 SUPERUSER 权限。

创建用户 User Role

User Role 意味着其可以登录 DB 并发起 DB 会话。因此，在使用 CREATE ROLE 来创建一个 User 时，需要指定 LOGIN 权限。比如：

```
=# CREATE ROLE jsmith WITH LOGIN;
```

Role 可能还需要伴随着一系列的属性从而决定其可以执行哪些 DB 任务。可以在 CREATE ROLE 的时候指定这些属性，也可以在 CREATE 之后使用 ALTER ROLE 命令来完成。

ALTER ROLE 属性

属性	描述
SUPERUSER | NOSUPERUSER	SUPERUSER 可以绕过所有所有权限限制，SUPERUSER 使用有风险，请仅在需要的时候使用。CREATE ROLE 时默认属性为 NOSUPERUSER
CREATEROLE | NOCREATEROLE	是否具备 CREATE DATABASE 权限。默认为 NOCREATEROLE。
CREATEROLE | NOCREATEROLE	是否有 CREATE 和管理其他 ROLE 的权限，默认为 NOCREATEROLE。
INHERIT | NOINHERIT	决定权限是否被其子成员继承。默认属性为 INHERIT。
LOGIN | NOLOGIN	决定 ROLE 是否可以登录 DB。具备 LOGIN 属性的 ROLE 就是 USER。不具备 LOGIN 属性的 ROLE 往往用来做权限管理(GROUP)。默认值为 NOLOGIN。
CONNECTION LIMIT connlimit	对于可以 LOG IN 的 ROLE 来说，决定其最多可同时有多少个连接。默认值为-1(无限制)。
PASSWORD ‘password’	设置 ROLE 的 PASSWORD。如果不打算使用密码登录，可忽略该属性，如果不指定密码，PASSWORD 会被设置为 NULL 并且始终无法登录。空密码可以明确定义 PASSWORD NULL。
ENCRYPTED | UNENCRYPTED	指定密码是否加密，默认行为受 password_encryption 参数决定(默认为 MD5)。如果目前的密码已经使用了加密存储，应忽略该属性。
VALID UNTIL ‘timestamp’	设置在指定的日期后该 ROLE 的密码失效。不设置的情况下为永远有效。
RESOURCE QUEUE queue_name	将 ROLE 分配到指定的资源队列(RQ)。所有的语句都受到该 RQ 的约束。需要注意的是该属性不会被继承，必须为每个 USER 指定该属性。默认的是 pg_default。
DENY {deny_interval | deny_point}	定义允许登录的时间段。

比如下面的例子：

```
=# ALTER ROLE jsmith WITH PASSWORD 'passwd123';
=# ALTER ROLE admin VALID UNTIL 'infinity';
=# ALTER ROLE jsmith LOGIN;
=# ALTER ROLE jsmith RESOURCE QUEUE adhoc;
```

```
=# ALTER ROLE jsmith DENY DAY 'Sunday';
```

除了上述的一些定义属性外，ROLE 还可以配置一些 Server 层面的属性值，比如设置默认的搜索路径：

```
=# ALTER ROLE admin SET search_path TO myschema, public;
```

创建组 Group Role

对于管理一组 User 来说，将它们绑定到一个 Group 是很方便的。通过这种方式，一组 User 可以通过一个 Group 来统一授予权限和回收权限。在 GPDB 中，通过 CREATE ROLE 的方式来创建 GROUP，并通过 GRANT MEMBERSHIP 的方式为 USER 分组。

通过 CREATE ROLE 命令创建新的 GROUP ROLE：

```
=# CREATE ROLE admin CREATEROLE CREATEDB;
```

一旦 GROUP 创建好之后，就可以通过 GRANT 和 REVOKE 添加或者删除 Member(USER ROLE)了：

```
=# GRANT admin TO john, sally;
=# REVOKE admin FROM bob;
```

为了合理的管理对象权限，需要将合适的权限赋予 GROUP ROLE。因为其所有的 USER ROLE 成员都会继承该 GROUP ROLE 的对象权限。比如：

```
=# GRANT ALL ON TABLE mytable TO admin;
=# GRANT ALL ON SCHEMA myschema TO admin;
=# GRANT ALL ON DATABASE mydb TO admin;
```

不过，对于 ROLE 的 LOGIN、SUPERUSER、CREATEDB 和 CREATEROLE 属性是不会像普通的权限一样被继承，需要被明确指定。GROUP 的 USER 成员需要明确的使用 SET ROLE 来授权这些属性。比如，admin ROLE 具备 CREATEDB 和 CREATEROLE 属性，sally 是 admin 的成员，其可以通过下面的语句来获取 admin ROLE 的属性：

```
=> SET ROLE admin;
```

管理对象权限

当一个对象(Table、View、Sequence、Database、Function、Language、Schema、Tablespace)被创建时，其会被分配一个所有者(Owner)。通常 Owner 是执行了该创建语句的 User。对于大多数(Object)的对象来说，默认只有其 Owner(或者 SUPERUSER)可以对其做任何操作。要允许其他 User 使用该对象，需要使用 Grant。GPDB 对于每种对象支持的权限为：

对象类型	权限
Tables、Views、Sequences	SELECT INSERT UPDATE DELETE RULE ALL
External Tables	SELECT RULE ALL

Databases	CONNECT CREATE TEMPORARY | TEMP ALL
Functions	EXECUTE
Procedural Languages	USAGE
Schemas	CREATE USAGE ALL

注意：每个对象的权限必须被独立的授权。比如：把一个 DB 的 ALL 授权不等于把该 DB 的所有对象都授权了。这仅仅是授权了 DB 自身的全部权限(CONNECT、CREATE、TEMPORARY)。

使用 GRANT SQL 命令给指定的 Role 授权一个对象权限。比如：

```
=# GRANT INSERT ON mytable TO jsmith;
```

还可以通过 DROP OWNED 和 REASSIGN OWNED 命令来取消 Role 的 Owner 权限(只有该对象的 Owner 或者 SUPERUSER 可以执行这样的操作)。例如：

```
=# REASSIGN OWNED BY sally TO bob;
=# DROP OWNED BY visitor;
```

模拟 Row 或者 Column 级别的权限控制

GPDB 本身不支持 Row 和 Column 级别的访问权限控制，而只是支持对象(Object)级别的权限控制。

Row 和 Column 级别的访问控制可以通过 View 的方式来模拟。Row 级别的访问控制可以通过添加一个 Column 的形式来存储敏感信息，使用基于该 Column 的 View 来控制访问的 Row，再将该 View 授权给相应的 User。Column 的访问控制也可以通过选择一系列的 Column 作为 View 授权给相应的 User 来模拟。

数据加密

PostgreSQL 提供了一个加密/解密函数包叫做 pgcrypto，其同样可以在 GPDB 中安装使用。默认情况下 GPDB 是不安装 pgcrypto 包的，不过可以在 EMC Download Center 下载该包。并使用 GP 的包管理命令(gppkg)在分布集群上安装 pgcrypto。

pgcrypto 允许 DB 管理员将 Table 特定的 Column 存储为加密格式。在为敏感数据增加了进一步的加密处理之后，存储在 GPDB 中的数据是无法被没有密钥的用户访问的，并且无法直接从磁盘访问加密后的数据文件。

值得注意的是，pgcrypto 函数运行在 DB Server 内部。数据的密码在 pgcrypto 之间和 Client App 之间可能会通过明文传输。为了安全期间，可以考虑在 Client 和 GP Master 之间使用 SSL 安全连接。

密码加密

在 GPDB4.2.1 版本之前，密码默认使用 MD5 进行加密。由于诸多用户要求更高级别的加密算法(Federal Information Processing Standard140-2)，从 4.2.1 版本开始，GPDB 开始使用 SHA-256 加密。

基于时间的登录认证

GPDB 允许管理员将 Role 的访问限制在一定的时间段。使用 CREATE ROLE 或者 ALTER ROLE 命令来执行基于时间的限制。

访问限制可以控制到具体时间点，且约束的改变不需要删除或者重建 ROLE。为权限的管理提高了灵活性。

时间约束仅仅对于设置的 Role 有效。如果一个 Role 包含在另一个受时间约束的 Role，该时间约束不会被继承。

时间约束仅仅是在 LOGIN 的时候有效。SET ROLE 和 SET SESSION AUTHORIZATION 命令对于时间约束不受任何影响，也就是说，即便执行了这些语句也无法继承时间约束设置。

需要的权限

要设置时间约束限制，SUPERUSER 或者 CREATEROLE 权限是必须的。另外没有任何 User 可以给 SUPERUSER 设置时间约束。

如何添加时间约束

有两种办法添加时间约束。在 CREATE ROLE 或者 ALTER ROLE 的时候使用 DENY 关键字并跟随如下的命令来实现：

- 某天或者某个时间访问限制，比如：no access on Wednesdays。
- 一个有开始时间和结束时间的访问限制，比如：no access from Wednesday 10 p.m. through Thursday at 8 a.m.

还可以指定多个限制，比如：no access Wednesdays at any time and no access on Fridays between 3:00 p.m. and 5:00 p.m.

指明日期和时间

有两种方法指明哪一天。使用 DAY 关键字并紧跟英文的星期几或者 0~6 的数字，如下表所示：

英文表述	数字表述
DAY ‘Sunday’	DAY 0
DAY ‘Monday’	DAY 1
DAY ‘Tuesday’	DAY 2
DAY ‘Wednesday’	DAY 3

DAY ‘Thursday’	DAY 4
DAY ‘Friday’	DAY 5
DAY ‘Saturday’	DAY 6

每日中的时间可以使用 12 小时或者 24 小时格式。在 TIME 关键字之后跟随单引号引起来的时间格式。仅仅含有小时和分钟的时间即可，且使用冒号(:)作为分隔符。如果使用 12 小时格式，需要指定 AM 或者 PM 来结尾以确定上下午。下面的例子表明了几种时间格式：

TIME ’14:00’ (24 小时格式的时间)

TIME ’02:00 PM’ (12 消失格式的时间)

TIME ’02:00’ (24 消失格式的时间) 其等价于 TIME ’02:00 AM’

注意：时间约束是强制以服务器时间为基准的。时区信息被忽略。

指定时间间隔

要指定限制访问的时间间隔，需要两个[日期/时间]来确定，且通过 BETWEEN 和 AND 关键字连接。DAY 是必须的。

BETWEEN DAY ‘Monday’ AND DAY ‘Tuesday’

BETWEEN DAY ‘Monday’ TIME ’00:00’ AND DAY ‘Monday’ TIME ’01:00’

BETWEEN DAY ‘Monday’ TIME ’12:00 AM’ AND DAY ‘Tuesday’ TIME ’02:00 AM’

BETWEEN DAY ‘Monday’ TIME ’00:00’ AND DAY ‘Tuesday’ TIME ’02:00’

BETWEEN DAY 1 TIME ’00:00’ AND DAY 2 TIME ’02:00’

最后 3 句是等价的。

注意：日期间隔不能跨越 Saturday(周六)。

Incorrect：DENY BETWEEN DAY ‘Saturday’ AND DAY ‘Sunday’

正确语法为：

DENY DAY ‘Saturday’

DENY DAY ‘Sunday’

例子：

下面的例子说明在 CREATE ROLE 和 ALTER ROLE 的时候使用时间约束。这里只是展示了时间约束部分的命令。关于 CREATE ROLE 和 ALTER ROLE 的细节可参考相关章节。

例 1-创建包含时间约束的 ROLE，限制周末访问。

CREATE ROLE generaluser DENY DAY ‘Saturday’ DENY DAY ‘Sunday’…

例 2-修改 ROLE 添加时间约束，每天晚上 2:00 到 4:00 限制访问。

ALTER ROLE generaluser

DENY BETWEEN DAY ‘Monday’ TIME ’02:00’ AND DAY ‘Monday’ TIME ’04:00’

DENY BETWEEN DAY ‘Tuesday’ TIME ’02:00’ AND DAY ‘Tuesday’ TIME ’04:00’

DENY BETWEEN DAY ‘Wednesday’ TIME ’02:00’ AND DAY ‘Wednesday’ TIME ’04:00’

DENY BETWEEN DAY ‘Thursday’ TIME ’02:00’ AND DAY ‘Thursday’ TIME ’04:00’

DENY BETWEEN DAY ‘Friday’ TIME ’02:00’ AND DAY ‘Friday’ TIME ’04:00’

DENY BETWEEN DAY ‘Saturday’ TIME ’02:00’ AND DAY ‘Saturday’ TIME ’04:00’

DENY BETWEEN DAY ‘Sunday’ TIME ’02:00’ AND DAY ‘Sunday’ TIME ’04:00’

例 3-修改 ROLE 添加时间约束，周三或者周五下午 3:00 到 5:00 限制访问。

ALTER ROLE generaluser

DENY BETWEEN DAY ‘Wednesday’

DENY BETWEEN DAY ‘Friday’ TIME ’15:00’ AND DAY ‘Friday’ TIME ’17:00’

删除时间约束

要删除时间约束，可使用 ALTER ROLE 命令。跟着 DROP DENY FOR，并跟日期/时间。

DROP DENY FOR DAY ‘Sunday’

任何与该条件有交集的约束都会被移除。比如存在的约束为 BETWEEN DAY ‘Monday’ AND DAY ‘Tuesday’。那么删除’Monday’限制时会移除整个限制。原则是有交集即移除。例如：

ALTER ROLE generaluser DROP DENY FOR DAY ‘Monday’…

第六章：配置客户端认证

在 GPDB 系统初始化之后，系统包含一个预定义的 SUPERUSER ROLE。该 USER 的 USER NAME 与初始化 GPDB 系统的 OS USER 同名。该 ROLE 被称为 gpadmin。默认情况下，系统会被设置为只允许 gpadmin 从本地连接。为了让其他 ROLE 可以连接，或者允许从远程主机连接，必须配置 GPDB 来允许这些连接。本章介绍如何配置客户端连接和 GPDB 认证。

- 允许连接到 GPDB
- 限制并发连接

允许连接到 GPDB

客户端的访问认证是通过一个叫做 pg_hba.conf(也是标准的 PostgreSQL 的认证文件)的配置文件来控制的。关于该文件的细节可以参考 PostgreSQL 的文档。

在 GPDB 中，Master 的 pg_hba.conf 文件控制着客户端连接到 GPDB 系统的认证。在 Instance 上也存在 pg_hba.conf 文件，通常该文件已经被正确配置为允许从 Master 访问。不过就译者的经验来说，也出现过其配置错误的情况，该情况会导致 gpexpand 或 gp_dump 之类的操作出错并失败。通常来说，Instance 是不需要接受外部客户端连接的，不太有必要修改其 pg_hba.conf 文件。

pg_hba.conf 是包含每行一条记录的平面文件。空行被忽略，任何在井号(#)后的符号串都会被忽略。每行记录由一系列 Space 和 Tab 混合分割的字段组成。如果需要在字段中出现空白字符，需要将字段用引号引起来。记录不可跨行。每个远程客户端访问权限记录都像这种格式：

```
host database role CIDR-address authentication-method
```

而每个 UNIX 嵌套连接的访问权限记录像这种格式：

```
local database role authentication-method
```

这些字段的含义如下：

字段	描述
local	匹配 UNIX 嵌套连接。如果没有这种记录，UNIX 嵌套连接是不被允许的。
host	匹配 TCP/IP 方式的连接。除非该 Server 属于一个合适的 IP 段，否则其访问是不被允许的。
hostssl	匹配 TCP/IP 方式的 SSL 加密连接。这个配置需要配合 SSL 参数的设置，该参数在 GPDB 启动时生效。
hostnossl	匹配 TCP/IP 方式的非 SSL 加密连接。
database	设置该记录匹配的 DB Name。all 可以匹配全部 DB。多个 DB Name 可以使用逗号(,)分割。或者使用@符号跟随文件名的方式指定，该文件包含需要匹配的 DB Name。
role	匹配哪个 ROLE。all 可以匹配全部的 ROLE。如果想把一个 GROUP 的所有成员匹配上，可以在 ROLE Name 前使用加号(+)表示。多个 ROLE Name 可以使用逗号(,)分割。或者使用@符号跟随文件名的方式指定，该文件包含需要匹配的 ROLE Name。

CIDR-address	指定该记录匹配的客户端 IP 地址范围。其包含一个标准的逗号分割 IP 地址和一个掩码长度值。IP 地址只能使用数字形式，不可以使用域名或者 Hostname。掩码长度表示 IP 地址高位与客户端 IP 匹配的长度。指定的掩码长度右边的二进制 IP 地址位必须是 0。IP 地址与分隔符(/)和掩码长度之间不可以有任何的空字符。比如 172.20.143.89/32。其只能匹配 172.20.143.89IP 地址。172.20.143.0/24 可以匹配 172.20.143 开始的任何 IP 地址。要匹配单个 IP 地址 IPv4 使用 32 作为掩码长度，IPv6 使用 128 作为掩码长度。
IP-address IP-mask	通过标准子网掩码的格式作为掩码长度的可选方案。其被作为一个单独的字段。255.0.0.0 等效于 IPv4 的 8 位掩码长度。255.255.255.255 等效于 IPv4 的 32 位掩码长度。
authentication-method	指定连接时使用的认证方法。比如 trust 为不需要密码，md5 为使用 md5 加密认证。更多细节可以查看 PostgreSQL8.4 的文档中认证方法的部分。

编辑 pg_hba.conf 文件

下面的例子展示如何编辑 Master 上的 pg_hba.conf 文件从而允许远程的客户端通过加密认证的方式访问数据库。

编辑 pg_hba.conf 文件

1. 使用文本编辑器(比如 VI)打开$MASTER_DATA_DIRECTORY/pg_hba.conf 文件。
2. 为每类需要允许的连接添加一行记录。记录是被顺序读取的，所有记录的顺序是有象征性意义的。通常前面的记录匹配更少的连接但要求较弱的认证，后面的记录匹配更多的连接但要求更严格的认证。比如：

```
# allow the gpadmin user local access to all databases
# using ident authentication
local all gpadmin ident sameuser
host all gpadmin 127.0.0.1/32 ident
host all gpadmin ::1/128 ident
# allow the 'dba' role access to any database from any
# host with IP address 192.168.x.x and use md5 encrypted
# passwords to authenticate the user
# Note that to use SHA-256 encryption, replace md5 with
# password in the line below
host all dba 192.168.0.0/32 md5
# allow all roles access to any database from any
# host and use ldap to authenticate the user. Greenplum role
# names must match the LDAP common name.
host all all 192.168.0.0/32 ldap ldapserver=usldap1 ldapport=1389 ldapprefix="cn="
ldapsuffix=",ou=People,dc=company,dc=com"
```

3. 保存并关闭文件
4. 重新加载 pg_hba.conf 文件从而使得刚刚的修改生效：

```
$ gpstop –u
```

限制并发连接

为了限制对 GPDB 系统的并发访问，可以通过配置 Server 参数 max_connections 来实现。这是一个本地化参数，就是说，需要把 Master，Standby 以及所有的 Instance 都修改该参数。通常建议 Instance 的值是 Master 的 5-10 倍，不过这个规律并非总是如此，在 max_connections 比较大的时候通常没有这么高的倍数，2-3 倍也是允许的，但无论如何 Instance 的值不能小于 Master。在设置 max_connections 时，其依赖的参数 max_prepared_transactions 参数也需要修改，该参数的值至少要和 Master 上的 max_connections 值一样大，另外 Instance 上的值需与 Master 相同。

例如：

在$MASTER_DATA_DIRECTORY/postgresql.conf(包括 Standby)：

max_connections=100

max_prepared_transactions=100

在所有的 Instance 上 SEGMENT_DATA_DIRECTORY/postgresql.conf：

max_connections=500

max_prepared_transactions=100

修改最大连接数

1. 停掉 GPDB 系统：
   $ gpstop
2. 在 Master 上编辑$MASTER_DATA_DIRECTORY/postgresql.conf 文件并修改下面两个参数：
   max_connections (允许连接的数量+superuser_reserved_connections)
   max_prepared_transactions (大于或等于max_connections)
3. 在每个 Instance 上编辑 SEGMENT_DATA_DIRECTORY/postgresql.conf 文件并修改下面两个参数：
   max_connections (建议5-10倍Master上的值)
   max_prepared_transactions (大于等于Master上的值)
4. 重启 GPDB 系统
   $ gpstart

注意：增加该参数可能需要更多的共享内存。为了降低影响，可以考虑减少其他内存相关的参数，比如 gp_cached_segworkers_threshold。

客户端/服务端间的加密连接

GPDB 原生支持客户端与 Master 服务端之间的 SSL 连接。SSL 连接可以有效的防止第三方对包的窥探，防止中间层的攻击。在非安全连接环境中有必要使用 SSL，且在使用权限认证时更为必要。使用 SSL 需要在客户端和 Master 端都安装有 OpenSSL。GP 在设置参数 ssl=on(在 Master 的 postgresql.conf 文件)后启动就开启了 SSL。在使用 SSL 模式启动时，GP 会查找 Master 目录下的 server.key(服务器密钥)文件和 server.crt(服务器证书)文件。这些文件必须被正确的安装，否则 GP 系统将无法启动。

重要提示：不要为 server.key 设置访问口令。GP 不会为密钥提示输入口令，这样会导致

出错并无法启动 DB 系统。

关于如何安装 OpenSSL，这里不做翻译解释，可参考源英文手册或相关文档。通常 GP 都是作为后台 OLAP 库部署在安全的网络环境，不太有必要开启 SSL。

第七章：访问数据库

本章介绍可以使用哪些客户端工具连接 GPDB，以及如何建立数据库会话：

- 建立数据库会话
- 支持的客户端应用
- 连接故障排除

建立数据库会话

用户可以使用任何 PostgreSQL 兼容的客户端程序连接到 GPDB，比如 psql。用户或者管理员总是通过 Master 连接到 GPDB，Instance 不能使用客户端连接。

要连接到 GPDB 的 Master，需要知道下面这些连接参数并在客户端程序配置正确。

连接参数	描述	环境变量
Application name	连接到数据库的应用名称	$PGAPPNAME
Database name	需要连接的数据库名称。对于新初始化的系统来说，首次可以使用 template1。	$PGDATABASE
Host name	要连接的 GPDB Master 的主机名称。默认为 localhost。	$PGHOST
Port	GPDB Master 上 Instance 的端口号。默认为 5432.	$PGPORT
User name	要连接的用户名。其没必要与 OS 的 User Name 相匹配。在不知道 User Name 的情况下最好联系询问 GP 管理员。每个 GPDB 系统都有一个初始化时产生的 SUPERUSER。该 User Name 与初始化的 OS User Name 相同(通常为 gpadmin)。	$PGUSER

支持的客户端应用

可以使用这些客户端应用连接 GPDB：

- 在 GPDB 安装时提供的客户端应用。psql 提供了交互式的命令行方式访问 GPDB。
- 针对 GPDB 的 pgAdminIII 作为一种强化版本支持 GP。从 1.10.0 版本开始，PostgreSQL 客户端工具 pgAdminIII 开始支持 GP 特性。安装包可以从 pgAdmin 网站下载。
- 使用标准数据库应用接口，比如 ODBC、JDBC，用户可以开发出自己的客户端程序。由于 GPDB 基于 PostgreSQL 而来，可以直接使用 PostgreSQL 驱动访问 GPDB。
- 诸多使用标准数据库应用接口，比如 ODBC、JDBC 的客户端程序，可以通过配置的方式连接到 GPDB。

GPDB 的客户端应用程序

在 GPDB 安装时在 Master 主机的$GPHOME/bin 下有一系列的客户端应用程序。下面是一些常用的客户端应用程序：

名称	用途
createdb	创建新的数据库
createlang	创建新的程序语言
createuser	创建新的数据库 ROLE
dropdb	删除数据库

droplang	删除程序语言
psql	PostgreSQL 交互式终端
reindexdb	将数据库重建索引
vacuumdb	收集数据库的磁盘空间并分析数据库

在使用这些客户端应用程序时，必须通过 GP Master 实例。还必须知道目标 DB Name、Host Name、Port 以及连接使用的 User Name。这些参数在命令行可以分别使用-d、-h、-p 和-U 提供。没有指定任何选项的参数会优先被解读为 DB Name。

那些有缺省值的参数可以不指定。缺省的 Host Name 是 localhost。缺省的 Port 是 5432。缺省的 User Name 是当前的 OS User Name，会被当作缺省的 DB Name。GPDB 的 User Name 与 OS User Name 未必相同。

如果缺省参数值是错误的，可以选择将正确的值保存在环境变量 PGDATABASE、PGHOST、PGPORT、PGUSER 中。在~/.pgpass 中设置合适的值可以免除反复的密码输入的麻烦。

使用 psql 连接

根据缺省值或者环境变量，下面的例子说明如何通过 psql 连接：

```
$ psql -d gpdatabase -h master_host -p 5432 -U gpadmin
$ psql gpdatabase
$ psql
```

如果还没有用户数据库存在，可以连接系统数据库 template1。例如：

```
$ psql template1
```

在成功连接到数据库之后，psql 会出现一个提示符，包含连接的 DB Name 和一串字符(=>)(或者=#，紧当该用户是 SUPERUSER 时)。例如：

```
gpdatabase=>
```

在提示符处，可以输入 SQL 命令执行了。SQL 命令必须已分号(;)结尾才能将命令发给 Master 并得到执行。比如：

```
=> SELECT * FROM mytable;
```

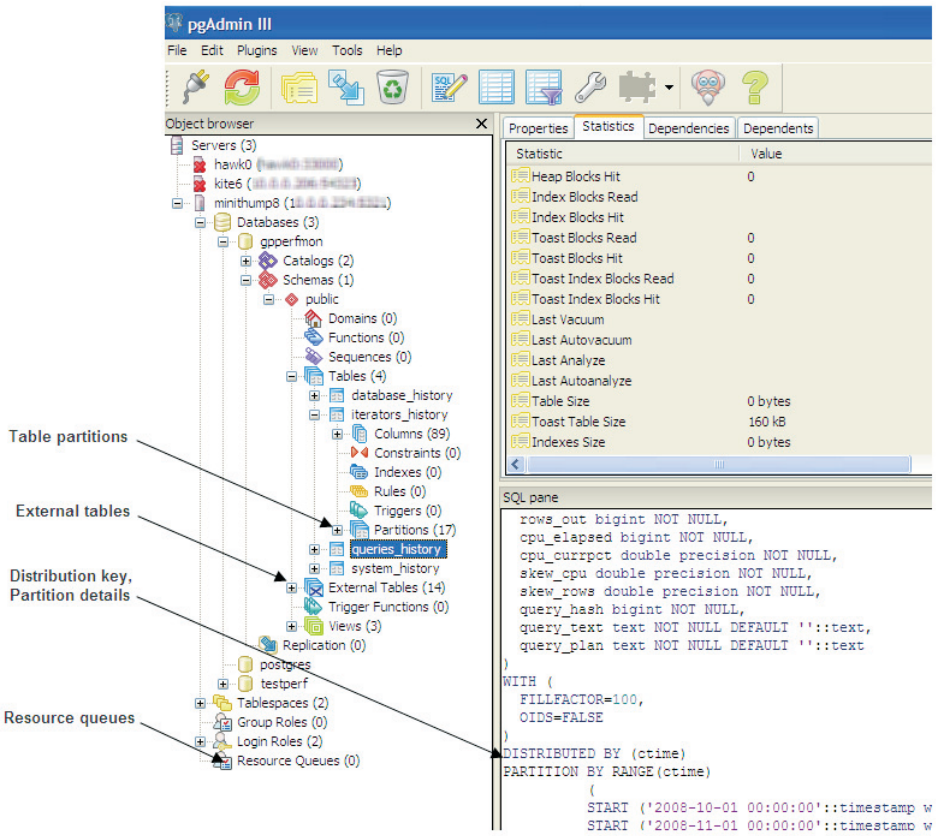
要得到关于 psql 客户端应用程序的更多信息，可以查看相关附录章节。

针对 GPDB 的 pgAdminIII

如果更喜欢图形化界面(有谁不喜欢吗)，可以使用针对 GPDB 的 pgAdminIII。该 GUI 客户端除了支持标准 PostgreSQL 外，还支持一些 GP 的专有特性。

针对 GPDB 的 pgAdminIII 支持下列的 GP 专有特性：

- 外部表(External tables)
- 只读表(Append-only table)，压缩只读表(Compressed append-only table)
- 图形化的解释器(EPLAIN ANALYZE)
- GP Server 的参数配置



安装针对 GPDB 的 pgAdminIII

安装包可以从 pgAdminIII 的官方网站(http://www.pgadmin.org)下载。安装包中包含安装说明。其实和普通的 Windows 安装文件一样，就是诸多的 Next 和选择安装位置等。

针对 GPDB 的 pgAdminIII 文档

普通的帮助信息可以在 **Help**>**Help contents** 菜单中查看。

关于 GP 方面的 SQL 帮助，可以点击 **Help**>**Greenplum Database Help** 菜单查看，在联网状态下，将会自动跳转到 Greenplum SQL 参考文档，只是在 Greenplum 被 EMC 收购整合后这个在线帮助已经打不开了。另外还可以安装 GP 客户端工具包，其包含了 pgAdminIII 连接的参考文档。

使用 pgAdminIII 执行管理操作

该节介绍诸多 GPDB 管理操作中的两个重要部分：编辑服务器配置，图形化查看查询计划。

编辑服务器配置

pgAdminIII提供了两种修改Server配置文件postgresql.conf的方式：本地，通过**File**菜单，远程，通过**Tools**菜单。修改远程参数的方便之处在于不需要将postgresql.conf文件再上传到Server端。

远程编辑服务器配置

1. 连接到需要修改的Server。如果连接了多个Server，要确保已经选中需要修改的Server。
2. 选择**Tools**>**Server Configuration**>**postgresql.conf**菜单。配置信息将会

以列表的形式打开。

3. 双击需要修改的参数打开一个参数设置对话框。
4. 输入新的参数值或者[选择/取消]**Enable**。修改好之后点击**OK**按钮。
5. 如果修改的参数可以通过重新加载配置的方式生效，点击绿色的重载按钮即可。有些参数修改是需要GP完全重新启动才生效。

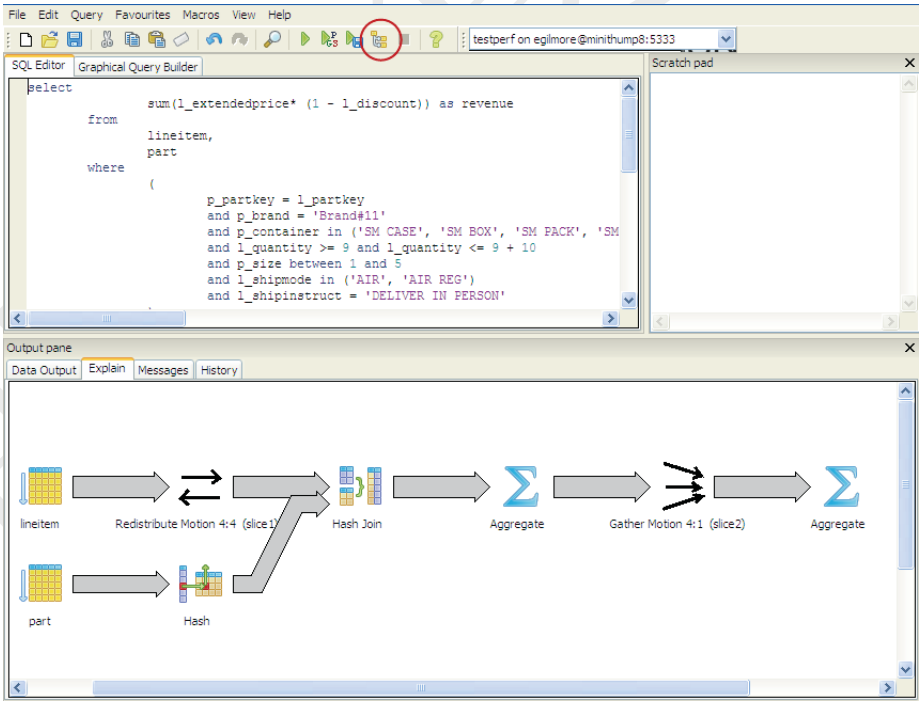
图形化查看查询计划

使用pgAdminIII工具，可以通过执行EXPLAIN命令查看查询计划。输出内容包括GP的分布式查询处理如计划分割和Instance之间数据移动。还可以查看图形化的查询计划。

查看图形化的查询计划

1. 在正确的数据库连接下，选择**Tools**>**Query Tools**。
2. 使用SQL编辑器输入查询语句，还可以通过图形化对象编辑器或者打开一个SQL文件的方式。
3. 选择**Query**>**Explain Option**确认下面的选项：
   - Verbose-如果想查看图形化的查询计划，需要取消该选项
   - Analyze-如果你想在查看查询计划时就直接执行查询语句的话可以选择该选项
4. 点击查询面板上端的查询计划按钮，或者选择**Query**>**Explain**。

查询计划在屏幕的底部展现。比如：



DB 应用程序接口

若需要开发针对GPDB的应用程序，PostgreSQL提供的一些通用的API同样可以应用在GPDB上。这些驱动包并没有与GPDB一起发布，而是一些独立的PostgreSQL项目，需要单独下载和安装配置从而连接GPDB。下面这些驱动可以获取：

API	PostgreSQL Driver	下载连接
ODBC	pgodbc	在 GPDB 的 Connectivity 包中有。可以在 EMC 下载中心下载。没有 EMC 的 Powerlink 账号等于没用。
JDBC	pgjdbc	在 GPDB 的 Connectivity 包中有。可以在 EMC 下载中心下载。没有 EMC 的 Powerlink 账号等于没用。
Perl DBI	pgperl	http://gborg.postgresql.org/project/pgperl
Python DBI	pygresql	http://www.pygresql.org

使用通用API来访问GPDB的说明：

1. 下载相应的语言平台和对应的API。比如下载JDK和JDBC。
2. 编写相应的程序连接GPDB。需要注意SQL的语法支持问题。

下载合适的PostgreSQL驱动并配置到GPDB Master Instance的连接。GP提供了一个客户端包，其包含了支持的GPDB驱动。可以在EMC下载中心下载。没有EMC的Powerlink账号等于没用。

第三方客户端工具

很多第三方的ETL和BI工具使用标准的API如ODBC、JDBC，都可以通过配置连接到GPDB。下面这些工具经过用户证实可以很好的协同GP一同工作：

- Business Objects
- Microstrategy
- Informatica Power Center
- Microsoft SQL Server Integration Services (SSIS) and Reporting Services (SSRS)
- Ascential Datastage
- SAS
- Cognos

GP专业服务可以协助用户配置他们选定的第三方工具协同GP工作。

连接故障排除

有很多导致客户端程序无法成功连接GPDB的原因。本节介绍一些常见的问题并说明如何解决这些问题。

问题	解决办法
No pg_hba.conf entry for host or user	要允许远程客户端连接到 GPDB，必须正确的配置 GPDB Master Instance 的 pg_hba.conf 文件。参见”允许连接到 GPDB”。
Greenplum Database is not running	在 GPDB Master Instance 失败的情况下，用户是无法连接的。可以通过在 GP Master 上使用 gpstate 工具检查 GPDB 系统是否已经启动。
Network problems Interconnect timeouts	从远程连接 GP Master 时，网络问题可能会妨碍连接，比如 DNS 解析错误。要排除这种问题可先 ping Master 主机，先确保需要连接的主机可以 ping 通 Master。另外需要分清 localhost 与实际的 Host Name。当然有时候还存在网络防火墙的问题，这种情况下，可以 ping 通 Master，在 Master 主机通过 psql 可以连接数据库，但从客户端连接 Master 无法连接，此时应该找相关的网络管理人员了。

Too many clients already	缺省情况下，GPDB 设置 Master 最大连接数是 250，Segment 为 750。超过这个限制的请求会被拒绝。该限制是有 postgresql.conf 文件中的 max_connections 参数配置的。若修改 Master 上的该参数，还必须修改 Segment 上该参数为合适的值。

第八章：管理工作负载与资源

本章介绍GPDB的工作负载管理的特征，资源队列(Resource Queue/RQ)的创建和管理。本章包含以下内容：

- GP工作负载管理概述
- 配置负载管理
- 创建资源队列
- 分配ROLE(User)到资源队列
- 修改资源队列
- 检查资源队列状态

GP 工作负载管理概述

工作负载管理的目的是控制同时活动的查询数量以避免造成系统资源耗尽，比如Memory、CPU、磁盘I/O。在GPDB中已经具备基于ROLE体系的资源队列。资源队列可以限制该队列中ROLE执行查询的数量。还可以指定共享CPU资源的使用优先级。将ROLE分配到合适的资源队列，管理员将可以有效的控制用户的查询并避免系统超负荷运行。

GPDB 中资源队列如何工作

在安装GPDB时资源调度是缺省打开的。所有的ROLE都必须分配到资源队列。如果管理员创建ROLE时没有指定只愿队列，该ROLE将会被分配到缺省的资源队列pg_default。

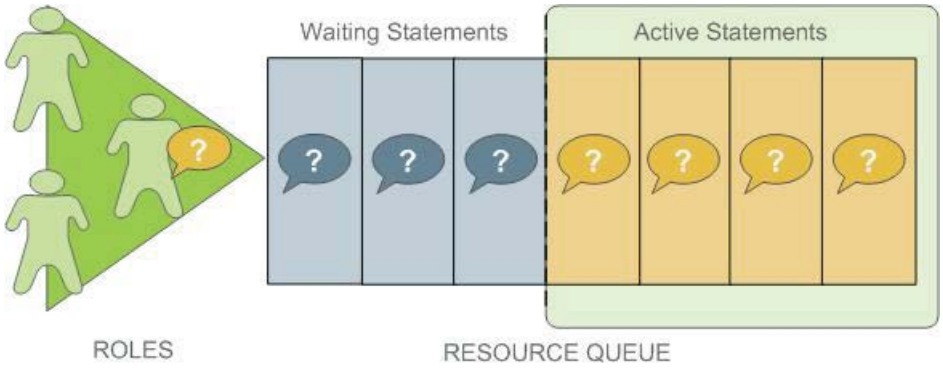
GP建议管理员为不同类型工作负载创建结构性独立的资源队列。比如，可以为高级用户、WEB用户、报表管理等创建不同的资源队列。可以根据相关工作的负载压力设置合适的资源队列限制。目前资源队列的限制包括：

- 活动语句数量。并行执行的最大语句数量。
- 活动语句内存使用量。所有提交的语句使用的总内存不能超过该限制。
- 活动语句优先级。该值设定了该资源队列相对于其他资源队列在CPU资源使用上的优先级，这里说的优先是相对的。
- 活动语句的成本。该值是由查询规划器做的成本评估，该值以涉及的磁盘页(disk page)作为计量单位。

资源队列创建好之后，ROLE(User)可被分配到合适的资源队列。一个资源队列可以分配多个ROLE，但每个ROLE只能被分配到一个资源队列。

资源队列如何工作

在运行时，用户提交一个查询，该查询会被针对其资源队列的限制进行评估。如果评估认为该查询不会超过资源限制，该查询将被立即执行。如果评估认为该查询超过了资源限制(比如最大活动语句数的查询正在执行)，该查询需要等到有足够的资源时才能得到执行。查询按照先进先出的方式排队。在查询优先级启用的情况下，系统会定期的重新分配计算资源。可参见”优先级如何工作”。



SUPERUSER是不受资源队列限制的。超级用户的查询语句总是被立即执行，不管其所在的资源队列如何限制。

内存限制如何工作

在资源队列上设置的内存限制使得每个Instance上该资源队列能够使用的内存总和不能超过设定的最大值。每个查询语句分配的内存大小是资源队列的内存限制除以最大活动语句数量(GP建议与活动语句数限制结合使用，而不是与成本限制结合使用)。比如，资源队列的内存限制为2000MB，活动语句数限制为10，那么每条执行语句可以分配到200MB的内存。缺省的内存分配可以针对每条语句通过设置statement_mem参数来复写(最大可以达到资源队列限制的值)。一旦一条语句开始执行，其分配的内存一直到执行结束才会释放(即便其实际使用的内存小于分配的内存)。

关于资源队列中内存限制的详细信息以及内存使用控制，可参考”创建内存限制的资源队列”。

执行优先级如何工作

资源限制是针对活动语句来说的，内存和成本的限制属于是否许可类型，其决定查询语句是允许进入查询状态还是保持排队状态。在语句处于活动状态时，其需要分享CPU资源，这部分的资源由资源队列的优先级控制。当一个更高优先级的语句进入运行状态时，其要求获得更多的CPU资源，相应的需要减少其他运行中语句的CPU资源。

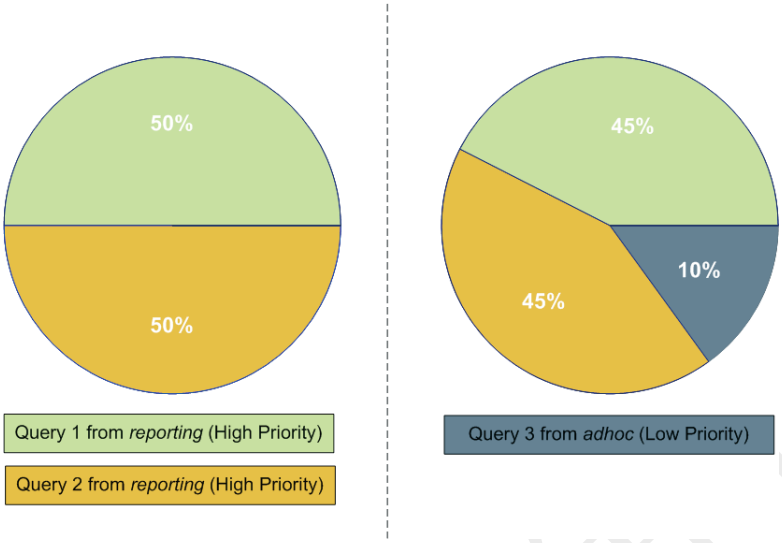
语句的规模和复杂程度不会影响到CPU资源的分配。比如一个简单低成本的查询与一个庞大复杂的查询同时执行却有着相同的优先级，他们将分配到相同的CPU资源。当一个新的语句开始后，CPU资源分配的比重需要被重新计算，不过，相同优先级的语句之间获得的CPU资源仍然是相同的。

例如，管理员想要创建3个资源队列：adhoc用于做持续查询的业务分析，reporting用于定期的报表工作，executive用于高级用户查询。管理员希望确保定期报表工作不受到adhoc分析查询不可预测的资源消耗影响，希望高级用户提交的查询能够获得更多计算资源。因此，资源队列优先级可以设置为这样：

- adhoc – Low priority
- reporting – High priority
- executive – Maximum priority

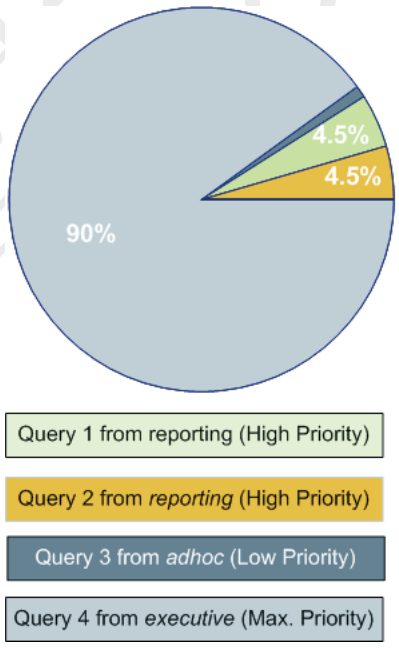
运行时，CPU资源由正在运行语句的优先级决定如何分配。如果语句1与语句2来自reporting队列且同时运行，他们将获得相同CPU资源。当一个来自adhoc队

列语句开始运行，其需要较少的CPU资源。CPU资源的分配将做调整，他们之间的比重受他们的优先级决定：



注意：该图显示的是粗略的百分比。在CPU的使用上，并不总是按照精确计算来分配资源。

当一个来自executive的语句开始执行，CPU资源将根据他们的优先级重新调整。该语句相对于adhoc和reporting来说可能是很简单的，在其执行完之前，其仍获取最大份额的CPU资源。



资源队列评估的语句类型

并不是所有的SQL语句都被资源队列评估和限制。缺省状态下只有SELECT、SELECT INTO、CREATE TABLE AS SELECT和DECLARE CURSOR语句被评估限制。若将Server端参数resource_select_only设置为off，INSERT、UPDATE和DELETE语句也将被评估和限制。

开启工作负载管理的步骤

在GPDB中开启工作负载管理涉及到如下几个高级任务：

1. 创建资源队列并设置合适的限制
2. 为User Role指定资源队列
3. 使用工作负载管理相关的系统视图监控和管理资源队列

配置工作负载管理

在安装GPDB时资源调度缺省是开启的，并强制所有ROLE遵从。缺省的资源队列是pg_default，活动语句数量限制为20，成本(Cost)无限制，内存(Memory)无限制，中(Medium)优先级。GP建议创建不同类型的资源队列。

配置工作负载管理

1. 以下为一般的资源队列配置参数
   - max_resource_queues – 设置最多可以有多少个资源队列
   - max_resource_portals_per_transaction – 设置每个事务最多可以打开几个游标(Cursor)。值得注意的是每个游标需要占用资源队列的一个活动查询。
   - resource_select_only – 若设置为on，SELECT、SELECT INTO、CREATE TABLE AS SELECT和DECLARE CURSOR语句被评估限制。若设置为off，INSERT、UPDATE和DELETE语句也将被评估和限制。
   - resource_cleanup_gangs_on_wait – 在开始一个新的查询之前先清空所在资源队列中其他空闲的工作进程。
   - stats_queue_level – 激活资源队列使用信息收集，这样就可以通过查询系统视图pg_stat_resqueues查看。
2. 以下参数与内存使用有关：
   - gp_resqueue_memory_policy – 激活GP内存管理特性。设置为none的情况下内存管理与4.1版本之前相同。设置为auto内存受statement_mem和资源队列内存限制的控制。缺省为eager_free，译者认为该模式利于合理调整内存使用量。
   - statement_mem与max_statement_mem – 用于为每个活动语句分配内存(可以复写资源队列的缺省值)。max_statement_mem由SUPERUSER设置，应考虑避免普通用户的超负荷使用。在任何时候statement_mem都必须小于max_statement_mem。
   - gp_vmem_protect_limit – 限制每个Segment Instance上所有语句可以使用的物理内存总量的上限值。导致内存使用超过该上限的语句会被取消掉(Cancel)而得不到执行。译者认为该参数要根据具体硬件情况进行合理的评估，从OS层面来说，内存颗粒的容量用完之后会使用SWAP充当内存，对于普通磁盘来说，不到万不得已，强烈建议不要使用SWAP。
   - gp_vmem_idle_resource_timeout 与 gp_vmem_protect_segworker_cache_limit – 用于释放Instance上空闲DB进程的内存。为了提高并发量管理员可以考虑调整这些配置。详细说明可参考相关附录。
3. 以下参数与查询优先级有关。注意，这些参数都是本地(LOCAL)参数，必须修改所有Instance的postgresql.conf文件：
   - gp_resqueue_priority – 缺省状态下查询优先级特性开启。

- gp_resqueue_priority_sweeper_interval – 设置CPU为所有活动语句重新计算CPU资源分配的时间。缺省值通常已经可以满足常规的DB操作。
- gp_resqueue_priority_cpucores_per_segment – 设置每个Instance使用的CPU核数。缺省状态下Instance为4而Master为24，这对于EMC DCA是正确的。该参数是LOCAL参数。该参数对于Master也是有影响的，需要配置到一个较高的值。比如，在一个集群中，每个Host主机有8个CPU核且每个Segment Host有4个Instance，那可以按照如下配置：

  Master和Standby

  gp_resqueue_priority_cpucores_per_segment = 8

  Instance

  gp_resqueue_priority_cpucores_per_segment = 2

重要提示：如果每个Segment Host主机上配置的Instance数量低于CPU核数，请确保将该参数调整到一个合适的值。过低的值可能会导致CPU资源利用不足。

4. 要查看和修改这个工作负载管理参数，还可以使用gpconfig命令。译者认为，往往都是使用这个命令，很少有人直接去改postgresql.conf文件的，不过如果由于参数修改不当导致GPDB系统无法启动，目前这个命令是无能为力了。
5. 比如，要查看一个参数值：

   $ gpconfig --show gp_vmem_protect_limit
6. 比如，要修改一个参数的值，且Master的值与Segment不同：

   $ gpconfig -c gp_resqueue_priority_cpucores_per_segment -v 2 -m 8
7. 重启GPDB以确保修改的参数生效(本节介绍的参数都需要重启才能生效)：

   $ gpstop –r

创建资源队列

创建资源队列涉及到Name、成本、活动语句数量、执行优先级等。通过CREATE RESOURCE QUEUE命令来创建新的资源队列。

创建含活动语句数量的资源队列

资源队列通过设置ACTIVE_STATEMENTS控制活动语句的数量。例如，创建一个名称为adhoc活动语句数量为3的资源队列：

```
=# CREATE RESOURCE QUEUE adhoc WITH (ACTIVE_STATEMENTS=3);
```

这意味着，分配到adhoc资源队列的所有ROLE，在同一时刻最多只能有3个语句处于执行状态。如果该队列当前已经有3个语句正在执行，在该队列的ROLE提交第4个语句时，其将处于等待状态，直到前面3个语句有一个执行完。

创建含内存限制的资源队列

资源队列通过设置MEMORY_LIMIT控制该队列所有语句可以使用的内存总量。每个主机上所有Instance可以获得的物理内存总数不得超过该主机的物理内存总数。GP建议将MEMORY_LIMIT控制在该Instance可以得的的物理内存总数的90%以下。比如，一个Host主机有48GB的物理内存，有6个Instance，那么每个Instance可以获得的物理内存为8GB。这样就可以简单的得到

MEMORY_LIMIT=0.9*8GB=7.2GB。如果存在多个资源队列，他们的MEMORY_LIMIT总和应被控制为7.2GB。

当与ACTIVE_STATEMENTS结合使用时，缺省每个语句获得的内存为：MEMORY_LIMIT / ACTIVE_STATEMENTS。当与MAX_COST结合使用时，缺省的内存分配为：MEMORY_LIMIT * (query_cost / MAX_COST)。GP推荐与ACTIVE_STATEMENTS结合使用而不是与MAX_COST结合使用。

比如，创建一个活动语句数量为10，内存限制为2000MB的资源队列(每个语句在执行时在每个Instance上获得200MB的内存)：

```
=# CREATE RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=10, MEMORY_LIMIT='2000MB');
```

缺省的内存分配可以在每个语句通过设置statement_mem参数复写，但不可超过MEMORY_LIMIT和max_statement_mem设定的值。比如，分配更多的内存：

```
=> SET statement_mem='2GB';
=> SELECT * FROM my_big_table WHERE column='value' ORDER BY id;
=> RESET statement_mem;
```

通常来说，MEMORY_LIMIT的设置建议所有的资源队列的总和不要超过Instance可以获得的物理内存总数。如果不同类型语句之间交错执行尚且可以，但仍需要留意，如果某个Instance超出了内存限制超，相关语句会被取消。

创建含成本限制的资源队列

资源队列通过设置MAX_COST限制被执行的语句可消耗的最大成本(Cost)。Cost以一个浮点数(如100.0或使用科学计数法如1e+2)来指定。

Cost是GP查询规划器(如使用EXPLAIN查看)评估出来的总预估成本。因此管理员在设置时需要对该系统执行的查询很熟悉才可以得到一个恰当的Cost阀值。Cost意味着对磁盘的操作数量。1.0等于获取一个磁盘页(disk page)。

例如，创建一个Cost阀值为100000.0 (1e+5)，名称为webuser的资源队列：

```
=# CREATE RESOURCE QUEUE webuser WITH (MAX _COST=100000.0);
```

或者

```
=# CREATE RESOURCE QUEUE webuser WITH (MAX _COST=1e+5);
```

这意味着，分配到webuser资源队列的所有ROLE执行的全部语句Cost总和不能超过100000.0的限制。比如，有200个Cost为500.0的语句正在执行，第201个Cost为1000.0的语句提交后只能等到空闲的Cost足够时才能得到执行。

允许在系统空闲时执行语句

若一个资源队列配置了Cost阀值，管理员可以允许COST_OVERCOMMIT(这是缺省设置)。在系统没有其他语句执行时，超过资源队列Cost阀值的语句可以被执行。而当有其他语句在执行时，Cost阀值仍被强制执行。

如果COST_OVERCOMMIT被设置为false，超过Cost阀值的语句将永远被拒绝。

允许小查询绕过队列限制

可能存在一些工作负载很小的查询，管理员希望其不占用资源队列的活动语句

数量而允许得到执行。比如，一些检索系统表的语句不涉及大的资源消耗甚至不需要与Instance进行交互。管理员可以设置MIN_COST指明什么样的开销作为小查询。那些低于MIN_COST的语句将立即得到执行。MIN_COST不仅可以同最大Cost一起使用，还可以和活动语句数量一起使用。例如：

```
=# CREATE RESOURCE QUEUE adhoc WITH (ACTIVE_STATEMENTS=10, MIN_COST=100.0);
```

设置优先级级别

为了控制CPU资源的使用，管理员可以设置合适的优先级。在并发争用CPU资源时，高优先级资源队列中的语句将可以获得比低优先级资源队列中的语句更多的CPU资源。

优先级可以在CREATE RESOURCE QUEUE和ALTER RESOURCE QUEUE的时候通过WITH来设置。例如，为adhoc和reporting队列指定优先级，管理员可以使用下面的命令：

```
=# ALTER RESOURCE QUEUE adhoc WITH (PRIORITY=LOW);
=# ALTER RESOURCE QUEUE reporting WITH (PRIORITY=HIGH);
```

创建最高优先级的队列executive，管理员可以使用下面的命令：

```
=# CREATE RESOURCE QUEUE executive WITH (ACTIVE_STATEMENTS=3, PRIORITY=MAX);
```

在优查询优先级特性开启时，资源队列的优先级缺省为MEDIUM。

重要提示：要使得资源队列的优先级设置在执行语句中强制生效，必须确保优先级特性的相关参数已经设置好。参见”配置工作负载管理”。

分配 ROLE(User)到资源队列

一旦资源队列被创建好了，就需要把ROLE(User)分配到合适的资源队列。如果ROLE没有被显式的分配到一个资源队列，其将被分配到缺省的资源队列pg_default。缺省的资源队列含20个活动语句数量和MEDIUM的优先级。

使用ALTER ROLE或者CREATE ROLE命令来分配ROLE到资源队列。比如：

```
=# ALTER ROLE name RESOURCE QUEUE queue_name;
=# CREATE ROLE name WITH LOGIN RESOURCE QUEUE queue_name;
```

每个ROLE同一时间只能被分配到一个资源队列，可以使用ALTER ROLE命令修改ROLE的资源队列。

资源队列的分配必须通过逐个User的方式进行。如果有一个层级较高的ROLE(比如GROUP ROLE)，将给ROLE分配到一个资源队列并不会将其包含的User分配到该资源队列。

SUPERUSER总是不受资源队列限制的。SUPERUSER的查询总是可以立即得到执行，而不管资源队列的限制如何设置。

从资源队列中移除 ROLE

所有ROLE都需要分配到资源队列。如果没有被显式分配到指定的资源队列，

该ROLE将会进入缺省资源队列pg_default。如果想将ROLE从现有资源队列中移除并放到缺省队列中，可将其资源队列分配到none。比如：

```
=# ALTER ROLE role_name RESOURCE QUEUE none;
```

修改资源队列

在创建资源队列后，可以使用ALTER RESOURCE QUEUE命令来改变或者重置队列的限制。还可以使用DROP RESOURCE QUEUE命令删除资源队列。

变更资源队列

使用ALTER RESOURCE QUEUE命令来改变资源队列的限制。一个资源队列必须包含ACTIVE_STATEMENTS或者MAX_COST(或者都包含)。变更资源队列，执行新的值。例如：

```
=# ALTER RESOURCE QUEUE adhoc WITH (ACTIVE_STATEMENTS=5);
=# ALTER RESOURCE QUEUE exec WITH (MAX_COST=100000.0);
```

要将活动语句数量或者内存限制重置为无限制，可以使用-1值。要重置Cost门槛为无限制，可以设置为-1值。比如：

```
=# ALTER RESOURCE QUEUE adhoc WITH (MAX_COST=-1.0, MEMORY_LIMIT='2GB');
```

可以使用ALTER RESOURCE QUEUE命令改变查询优先级。比如，设置一个资源队列的优先级为最低级别：

```
ALTER RESOURCE QUEUE webuser WITH (PRIORITY=MIN);
```

删除资源队列

使用DROP RESOURCE QUEUE命令删除资源队列。要删除一个资源队列，该资源队列不能与任何ROLE相关，或者队列中有语句正等待执行。删除一个资源队列：

```
=# DROP RESOURCE QUEUE name;
```

检查资源队列状态

检查资源队列状态涉及下列内容：

- 查看排队语句和资源队列状态
- 查看资源队列统计信息
- 查看分配到资源队列的ROLE
- 查看资源队列中等待的语句
- 清除资源队列中等待的语句
- 查看活动语句的优先级
- 重置活动语句的优先级

查看排队语句和资源队列状态

管理员可以通过查看视图gp_toolkit.gp_resqueue_status来查看资源队列的状态。

该视图展示系统中每个资源队列有多少语句在等待执行，多少语句正在执行。查看资源队列在系统中的创建、限制、当前的状态：

```
=# SELECT * FROM gp_toolkit.gp_resqueue_status;
```

查看资源队列统计信息

如果要追踪资源队列的统计信息和性能，需要为资源队列开启统计信息收集配置。这可以通过配置Master上postgresql.conf文件的这个参数来实现：

```
stats_queue_level = on
```

一旦该配置开启，就可以使用系统视图pg_stat_resqueues来查看资源队列使用的统计信息。注意，开启该配置会带来轻微的资源开销，每个经资源队列执行的语句都会被追踪。开启统计信息收集对于初期的资源队列诊断是有帮助的，而后续的运行应该关闭该参数。

更多关于资源队列统计信息收集的问题可以参考PostgreSQL的相关文档。

查看分配到资源队列的 ROLE

要查看ROLE与资源队列之间的关联关系，使用系统日志表pg_roles和gp_toolkit.gp_resqueue_status来获得：

```
=# SELECT rolname, rsqname FROM pg_roles, gp_toolkit.gp_resqueue_status
WHERE pg_roles.rolresqueue=gp_toolkit.gp_resqueue_status.queueid;
```

或者可以创建一个视图来简化以后的使用。例如：

```
=# CREATE VIEW role2queue AS
SELECT rolname, rsqname FROM pg_roles, pg_resqueue
WHERE pg_roles.rolresqueue=gp_toolkit.gp_resqueue_status.queueid;
```

这样就可以直接查询试图了：

```
=# SELECT * FROM role2queue;
```

查看资源队列中等待的语句

当语句在资源队列中执行时，其会被记录在pg_locks系统日志表中。这里可以查看到所有的活动语句和等待语句。为了检查处于等待状态的语句(即便没有语句在等待)，可以使用gp_toolkit.gp_locks_on_resqueue视图。例如：

```
=# SELECT * FROM gp_toolkit.gp_locks_on_resqueue WHERE lorwaiting='true';
```

若该查询没有结果返回，意味着此时没有语句在资源队列中等待执行。

清除资源队列中等待的语句

有时候可能需要清除资源队列中处于等待状态的语句。比如，想要清除在资源队列中等待还没有得到执行的语句。还有可能想要终止一个已经开始而需要很长时间才能完成的语句，或者该语句处于空闲的事务状态而希望其把资源让给其他需要的ROLE。要达到这些目的，首先需要知道哪些语句需要被清除，确定该进程的ID，然后使用pg_cancel_backend函数来终止该进程。

比如，查看当前处于活动状态或者等待状态的语句：

```
=# SELECT rolname, rsqname, pid, granted, current_query, datname
FROM pg_roles, gp_toolkit.gp_resqueue_status, pg_locks, pg_stat_activity
WHERE pg_roles.rolresqueue=pg_locks.objid
AND pg_locks.objid=gp_toolkit.gp_resqueue_status.queueid
AND pg_stat_activity.procpid=pg_locks.pid;
```

若没有结果返回，意味着当前没有语句处于资源队列中。比如有两个语句在资源队列中可能是这种样子的：

```
rolname  | rsqname | pid     | granted | current_query                | datname
---------------------------------------------------------------------------
Sammy    | webuser | 31861   | t       | <IDLE> in transaction        | namesdb
daria    | webuser | 31905   | f       | SELECT * FROM topten;        | namesdb
```

根据输出结果确定需要清除语句的进程ID(pid)。通过下面的方式清除语句：

```
=# pg_cancel_backend(31905)
```

注意：尽量不要使用OS的KILL命令。译者认为，不是完全不可以用，如果不小心把系统搞崩溃了有能力拯救的话，其实怎么杀都无所谓。

查看活动语句的优先级

在gp_toolkit模式中有个视图gp_resq_priority_statement，其包含了所有正在执行的语句的优先级，会话ID等信息。该视图只能通过gp_toolkit模式访问。

重置活动语句的优先级

SUPERUSER可以在语句运行期间通过内置函数gp_adjust_priority(session_id, statement_count, priority)调整其优先级。通过该函数SUPERUSER可以提升或者降低任何语句的优先级。例如：

```
=# SELECT gp_adjust_priority(752, 24905, 'HIGH')
```

该函数需要获取语句的SESSION ID和Statement Count两个参数，SUPERUSER可以通过gp_toolkit模式的视图gp_resq_priority_statement获得，session_id和statement_count两个参数分别对应 rqpsession和rqpcommand。该函数只对指定的语句有效，同一个资源队列随后的语句仍然使用其预先设定的优先级。

第九章：定义数据库对象

本章介绍GPDB的数据定义语言(DDL)以及如何创建和管理数据库对象。

- 创建与管理数据库
- 创建与管理表空间
- 创建与管理模式
- 创建与管理表
- 分区大表
- 创建与使用序列
- 在GPDB中使用索引
- 创建与管理视图

创建与管理数据库

一个GPDB系统可以有多个数据库(Database)。这与一些DBMS不同(比如Oracle)，它们的Instance就是Database。在GP系统中，虽然可以创建多个DB，但是客户端程序一次只能连接一个DB –不可以跨越DB执行查询语句。

关于数据库模版

每个新的数据库都是基于一个模版创建的。缺省的数据库模版为template1。在初始化GPDB系统初期可以连接到该库。在没有明确指定模版的情况下创建新的数据库将缺省使用该DB作为模版。除非你希望之后创建的DB包含你所创建的对象，请不要在该DB中创建任何对象。

除了template1之外，每个新建的GP系统还包含另外两个模版template0和postgres，这两个DB是系统内部使用的，最好不要删除或者修改。template0模版库可以用来创建仅仅包含标准对象的完全干净的数据库。如果想避免从template1中拷贝任何的对象，可以考虑使用该模版。

创建数据库

通过CREATE DATABASE命令来创建一个新的数据库。例如：

```
=> CREATE DATABASE new_dbname;
```

要创建一个数据库，必须具备创建数据库的权限或者是SUPERUSER身份。若没有正确的权限是无法创建数据库的。需要联系GP管理员授予必要的权限或者帮助创建一个数据库。

还有一个客户端程序createdb可以用来创建数据库。例如，通过命令行终端执行下面的命令将会在指定的Host上创建名为mydatabase的数据库：

```
$ createdb -h masterhost -p 5432 mydatabase
```

克隆一个数据库

缺省状态下，创建数据库是通过克隆数据库模版template1的方式完成的。然而任何的DB都可以作为模版来创建一个新的数据库，因此可以通过指定DB的方

式克隆或者拷贝出一个新的数据库，新的DB包含模版的所有对象和数据。例如：

```
=> CREATE DATABASE new_dbname TEMPLATE old_dbname;
```

查看数据库列表

在psql客户端程序中，直接使用\l指令查看GPDB中包含模版在内的所有DB的列表。使用其他客户端程序时，可以通过查询pg_database系统日志表(必须是SUPERUSER用户)来得到。比如：

```
=> SELECT datname from pg_database;
```

变更数据库

使用ALTER DATABASE命令来改变DB的属性，例如Owner、Name以及缺省配置等。必须是该DB的Owner或者SUPERUSER才可以执行这样的操作。下面的例子演示修改默认的搜索路径：

```
=> ALTER DATABASE mydatabase SET search_path TO myschema, public, pg_catalog;
```

删除数据库

使用DROP DATABASE命令来删除DB。该操作将从系统信息表中删除该DB的信息记录，并删除该DB包含的全部磁盘数据。必须是该DB的Owner或者SUPERUSER才可以执行删除DB操作，其他用户无法删除该DB，有其他用户连接时也无法删除。可以先连接到template1(或者其他DB)，然后再删除需要删除的DB。例如：

```
=> \c template1
=> DROP DATABASE mydatabase;
```

这里同样有一个客户端程序叫做dropdb用以删除DB。例如，通过命令行终端执行下面的命令将会在指定的Host上删除名为mydatabase的数据库：

```
$ dropdb -h masterhost -p 5432 mydatabase
```

警告：删除数据库操作是无法回滚的。慎用该操作！

创建与管理表空间

表空间(tablespace)允许DB管理员使用多个文件系统来存储数据库对象，从而可以决定如何更好的利用他们的物理储存设备。表空间的使用是有好处的，比如在访问频度不同的数据库对象上使用不同性能的磁盘。例如，将经常使用的表放在高性能磁盘的文件系统上(比如SSD固态盘)，而将其他表放在普通硬盘的文件系统上。

一个表空间是需要一个文件系统来存储其数据库文件的。在GPDB中，Master和每个Instance(primary和mirror)都需要独立的存储位置(或者说目录)。所有这些文件系统组件构成了GP系统中所谓的文件空间(filespace)。文件空间定义之后，其可以被多个表空间使用。简言之，在GP系统中，文件空间建立在一系列的文件系统之上，表空间建立在文件空间之上。

创建文件空间

要创建文件空间，首先需要在所有相关的GP Host主机上准备好逻辑文件系统。文件系统位置对于Master和所有的Primary和Mirror来说都是必须的。在准备好了文件系统之后，使用gpfilespace命令来定义文件空间。只有SUPERUSER才能进行该操作。

注意：GP并不直接知晓文件系统的界限，其只是把文件存向指定的位置。因此，在一个逻辑磁盘位置定义多个文件空间是没有意义的，因为根本无法控制文件在逻辑文件系统上储存的具体位置。

使用gpfilespace创建文件空间

1. 使用gpadmin用户登录到GPDB系统的Master主机。
   ```
   $ su - gpadmin
   ```
2. 创建一个文件空间的配置文件：
   ```
   $ gpfilespace -o gpfilespace_config
   ```
3. 将会提示输入一个文件空间的名称，Primary Instance的文件系统位置，Mirror Instance的文件系统位置，Master的文件系统位置。例如，若每个Segment Host配置了2个Primary和2个Mirror，将会提示输入5个文件系统位置(包括Master)。就像这样：
   ```
   Enter a name for this filespace> fastdisk
   primary location 1> /gpfs1/seg1
   primary location 2> /gpfs1/seg2
   mirror location 1> /gpfs2/mir1
   mirror location 2> /gpfs2/mir2
   master location> /gpfs1/master
   ```
4. 该命令将会输出一个配置文件。请再次检查该文件确保其按照期望的那样反映出了想要使用的文件系统位置。
5. 再次执行该命令，基于之前生成的配置文件创建文件空间：
   ```
   $ gpfilespace -c gpfilespace_config
   ```

转移临时文件或事务文件的位置

可以选择将临时文件或事务文件转移到一个特殊的文件空间从而改善DB的查询性能、备份性能、连续存储数据的性能。

临时文件和事务文件默认都是存储在每个Instance(包括Master、Standby、Primary和Mirror)目录下。只有SUPERUSER可以移动该位置。只有gpfilespace工具可以写该文件。

关于临时文件和事务文件

除非另有指明，临时文件和事务文件都是和用户数据放在一起的。缺省的临时文件位置为<filespace_directory>/<tablespace_oid>/<database_oid>/pgsql_tmp，使用gpfilespace --movetempfiles命令将改变该文件的位置。

关于临时文件和事务文件，需要注意以下几点信息：

- 虽然可以使用同一个文件空间存储不同类型文件，但只能为临时文件或者事务文件指定一个文件空间。
- 如果文件空间被临时文件使用，该文件空间将不能被删除。
- 文件空间必须提前被创建好才能使用。

使用gpfilespace移动临时文件

1. 确保文件空间存在，且与存储其他用户数据的文件空间不同。
2. 将GPDB系统停掉，保持停机状态。

注意：任何活动的连接都会导致gpfilespace –movetempfiles操作的失败。

3. 把GPDB启动为限制模式，确保其他用户无法连接，执行下面的命令：

```
gpfilespace --movetempfilespace filespace_name
```

注意：临时文件位置在Instance中配合共享内存使用，在创建、打开、删除临时文件时用到。如果查询用到了临时文件，表明当前的可用内存已经无法满足该查询对内存的需求。很多时候我们宁愿使用临时文件也不选择使用SWAP作为内存扩展方案。除非SWAP的性能比临时文件目录所在文件系统的性能还要高。

使用gpfilespace移动事务文件

1. 确保文件空间存在，且与存储其他用户数据的文件空间不同。
2. 将GPDB系统停掉，保持停机状态。

注意：任何活动的连接都会导致gpfilespace --movetransfiles操作的失败。

3. 把GPDB启动为限制模式，确保其他用户无法连接，执行下面的命令：

```
gpfilespace --movetransfilespace filespace_name
```

注意：事务文件位置在Instance中配合共享内存使用，在创建、打开、删除事务文件时用到。

创建表空间

一旦文件空间创建好了，就可以使用该文件空间定义表空间了。要定义一个表空间，使用CREATE TABLESPACE命令，比如：

```
=# CREATE TABLESPACE fastspace FILESPACE fastdisk;
```

表空间必须由SUPERUSER才可以创建，不过在创建好之后可以允许普通的DB User来使用该表空间。可以将CREATE权限授予相应的用户。例如：

```
=# GRANT CREATE ON TABLESPACE fastspace TO admin;
```

使用表空间存储 DB 对象

表、索引、甚至整个DB都可以指定在特定的表空间。若要如此，拥有给定表空间CREATE权限的Role必须通过表空间的名称作为相关命令的参数来实现，下面是创建一个space1表空间上的表：

```
CREATE TABLE foo(i int) TABLESPACE space1;
```

或者使用缺省表空间参数default_tablespace来设定：

```
SET default_tablespace = space1;
CREATE TABLE foo(i int);
```

在将default_tablespace设置为一个非空字符串后，其相当于给CREATE TABLE和CREATE INDEX命令添加一个TABLESPACE的子句，但却不必显式注明。

如果一个表空间与DB关联，那么其将存储所有该DB的系统日志、临时文件等。此外，其也是在该DB上创建表、索引等缺省的表空间(除非通过TABLESPACE或者default_tablespace参数指定)。如果DB在创建的时候没有与特定的表空间相关联，那么该DB与其使用的模版DB使用相同的表空间。

一旦表空间被创建，将可以被任何DB使用并提供足够的用户访问权限。

查看现有的表空间和文件空间

每个GPDB系统都有两个缺省的表空间：pg_global(用以存储系统日志信息)和pg_default(用以存储template1和template0模版DB的缺省表空间)。这些表空间使用系统缺省的文件空间pg_system(系统初始化时使用的数据目录data directory)。

要获取文件空间的信息，可以查看系统日志表pg_filespace和pg_filespace_entry。可通过与pg_tablespace关联查看表空间的完整定义。例如：

```
=# SELECT spcname as tblspc, fsname as filespc, fsedbid as seg_dbid, fselocation as datadir
    FROM pg_tablespace pgts, pg_filespace pgfs, pg_filespace_entry pgfse
WHERE pgts.spcfsoid=pgfse.fsefsoid AND pgfse.fsefsoid=pgfs.oid ORDER BY tblspc, seg_dbid;
```

删除表空间和文件空间

在表空间相关的所有对象被删除之前，该表空间是不能被删除的。同样的，在相关的表空间被删除之前，文件空间是不能被删除的。

要删除表空间，可通过DROP TABLESPACE命令完成。表空间只能被其Owner和SUPERUSER删除。

要删除文件空间，可通过DROP FILESPACE命令完成。只有SUPERUSER可以删除文件空间。

注意：如果文件空间被临时文件或者事务文件使用，该文件空间将不能被删除

创建与管理模式

模式(Schema)是在DB内组织对象的一种逻辑结构。模式可以允许用户在一个DB内不同的模式之间使用相同Name的对象(比如Table)。

缺省”Public”模式

每个新创建的DB都有一个缺省的模式public。如果没有创建其他的模式，在创建DB对象时将缺省使用public模式。缺省情况下所有的ROLE(User)都有public模式下的CREATE和USAGE权限。而在创建其他模式时，需要将该模式授权给相关的ROLE(User)。

创建模式

使用CREATE SCHEMA命令来创建一个新的模式。例如：

```
=> CREATE SCHEMA myschema;
```

要访问某个模式中的对象，可以通过模式名加圆点加对象名来指明对象所属的模式。比如：

```
schema.table
```

还可以在创建模式的时候将Owner设置为其他ROLE(User)。语法如下：

```
=> CREATE SCHEMA schemaname AUTHORIZATION username;
```

模式搜索路径

要知道在DB的哪个模式下搜索需要的对象，可以通过明确指定模式名的方式来实现。例如：

```
=> SELECT * FROM myschema.mytable;
```

若不想通过指定模式名称的方式来实现，可以通过设置search_path参数来完成。该参数告诉DB其应该在哪些可用的模式中搜索对象。在不指明模式名称的情况下，搜索路径(search_path)列表中的第一个模式将成为缺省模式，比如创建对象等。

设置模式搜索路径

search_path用于设置模式的搜索顺序。该参数可以通过ALTER DATABAST命令修改DB的模式搜索路径。例如：

```
=> ALTER DATABASE mydatabase SET search_path TO myschema, public, pg_catalog;
```

还可以通过ALTER ROLE命令修改特定ROLE(User)的模式搜索路径。例如：

```
=> ALTER ROLE sally SET search_path TO myschema, public, pg_catalog;
```

设置了模式搜索路径之后，在未明确指明模式名称的情况下访问DB对象，将会按照search_path列表的顺序依次在相应的Schema中查找对应的Object，直到找到为止，若在不同的Schema中存在相同Name的Object，DB优先匹配search_path中靠前的Schema下的Object。

查看当前的模式

有时不确定当前所在的模式或者搜索路径。要查看这些信息，可以通过使用current_schema()函数或者SHOW命令来查看。例如：

```
=> SELECT current_schema();
=> SHOW search_path;
```

删除模式

使用DROP SCHEMA命令来删除模式。例如：

```
=> DROP SCHEMA myschema;
```

缺省状态下，只有空的模式才可以被删除。若想要直接删除模式及相关的所有Object(Table、Index、Function等)。使用如下命令：

```
=> DROP SCHEMA myschema CASCADE;
```

系统模式

下面的这些系统级别的模式在所有的DB中都存在：

- pg_catalog模式存储着系统日志表(System Catalog Table)、内置类型(Type)、函数(Function)和运算符(Operator)。该模式无论是否在search_path中指明，都存在search_path中。
- information_schema模式由一个标准化视图构成，其包含DB中对象的信息。该视图用于以标准化的方法从系统日志表中查看系统信息。
- pg_toast模式是一个储存大对象的地方(那些超过页面尺寸(page size)的记录)。该模式仅供GPDB系统内部使用，通常不建议管理员或者任何用户访问。
- pg_bitmapindex视图是一个储存bitmap index对象的地方(值列表等)。该模式仅供GPDB系统内部使用，通常不建议管理员或者任何用户访问。
- pg_aoseg视图是一个储存append-only表的地方。该模式仅供GPDB系统内部使用，通常不建议管理员或者任何用户访问。
- gp_toolkit是一个管理用的模式，可以查看和检索系统日志文件和其他的系统信息。gp_toolkit视图包含一些外部表、视图、函数，可以通过SQL的方式访问它们。gp_toolkit视图对于所有DB User都是可以访问的。更多信息查看”gp_toolkit管理视图”。

创建与管理表

GPDB中的Table除了数据是分布在不同Segment主机这点外，和其他关系型数据库的Table是很类似的。在创建Table时，需要一个额外的SQL语法来指明Table的分布策略。

创建表

CREATE TABLE命令用于创建一张新的Table和定义其结构。在创建Table时，通常需要定义如下几个方面：

- 都有哪些列(Column)以及对应的数据类型。
- Table或者Column的约束(Constraint)，其限定了Table或者Column可以储存什么样的数据。
- Table的分布策略，其决定了Table的Data如何被分割存储在GPDB的各个Segment上。
- Table在Disk上的存储方式。比如压缩、按列存储等选项。
- 大表的分区策略(Partition Table)。

选择Column的数据类型

Column的Data Type决定了其可以储存什么类型的数据值。通常都希望用最小的空间储存数据。但还应该考虑到Data Type对数据的约束。比如，使用Character类型储存字符串，Date或者Timestamp储存日期，Numeric储存数字等。

对于Character类型来说，CHAR、VARCHAR和TEXT之间不存在性能差异，当

然是在不考虑填补空白导致的储存尺寸增加的影响情况下。然而在其他的DB系统中，可能CHAR会表现出最好的性能，但在GPDB中是不存在这种性能优势的。在多数情况下，应该选择使用TEXT或者VARCHAR而不是CHAR。

对于Numeric类型来说，应该尽量选择更小的数据类型来适应数据。比如，选择BIGINT类型来存储SMALLINT类型范围内的数值，会造成空间的大量浪费。

对于打算用来做Table Join的Column来说，应该考虑选择相同的数据类型。如果做Join的Column具有相同的数据类型(比如主键PrimaryKey与外键ForeignKey)，其工作效率会更高。如果两者的数据类型不同，DB还需要将其中一个类型做转换从而可以做关联比较，这种开销是不必要的浪费。

设置Table和Column的约束

数据类型用来限制在Table中可以存储的数据的性质。但对于很多应用来说，数据类型提供的限制粒度太大。SQL标准允许在Table和Column上定义约束。约束将允许在Table的数据上使用更多的限制。如果User试图在Table上储存违反约束的数据将会发生错误。在GPDB中使用约束是有一些限制的，最为显著的是外键(ForeignKey)、主键(PrimaryKey)和唯一约束(Unique Constraint)。其他约束的支持与PostgreSQL相同。

检查约束

检查约束是最常见的约束类型。其通过指定数据必须满足一个布尔表达式来约束。比如，要求产品的价格必须为正数，可以这样：

```
=> CREATE TABLE products ( product_no integer, name text, price numeric CHECK (price > 0) );
```

非空约束

非空约束简单的理解就是不可以存在空(NULL)值。非空约束是一种Column类型的约束。例如：

```
=> CREATE TABLE products (product_no integer NOT NULL, name text NOT NULL, price
numeric );
```

唯一约束

唯一约束可以确保包含某些Column的数据在整个Table中是唯一的。在GPDB中使用唯一约束存在强制条件，Table必须是HASH分布的(而不是DISTRIBUTED RANDOMLY)，并且唯一约束的Column集合必须完整包含所有的DK Column。

```
=> CREATE TABLE products (product_no integer UNIQUE, name text, price numeric)
DISTRIBUTED BY (product_no);
```

主键约束

主键约束就是唯一约束与非空约束的结合体。在GPDB中使用主键约束存在强制条件，Table必须是HASH分布的(而不是DISTRIBUTED RANDOMLY)，并且主键约束的Column集合必须完整包含所有的DK Column。如果一个Table包含主键，那么主键包含的所有Column会缺省成为DK。例如：

```
=> CREATE TABLE products (product_no integer PRIMARY KEY, name text, price numeric)
DISTRIBUTED BY (product_no);
```

外键约束

在目前版本的GPDB中外键约束是没有被支持的。可以定义外键约束，但参照完整性是无效的，就是说DB不会理会定义参照完整性限制。

外键约束要求一个Table中某些Column的值必须在另外一个Table中出现。这样可以保持两个相关表之间的数据完整性。在当前版本的GPDB中，在分布式的Table之间的数据完整性检查是无效的。

选择表的分布策略

GPDB的所有Table都是分布式存储的。在CREATE TABLE和ALTER TABLE的时候有个DISTRIBUTED BY(HASH分布)或DISTRIBUTED RANDOMLY(随机分布)子句用以决定Table的Row数据如何分布。

在选择表的分布策略时需要重点考虑以下几点(依次更重要)：

- **平坦的数据分布** – 为了尽可能达到最好的性能，所有的Instance应该尽量储存等量的数据。若数据的分布不平衡或倾斜，那些储存了较多数据的Instance在处理自己那部分数据时将需要耗费更多的工作量。为了实现数据的平坦分布，可以考虑选择具有唯一性的DK，如主键。往往很多Table是没有唯一键的，译者认为，只需尽量选择数据分布规律且取值范围远远大于Instance数量的Column作为DK即可。
- **本地操作与分布式操作** – 在处理查询时，很多处理如关联、排序、聚合等若能够在Instance本地完成，其效率将远高于跨越系统级别(需在Instance之间交叉传输数据)的操作。当不同的Table使用相同的DK时，在DK上的关联或者排序操作将会以最高效的方式把绝大部分工作在Instance本地完成。本地操作大约比分布式操作快5倍。若Table使用随机分布策略，将大大限制本地操作的可能性，虽然这种方式可以确保数据分布的平坦性。
- **平坦的查询处理** – 在一个查询正被处理时，我们希望所有的Instance都能够处理等量的工作负载，从而尽可能达到最好的性能。有时候查询场景与数据分布策略很不吻合，这时很可能导致工作负载的倾斜。例如，有一张销售交易Table。该Table的DK为公司名称，那么数据分布的HASH算法将基于公司名称的值来计算，假如有一个查询以某个特定的公司名称作为查询条件，该查询任务将仅在一个Instance上执行。若查询时不是指定特定的单位名称，这仍然是个可行的分布策略。译者认为，这个例子只是用以说明查询倾斜是如何发生的，在真实的应用中有时为了达到某种效果还可能会特意这样设计，比如提高并发访问能力。

声明分布键

在创建Table时有一个额外的子句用以指明分布策略。如果在创建Table时没有指明DISTRIBUTED BY或者DISTRIBUTED RANDOMLY子句，GPDB将会依次考虑使用主键(假如该Table有的话)或者第一个字段作为HASH分布的DK。几何类型或者自定义类型的Column是不适合作为GP的DK的。如果一个Table没有一个合适类型的Column作为DK，该表将使用随机分布策略。就译者的经验来说，还有办法使得Table不分布，即像System表那样只存储在Master上，要达到这样的效果，在CREATE TABLE的时候使用空的COLUMN配置且不指定分布策略。这样创建出来的是一张没有意义的空表，之后再通过ALTER TABLE命令添加COLUMN即可。有些特殊场合可以考虑这种选择，除非你已经对GP的工作机制深入了解，一般情况下不建议尝试。

为了确保数据的平坦分布，可能需要选择一个具有唯一性的Column作为DK，如果达不到平坦的效果，也可以选择DISTRIBUTED RANDOMLY策略，但这只应该作为最后的选择。例如：

```
=> CREATE TABLE products (name varchar(40), prod_id integer, supplier_id integer)
    DISTRIBUTED BY (prod_id);
=> CREATE TABLE random_stuff (things text, doodads text, etc text)
    DISTRIBUTED RANDOMLY;
```

选择表的存储模式

GPDB提供几种灵活的存储处理模式(或者混合模式)。在创建一张新的TABLE时，有几个选项来决定数据如何储存在磁盘上。本节介绍这几种选项，以及出于工作负载的考虑如何实现最佳的储存模式。

- 选择堆存储(Heap)或只追加(Append-Only/AO)存储
- 选择行存储(Row-Orientation)或列存储(Column-Orientation)
- 使用压缩(只可以是AO表)
- 检查只追加(AO)表的压缩和分布情况

选择堆存储或者只追加存储

缺省情况下GPDB使用与PostgreSQL相同的存储模式 -- 堆存储。堆存储模式在OLTP类型工作负载的DB中很常用 -- 数据在初始装载后经常变化。UPDATE和DELETE操作需要对ROW级别做版本控制从而确保DB事务处理的可靠性。堆表更适合一些小表，比如维表，这种表可能会在初始化装载后经常更新数据。

GPDB还提供了一种称之为只追加存储模式的TABLE。AO表更适合数据仓库中非规范化事实表，这些表通常都是系统中最大的表。事实表通常是批量装载数据且只进行只读式查询操作。AO表在数据装载后是不支持更新的。将事实表数据存入AO表时不会保留更新相关ROW级别的信息(大约每ROW需要20字节)。AO表达到了更精简和优化的页面存储结构。AO表不允许执行DELETE和UPDATE操作。该存储模式强化了批量数据装载的性能。不推荐一行一行的使用INSERT语句来装载数据。

创建堆表

行存堆表是缺省的存储模式，创建堆表时不需要额外的CREATE TABLE语法。例如：

```
=> CREATE TABLE foo (a int, b text) DISTRIBUTED BY (a);
```

创建只追加表

在CREATE TABLE时使用WITH子句来指明TABLE的存储模式。如果没有指明，该表将会是缺省的行存堆表。例如，要创建一张没有压缩的AO表：

```
=> CREATE TABLE bar (a int, b text) WITH (appendonly=true)
```

选择行存或列存

GPDB提供存储导向的选择：行存或列存(或者混合)。本节提供一些关于正确选择行存或列存的常规指导。不过具体还需根据查询工作负载来确切评估。

从最通用的目的和混合工作负载来考虑，行存可以提供灵活与性能的最佳结合。不过在一些特定的情况下，列存可以提供更好的I/O和存储性能。在考虑使用行

存还是列存时需要考虑以下几点：

- **表数据的更新。**如果一张表在装载完之后一定有更新操作，那么就选择行存表。因为列存表必须是AO表。AO表不可以更新。
- **经常做INSERT操作。**如果经常有数据被INSERT，考虑选择行存表。列存表对于写操作不是最优的，因为每条数据都需要被写到磁盘的多个位置(列存表的每列存储于不同的磁盘文件，而行存表是存储在同一个磁盘文件)。
- **查询涉及的COLUMN数量。**若通常在SELECT或者WHERE中涉及TABLE的全部或大部COLUMN，考虑选择行存表。行存适合在WHERE或HAVING中对单列做聚合操作：

  ```
  SELECT SUM(salary)...
  SELECT AVG(salary)... WHERE salary > 10000
  ```

  或者在WHERE条件中使用单个COLUMN条件且返回相对少量的ROW：

  ```
  SELECT salary, dept ... WHERE state='CA'
  ```

- **TABLE的COLUMN数量。**行存储对于使用COLUMN数量很多或者ROW的数据尺寸相对较小的TABLE来说更高效。列存表在只访问宽表的很少COLUMN的查询中可以表现出更好的性能。
- **压缩。**虽然行存与列存的数据类型完全相同，但在列存表上的一些压缩优势无法应用在行存表上。例如，许多压缩算法都利用相邻相似的数据进行压缩。然而，越深的压缩，其随机访问越困难，因为在读取数据时是需要解压缩的。

创建列存表

在CREATE TABLE时使用WITH子句来指明TABLE的存储模式。如果没有指明，该表将会是缺省的行存堆表。使用列存的TABLE必须是AO表。比如，创建一张列存储的TABLE：

```
=> CREATE TABLE bar (a int, b text) WITH (appendonly=true, orientation=column)
    DISTRIBUTED BY (a);
```

使用压缩(只可以是AO表)

在GPDB中，AO表有两种库内压缩可选，一种是表级的压缩，另外一种是COLUMN级别的压缩，前者应用到整个TABLE，后者应用到指定的COLUMN。在选择COLUMN级别压缩时，可以为不同的COLUMN选择不同的压缩算法。下表是可用的压缩算法：

表导向	可用压缩类型	支持压缩算法
行	表级别	ZLIB 和 QUICKLZ
列	列级别 和 表级别	RLE_TYPE、ZLIB 和 QUICKLZ

使用库内压缩要求Segment系统具备强劲的CPU来压缩和解压缩数据。不要在压缩文件系统使用压缩AO表。如果Instance数据目录是压缩文件系统，不要压缩使用AO表。

在选择AO表的压缩方式和级别时，需要考虑以下几点因素：

- CPU性能
- 压缩比或占用磁盘尺寸
- 压缩速度
- 解压速度或扫表效率

虽然说在考虑压缩表时最小的占用磁盘尺寸是主要目的，但影响压缩和扫表的

CPU性能也是需要重点考虑的。每个系统都有一个最优的压缩设置，应在保证不会显著提高压缩时间和降低扫表效率的前提下最有效的压缩减少数据尺寸。

QUICKLZ压缩通常适用于CPU能力一般的情况，其压缩速度比ZLIB快，但压缩率不如ZLIB。相反的，ZLIB提供更高的压缩率，但压缩速度较低。在压缩级别为1时，QUICKLZ与ZLIB可能获得差不多的压缩率(但压缩速度ZLIB可能差一些)。但在6级以上的ZLIB在压缩率方面的优势显著高于QUICKLZ(但压缩速度也因此显著的低于QUICKLZ)。

压缩AO表的性能相关的因素包括硬件、查询调优等方面。通常建议在特定的环境下选择储存模式时最好根据相应的比较测试的结果来确定。

创建压缩表

在CREATE TABLE时使用WITH子句来指明TABLE的存储模式。使用压缩模式的TABLE必须是AO表。例如，要创建一张5级ZLIB压缩的AO表：

```
=> CREATE TABLE foo (a int, b text)
    WITH (appendonly=true, compresstype=zlib, compresslevel=5);
```

注意：QUICKLZ压缩模式只有一种压缩级别，没有级别选项可以选择。而ZLIB压缩模式有1 – 9个压缩级别可选。

检查AO表的压缩与分布情况

GP提供了内置的函数用以检查AO表的压缩率和分布情况。这两个函数可以使用对象ID或者TABLE的NAME作为参数。表名可能需要带模式名限定。

函数	返回类型	解释
get_ao_distribution(name) get_ao_distribution(oid)	Set of (dbid, tuplecount) rows	展示 AO 表的分布情况，每 ROW 对应 Segment Instance 的 dbid 与储存的数据行数。
get_ao_compression_ratio(name) get_ao_compression_ratio(oid)	`float8`	计算出 AO 表的压缩率。如果该信息未得到，将返回-1 值。

压缩率得到的是一个常见的比值类型。比如，3.19的返回值或者3.19:1，意味着该TABLE未压缩状态下的储存尺寸是压缩下的储存尺寸的3倍多。

分布信息展示的是每个Instance存储该TABLE的ROW数量。例如，在一个有着4个Instance的系统，其dbid范围为0 – 3，该函数返回类似下面的结果集：

```
=# SELECT get_ao_distribution('lineitem_comp');
get_ao_distribution
--------------------
(0,7500721)
(1,7501365)
(2,7499978)
(3,7497731)
(4 rows)
```

支持运行长度编码

GPDB已支持COLUMN级别的运行长度编码(Run-length Encoding /RLE)压缩算法。RLE是一种将连续重复的数据作为一种计数方式存储的压缩算法。RLE对于重复元素是很有效的。比如，在一个表中有两个COLUMN，一个日期

COLUMN和一个描述COLUMN，其中包含200000个date1和400000个data2，RLE压缩处理这种数据为类似data1 200000 data2 400000这样的效果。对于那些没有很多重复值的数据RLE是不适合的，而且还可能会显著的增加存储文件的尺寸。

RLE压缩有4种级别。级别越高，压缩效率越高，但压缩速度也会越低。

使用了RLE压缩的行存表对4.2.1之前的版本是不兼容的。若需要将这些表备份并在之前的版本上恢复，可在恢复操作前，先将这些表ALTER为无压缩或者旧版本兼容的压缩模式(ZLIB或QUICKLZ)，再执行恢复操作，因为RLE模式的表定义在4.2.1之前的版本不兼容。

使用列级压缩

在CREATE TABLE、ALTER TABLE和CREATE TYPE命令中包含对COLUMN设置压缩类型、压缩级别和块尺寸(Block Size)的选项。这些参数统称为存储参数。存储参数可用于行导向和列导向的AO表。下面列举这3种存储参数及每种参数的可选值。

名称	解释	可选值
COMPRESSTYPE	使用的压缩类型	ZLIB(更高压缩) QUICKLZ(更快压缩) RLE_TYPE(运行长度编码) none(无压缩、缺省值)
COMPRESSLEVEL	压缩级别	ZLIB 为 1-9 级可选 1 级压缩较快但压缩率较低，9 级压缩较慢但压缩率较高
		QUICKLZ 仅 1 个级别可选(缺省不需指定)
		RLE_TYPE 为 1-4 级可选 1 级压缩较快但压缩率较低，4 级压缩较慢但压缩率较高
BLOCKSIZE	表的存储块大小	8192 – 209715(8K – 2M)该值必须是 8192 的倍数

使用存储参数的格式如下：

```
[ ENCODING ( storage_directive [,…] ) ]
```

这里ENCODING关键字是必须的，存储参数包含3个部分：参数名称、等于号、参数值。要指定多个存储参数，用逗号(,)分割即可。存储参数可以应用在单独的COLUMN上，还可以作为所有COLUMN的默认设置，如下面的CREATE TABLE语句所示：

一般用法：

```
column_name data_type ENCODING ( storage_directive [, … ] ), …
COLUMN column_name ENCODING ( storage_directive [, … ] ), …
DEFAULT COLUMN ENCODING ( storage_directive [, … ] )
```

例如：

```
C1 char ENCODING (compresstype=quicklz, blocksize=65536)
COLUMN C1 ENCODING (compresstype=quicklz, blocksize=65536)
DEFAULT COLUMN ENCODING (compresstype=quicklz)
```

缺省压缩属性

在未指定压缩方式、压缩级别和块尺寸的情况下，缺省不使用压缩存储，而块大小使用系统配置参数block_size指定的值。

压缩设置的优先级

COLUMN的压缩设置通过TABLE向分区再到子分区传递。在越低级别的设置具有越高的优先级。

- 子分区层面的COLUMN压缩设置将复写分区、COLUMN和TABLE层面的设置。
- 分区层面的COLUMN压缩设置将复写COLUMN和TABLE层面的设置。
- COLUMN层面的压缩设置将复写整个TABLE层面的设置。

注意：包含存储参数或者COLUMN层面的存储参数的表不可以被继承(INHERIT)。若使用LIKE子句来创建TABLE，表层面的存储参数以及COLUMN层面的存储参数都会被忽略。也就是说，如果你要为一张分区表新增一个分区的话，需要为该分区指定你所期望的压缩设置，而不要指望其可以从父表继承。

列压缩设置的最佳定位

最佳的方法是根据不同的数据设置特定的列压缩。下面的例5展示了2级分区表在子分区上使用RLE_TYPE压缩。

存储参数的例子

下面的例子展示了在使用CREATE TABLE语句时使用存储参数。

例1

该例子中，COLUMN c1使用ZLIB压缩并使用系统定义的块尺寸。COLUMN c2使用QUICKLZ压缩并使用块尺寸为65536。COLUMN c3不使用压缩且使用系统定义的块尺寸。

```
CREATE TABLE T1 (c1 int ENCODING (compresstype=zlib),
        c2 char ENCODING (compresstype=quicklz, blocksize=65536),
        c3 char)
WITH (appendonly=true, orientation=column);
```

例2

该例子中，COLUMN c1使用ZLIB压缩并使用系统定义的块尺寸。COLUMN c2使用QUICKLZ压缩并使用块尺寸为65536。COLUMN c3使用RLE_TYPE压缩并使用系统定义的块尺寸。

```
CREATE TABLE T2 (c1 int ENCODING (compresstype=zlib),
        c2 char ENCODING (compresstype=quicklz, blocksize=65536),
        c3 char, COLUMN c3 ENCODING (RLE_TYPE) )
WITH (appendonly=true, orientation=column)
```

例3

该例子中，COLUMN c1使用ZLIB压缩并使用系统定义的块尺寸。COLUMN c2使用QUICKLZ压缩并使用块尺寸为65536。COLUMN c3使用RLE_TYPE压缩并使用系统定义的块尺寸。值得注意的是在子分区的定义中，COLUMN c3使用了ZLIB(不是RLE_TYPE)压缩，由于分区中的COLUMN存储设置的优先级比TABLE层面的COLUMN存储设置的优先级高，实际上c3使用的是ZLIB压缩而非RLE_TYPE压缩。

```
CREATE TABLE T3 (c1 int ENCODING (compresstype=zlib),
        c2 char ENCODING (compresstype=quicklz, blocksize=65536),
        c3 char,COLUMN c3 ENCODING (compresstype=RLE_TYPE))
WITH (appendonly=true, orientation=column)
PARTITION BY RANGE (c3) (
        START ('1900-01-01'::DATE) END ('2100-12-31'::DATE),
```

```
        COLUMN c3 ENCODING (zlib)
    );
```

例4

该例子中，创建TABLE时，COLUMN c1直接指定了存储设置。而COLUMN c2没有指定存储设置，因此其将从DEFAULT COLUMN ENCODING子句继承压缩方式(QUICKLZ)和块尺寸(65536)。

COLUMN c3指定了压缩方式为RLE_TYPE，而块尺寸(65536)从DEFAULT COLUMN ENCODING子句继承而来。

COLUMN c4没有压缩。因为缺省列存储设置指定了压缩模式，COLUMN c4需要显式的指定压缩模式为none来复写默认的压缩设置。而块尺寸没有显式的复写设置，因此，其块尺寸为65536.

```
CREATE TABLE T4 (c1 int ENCODING (compresstype=zlib),
        c2 char,
        c3 char,
        c4 smallint ENCODING (compresstype=none),
        DEFAULT COLUMN ENCODING (compresstype=quicklz, blocksize=65536),
        COLUMN c3 ENCODING (compresstype=RLE_TYPE) )
    WITH (appendonly=true, orientation=column);
```

例5

该例子中，创建一个2层分区表。如果COLUMN j的值为1或者2，该ROW将进入RLE_TYPE压缩的分区，如果COLUMN j的值为其他，该ROW将进入ZLIB压缩的分区。所有其他的COLUMN使用QUICKLZ压缩。

```
CREATE TABLE T5 ( i int,
            j int,
            k date,
            DEFAULT COLUMN ENCODING (blocksize=1048576) )
        WITH (appendonly = true, orientation=column)
        PARTITION BY RANGE(k)
        SUBPARTITION BY LIST(j)
        SUBPARTITION TEMPLATE
            (PARTITION one_two VALUES(1, 2) COLUMN j ENCODING (compresstype=RLE_TYPE),
            PARTITION rest VALUES(3, 4, 5, 6, 7, 8, 9, …)
                COLUMN j ENCODING (compresstype=zlib, compresslevel=9),
            DEFAULT COLUMN ENCODING (compresstype=quicklz) )
(
        START (date '2011-01-01') END (date '2011-12-31')
        EVERY (interval '1 day')
);
```

若在已有的TABLE上设置COLUMN的压缩设置，使用ALTER TABLE命令。

通过TYPE命令的方式设置压缩配置

一个TYPE可以包含3个压缩参数。关于添加这些参数到TYPE的语法和限制，参考相关的CREATE TYPE命令。下面的命令使用精简的方式创建压缩表。

```
CREATE TABLE t2 (c1 comptype) WITH (APPENDONLY=true, ORIENTATION=column);
```

这里的comptype的定义为：

```
CREATE TYPE comptype (
    internallength = 4,
    input = comptype_in,
    output = comptype_out,
    alignment = int4,
    default = 123,
    passedbyvalue,
    compresstype="quicklz",
    blocksize=65536,
    compresslevel=1
);
```

注意：译者不建议使用这种不明显的方式，虽然在定义TABLE时看起来精简了不少，但对于别人来说，阅读和理解可能都存在障碍。另外替代原生TYPE的定义未必适应所有情况。建议慎用。

选择块尺寸

在一个TABLE中，每个块尺寸意味着相应数量byte的存储。块尺寸必须在8192到2097152之间，并且必须是8192的整数倍。缺省值为32768。需要注意的是，指定大的块大小会消耗大量的内存资源。块尺寸决定着存储层的尺寸，在GP中，每个块作为一部分数据来维护，因此多分区表和列存储表都会消耗更多的内存。

变更表

ALTER TABLE命令用于改变现有表的定义。通过ALTER TABLE命令可以改变TABLE的各种属性，如：列定义、分布策略、存储模式和分区结构(可参见”维护分区表”章节)等。例如，将TABLE的一个COLUMN添加一个非空限制：

```
=> ALTER TABLE address ALTER COLUMN street SET NOT NULL;
```

改变表的分布

ALTER TABLE命令提供了改变分布策略的选项。在修改TABLE的分布策略时，表中的数据要在磁盘上做重分布，该操作可能需要密集的资源消耗。还有一个按照现有策略重新分布数据的选项。

修改分布策略

ALTER TABLE命令可用于改变表的分布策略。对于分区表来说，修改分布策略会递归的应用于所有的子分区。该操作不会改变表的OWNER以及其他TABLE属性。例如，下面的命令在所有Segment之间按照customer_id作为DK重分布sales表：

```
ALTER TABLE sales SET DISTRIBUTED BY (customer_id);
```

在修改TABLE的HASH分布时，表数据会自动重新分布。然而，将分布策略改为随机分布时不会重新分布数据。例如：

```
ALTER TABLE sales SET DISTRIBUTED RANDOMLY;
```

重分布表数据

对于随机分布策略或者不改变分布策略的表，要重分布TABLE的数据，使用REORGANIZW=TRUE。这在处理数据倾斜问题时可能是很必要的，在添加新的Segment节点资源时也是必要的。比如：

```
ALTER TABLE sales SET WITH (REORGANIZE=TRUE);
```

该命令会在Instance之间按照现有的分布策略(包括随机分布策略)重新平衡表中数据。

修改表的存储模式

在TABLE被创建之后，修改表的存储模式是不可能的。存储模式只能在CREATE TABLE时被指定。如果要修改现有表的存储模式，必须使用正确的存储选项重建该表，重新加载数据到新的表，删除旧的表，修改新表为旧的表名。另外还必须重新授权表的权限。例如：

```
CREATE TABLE sales2 (LIKE sales)
WITH (appendonly=true, compresstype=quicklz, compresslevel=1, orientation=column);

INSERT INTO sales2 SELECT * FROM sales;
DROP TABLE sales;
ALTER TABLE sales2 RENAME TO sales;
GRANT ALL PRIVILEGES ON sales TO admin;
GRANT SELECT ON sales TO guest;
```

在现有表上添加压缩列

可以使用ALTER TABLE命令来添加一个压缩列。关于压缩列的设置，参见”使用列级压缩”章节。下面的例子演示了在现有的T1表上增加zlib压缩列：

```
ALTER TABLE T1 ADD COLUMN c4 int DEFAULT 0 ENCODING (COMPRESSTYPE=zlib);
```

继承压缩设置

在增加一个子分区(非一级分区)时，新的分区将继承子分区的压缩设置。下面的例子演示了创建一个带子分区设置的表，然后增加一个分区：

```
CREATE TABLE ccddl (i int, j int, k int, l int)
    WITH (APPENDONLY = TRUE, ORIENTATION=COLUMN)
    PARTITION BY range(j)
    SUBPARTITION BY list (k)
    SUBPARTITION template(
        SUBPARTITION sp1 values(1, 2, 3, 4, 5),
        COLUMN i ENCODING(COMPRESSTYPE=ZLIB),
        COLUMN j ENCODING(COMPRESSTYPE=QUICKLZ),
        COLUMN k ENCODING(COMPRESSTYPE=ZLIB),
        COLUMN l ENCODING(COMPRESSTYPE=ZLIB))
   (PARTITION p1 START(1) END(10), PARTITION p2 START(10) END(20));

ALTER TABLE ccddl ADD PARTITION p3 START(20) END(30);
```

运行ALTER TABLE命令创建了TABLE ccddl的分区ccddl_1_prt_p3和ccddl_1_prt_p3_2_prt_sp1。分区ccddl_1_prt_p3与子分区sp1继承了不同的压缩设置。译者提醒：这里的子分区(sp1)其实是沿用了SUBPARTITION TEMPLATE的设置，而不是所谓的继承，在GP中，分区的存储设置不会自动从父级分区继承下来，需要手动执行，SUBPARTITION TEMPLATE与普通的继承不是一回事，望读者细细揣摩理解。你可以认为这是一个bug，当然也可以认为这是一种灵活的设计思路。

删除表

可通过DROP TABLE命令从DB中删除表。例如：

```
DROP TABLE mytable;
```

要想在不删除表定义的情况下清空表中的记录，使用DELETE或TRUNCATE命令。例如：

```
DELETE FROM mytable;
```

```
TRUNCATE mytable;
```

DROP TABLE会删掉所有与该表相关的索引、规则、触发器、限制等。然而要一起删除与该表相关的视图VIEW，必须使用CASCADE。CASCADE会删除所有依赖该TABLE的VIEW。例如：

```
DROP TABLE mytable CASCADE;
```

分区大表

表分区用以解决特别大的表的问题，比如事实表，解决办法就是将表分成很多小且更容易管理的部分。分区表在执行给定的查询语句时扫描相关的部分数据而不是全表的数据从而提高查询性能。分区表对于数据库的管理也有帮助，比如在数据仓库中滚动旧的数据。

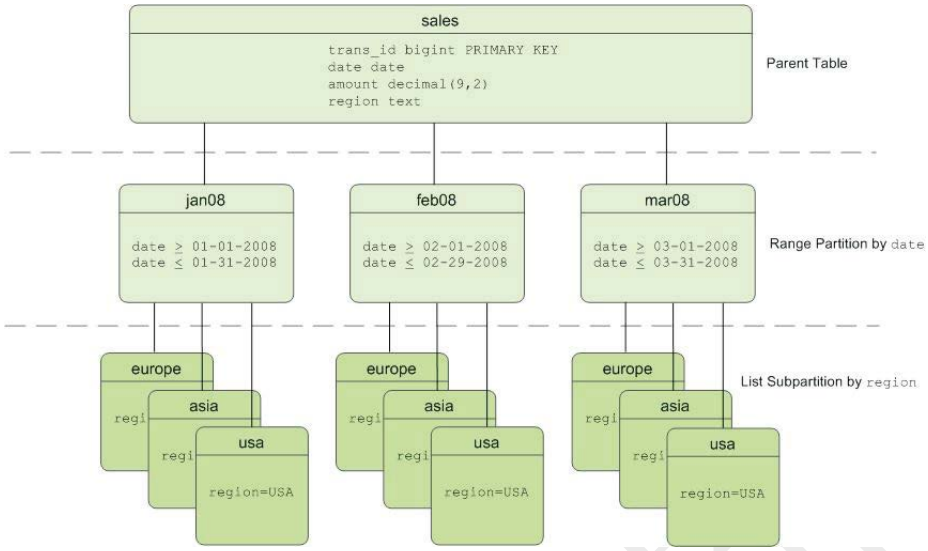
理解 GPDB 的表分区

在CREATE TABLE时使用PARTITION BY(以及可选的SUBPARTITION BY)子句来做分区。在GPDB中对一张表做分区，实际上是创建了一张顶层(父级)表和多个低层(子级)表。在内部，GPDB在顶级表与低级表之间创建了继承关系(类似于PostgreSQL中的继承/INHERIT功能)。

根据分区创建时定义的准则，每个分区在创建时都带有一个不同的检查(CHECK)约束，其限制了该表可以包含的数据。该检查约束还用于查询规划器在执行特定的查询语句时决定扫描哪些分区。

分区层级关系被储存在GP系统日志表中，因此插入到顶级父表的数据将被传到相应的分区表中。任何对分区结构的修改或者TABLE结构的修改都需要通过父表使用PARTITION子句结合ALTER TABLE命令来完成。

GPDB支持范围(根据数值型的范围分割数据，比如日期或价格)分区和列表(根据值列表分区，比如区域或生产线)分区，或者两种类型的结合。



分区表和其他非分区表一样都是在GPDB的Instance之间分布的。表在GPDB的Instance之间物理性的分布可以确保并行查询处理。表分区是一种大表逻辑切分和数据仓库任务的工具。分区本身不会改变Instance间物理上的数据分布规律。

决定表的分区策略

并不是任何TABLE都适合做分区。如果下列问题的全部或者大部分的答案是Yes，这样的表可以通过分区策略来提高查询性能。如果大部分的答案是No，分区不是好的方案：

- **表是否足够大？**大的事实表适合做表分区。若在一张表中有百万级甚至数亿条记录，从逻辑上把表分成较小的分区将可以改善性能。而对于只有数千条或者更少记录的表，对分区预先进行的管理开销将远大于可以获得的性能改善。
- **对目前的性能不满意？**作为一种调优方案，应该在查询性能低于预期时再考虑表分区。
- **查询条件是否能匹配分区条件？**检查查询语句的WHERE条件是否与考虑分区的COLUMN一致。例如，如果大部分的查询使用日期条件，那么按照月或者周的日期分区设计也许很有用，而如果查询条件更多的是使用地区条件，可以考虑使用地区将表做列表类型的分区。
- **数据仓库是否需要滚动历史数据？**历史数据的滚动需求也是分区设计的考虑因素。比如，数据仓库中仅需要保留过去两个月的数据。如果数据按照月进行分区，将可以很容易的删除掉两个月之前的数据，而最近的数据存入最近月份的分区即可。
- **按照某个规则数据是否可以被均匀的分拆？**应该选择尽量把数据均匀分拆的规则。若每个分区储存的数据量相当，那么查询性能的改善将与分区的数量相关。例如，把一张表分为10个分区，命中单个分区条件的查询扫表性能将比未分区的情况下高10倍。译者认为，我们不应该简单的说查询性能是10倍，因为多数的查询不是count(*)这样简单的计数，若是其他耗时的运算，除了扫表过滤数据这部分可以提升外，后续的处理部分是不会有性能提升的。

创建分区表

TABLE只能在执行CREATE TABLE命令时被分区。

表做分区的第一步是选择分区类型(范围分区、列表分区等)和分区字段。决定分区的层数。例如，先按照日期范围划分一级月分区，再将月分区按照区域做二级列表分区。本节通过示例演示如何创多种分区表。

- 定义日期范围分区表
- 定义数字范围分区表
- 定义列表分区表
- 定义多级分区表
- 将现有表分区

定义日期范围分区表

日期范围分区表使用单个date或者timestamp字段作为分区键。如果需要，还可以使用同样的字段做子分区(比如按照月分区后再按照日做子分区)。使用日期分区时优先考虑直接使用最细粒度的分区。比如，设置365个日分区，而不是先设置年分区再设置月分区再设置日分区。多级分区会降低查询计划的时间，但水平的分区设计可以提高执行的速度。

可以通过使用START值、END值和EVERY子句定义分区增量让GPDB自动产生分区。缺省情况下，START值总是被包含而END值总是被排除。例如：

```
CREATE TABLE sales (id int, date date, amt decimal(10,2)) DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
( START (date '2008-01-01') INCLUSIVE
      END (date '2009-01-01') EXCLUSIVE
      EVERY (INTERVAL '1 day') );
```

不过也可以为每个分区单独指定名称。比如：

```
CREATE TABLE sales (id int, date date, amt decimal(10,2))
DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
( PARTITION Jan08 START (date '2008-01-01') INCLUSIVE ,
  PARTITION Feb08 START (date '2008-02-01') INCLUSIVE ,
  PARTITION Mar08 START (date '2008-03-01') INCLUSIVE ,
  PARTITION Apr08 START (date '2008-04-01') INCLUSIVE ,
  PARTITION May08 START (date '2008-05-01') INCLUSIVE ,
  PARTITION Jun08 START (date '2008-06-01') INCLUSIVE ,
  PARTITION Jul08 START (date '2008-07-01') INCLUSIVE ,
  PARTITION Aug08 START (date '2008-08-01') INCLUSIVE ,
  PARTITION Sep08 START (date '2008-09-01') INCLUSIVE ,
  PARTITION Oct08 START (date '2008-10-01') INCLUSIVE ,
  PARTITION Nov08 START (date '2008-11-01') INCLUSIVE ,
  PARTITION Dec08 START (date '2008-12-01') INCLUSIVE END (date '2009-01-01') EXCLUSIVE );
```

注意： 由于分区的范围限制是连续的，不需要为每个分区指定END值，而只需

要为最后一个分区指定即可。但如果分区的范围不是连续的，可以考虑指定END值。

定义数字范围分区表

数字范围分区表使用单个数字列作为分区键。例如：

```
CREATE TABLE rank (id int, rank int, year int, gender char(1), count int)
DISTRIBUTED BY (id)
PARTITION BY RANGE (year)
( START (2001) END (2008) EVERY (1),
      DEFAULT PARTITION extra );
```

关于默认分区的更多信息，参见”添加默认分区”相关章节。

定义列表分区表

列表分区表可以使用任何数据类型的列作为分区键，分区规则使用等值比较。列表分区可以使用多个COLUMN(组合起来)作为分区键，而范围分区只允许使用单独COLUMN作为分区键。对于列表分区，必须为每个分区指定相应的值。例如：

```
CREATE TABLE rank (id int, rank int, year int, gender char(1), count int )
DISTRIBUTED BY (id)
PARTITION BY LIST (gender)
( PARTITION girls VALUES ('F'),
      PARTITION boys VALUES ('M'),
      DEFAULT PARTITION other );
```

关于默认分区的更多信息，参见”添加默认分区”相关章节。

定义多级分区表

当需要子分区时，可以使用多级分区的设计。使用subpartition template来确保每个分区具有相同的子分区结构，尤其是对那些后增加的分区来说。例如，创建一个两层的分区表：

```
CREATE TABLE sales (trans_id int, date date, amount decimal(9,2), region text)
DISTRIBUTED BY (trans_id)
PARTITION BY RANGE (date)
SUBPARTITION BY LIST (region)
SUBPARTITION TEMPLATE
( SUBPARTITION usa VALUES ('usa'),
  SUBPARTITION asia VALUES ('asia'),
  SUBPARTITION europe VALUES ('europe'),
  DEFAULT SUBPARTITION other_regions)
(START (date '2008-01-01') INCLUSIVE END (date '2009-01-01') EXCLUSIVE
EVERY (INTERVAL '1 month'), DEFAULT PARTITION outlying_dates );
```

下面是一个3级分区表的例子，这里表sales被分区为年、月、区域。SUBPARTITION TEMPLATE子句确保每个年分区有相同的子分区结构。另外，每个级别的分区都有一个默认分区：

```
CREATE TABLE sales (id int, year int, month int, day int, region text) DISTRIBUTED BY (id)
PARTITION BY RANGE (year)
      SUBPARTITION BY RANGE (month)
      SUBPARTITION TEMPLATE (
```

```
        START (1) END (13) EVERY (1),
        DEFAULT SUBPARTITION other_months )
    SUBPARTITION BY LIST (region)
    SUBPARTITION TEMPLATE (
        SUBPARTITION usa VALUES ('usa'),
        SUBPARTITION europe VALUES ('europe'),
        SUBPARTITION asia VALUES ('asia'),
        DEFAULT SUBPARTITION other_regions )
( START (2002) END (2010) EVERY (1), DEFAULT PARTITION outlying_years );
```

将现有表分区

对已经创建的表是不能分区的。只能在CREATE TABLE的时候做分区。要想对现有的表做分区，只能重新创建一个分区表、重新装载数据到新的分区表中、删掉旧表然后把新的分区表改为旧表的名称。还必须重新对TABLE做授权。例如：

```
CREATE TABLE sales2 (LIKE sales)
PARTITION BY RANGE (date)
( START (date '2008-01-01') INCLUSIVE END (date '2009-01-01') EXCLUSIVE
    EVERY (INTERVAL '1 month') );

INSERT INTO sales2 SELECT * FROM sales;
DROP TABLE sales;
ALTER TABLE sales2 RENAME TO sales;
GRANT ALL PRIVILEGES ON sales TO admin;
GRANT SELECT ON sales TO guest;
```

分区表的限制

主键或者唯一约束必须包含表上的所有分区键。而唯一索引可以不包含分区键，但是，其只对一个分区强制有效，而不是对整个分区表有效。

装载分区表

一旦创建了分区表，顶级表总是空的。数据值储存在最低层的表中。在多级分区表中，仅仅在层级最低的子分区中有数据。

如果某行记录无法匹配到子分区，该数据将会被拒绝并致使装载失败。若不希望在任何时候都出现记录在装载时被拒绝，可以选择定义默认分区。这样所有不能匹配分区CHECK约束的数据将装载到默认分区。可参考相关的”添加默认分区”章节。

在运行期间，查询规划器会扫描整个TABLE的层级结构并使用CHECK约束适配查询条件来决定哪些子表需要被扫描。默认分区(只要该层级中存在)总是会被扫描。如果默认分区中包含数据，其会拖慢整体的扫表时间。

在使用COPY或者INSERT向父级表装载数据时，这些数据会默认自动路由到正确的分区。因此，可以像普通的未分区表一样装载分区表。

如果有必要，还可以直接把数据装载到子表中。还可以先创建一个中间表、装载数据、然后与分区表进行分区交换。这种分区交换的性能高于直接的COPY和INSERT。参考相关的”交换分区”章节。

验证分区策略

表分区的目的是减少给定查询的数据扫描数量。若一张表基于相应的查询条件做分区，可以使用EXPLAIN查看查询计划来验证查询计划是否有选择的扫描相关的数据而不是全表扫描。

例如，假设有一张表sales已经根据日期按月做范围分区，同时按区域做了子分区，参见”定义多级分区表”章节的例子。对于下面的查询：

```
EXPLAIN SELECT * FROM sales WHERE date='01-07-08' AND region='usa';
```

对于这个查询计划应该只显示对下列表的扫描：

- 默认分区返回0-1条数据
- 2008年1月分区(sales_1_prt_1)返回0-1条数据
- USA地区子分区(sales_1_2_prt_usa)返回一些记录

下面是相关查询计划部分的示例：

```
-> Seq Scan on sales_1_prt_1 sales (cost=0.00..0.00 rows=0 width=0)
Filter: "date"=01-07-08::date AND region='USA'::text
-> Seq Scan on sales_1_2_prt_usa sales (cost=0.00..9.87 rows=20 width=40)
```

要确保查询计划没有扫描不必要的分区或者子分区，并且顶层表的扫表返回0-1条数据。

分区选择性扫描的限制

如果查询计划显示分区表没有被选择性的扫描，可能和以下的限制有关：

- 查询计划仅可以对稳定的比较运算符执行选择性扫描，如：
  = < <= > >= <>
- 查询计划不识别非稳定函数来执行选择性扫描。比如，WHERE子句中使用如date > CURRENT_DATE会使得查询计划选择性的扫描分区表，而time > TIMEOFDAY不会。

查看分区设计

要查看分区表的设计情况，通过pg_partitions视图查看。比如，查看sales表的分区情况：

```
SELECT partitionboundary, partitiontablename, partitionname, partitionlevel, partitionrank
FROM pg_partitions WHERE tablename='sales';
```

另外还有如下的视图查看分区表的信息：

- pg_partition_templates - 用以创建SUBPARTITION的SUBPARTITION template
- pg_partition_columns – 用于分区的分区键

维护分区表

必须使用ALTER TABLE命令从顶级表来维护分区。最常见的场景是根据日期范围的设计来维护数据时，删除旧分区并添加一个新的分区。还有一种可能就是把旧的分区交换为压缩AO表以节省空间。若在父表中存在默认分区，添加分区的操作只能是从默认分区拆分出一个新的分区。

- 添加新分区
- 重命名分区
- 添加默认分区
- 删除分区
- 清空分区
- 交换分区
- 拆分分区
- 修改子分区模版(Subpartition Template)

重要提示：在定义个更改分区时，使用分区名称而不是分区TABLE的名称。虽然可以使用SQL命令直接针对分区表进行查询和装载操作，但只能通过ALTER TABLE…PARTITION子句来修改分区结构。

由于分区不要求有名称，若分区没有名称，下面的表达式仍可以指定一个分区:

```
PARTITION FOR (value) or PARTITION FOR(RANK(number))
```

添加新分区

可以使用ALTER TABLE命令在已有的分区表上添加新分区。如果原有的分区表包含了subpartition template设计，新增的分区将根据该模版创建子分区。例如：

```
ALTER TABLE sales ADD PARTITION
   START (date '2009-02-01') INCLUSIVE
   END (date '2009-03-01') EXCLUSIVE;
```

如果在创建TABLE时没有subpartition template，在新增分区时需要定义子分区：

```
ALTER TABLE sales ADD PARTITION
   START (date '2009-02-01') INCLUSIVE
   END (date '2009-03-01') EXCLUSIVE
( SUBPARTITION usa VALUES ('usa'),
SUBPARTITION asia VALUES ('asia'),
SUBPARTITION europe VALUES ('europe') );
```

如果要在现有分区上添加子分区，可以指定分区执行ALTER。例如：

```
ALTER TABLE sales ALTER PARTITION FOR (RANK(12))
ADD PARTITION africa VALUES ('africa');
```

注意：若在有默认分区的分区表中添加新的分区，只能从默认分区拆分出一个新的分区。参见”拆分分区”相关章节。

重命名分区

子表的名称同样是受唯一性约束和长度限制的，GP中对象长度限制为63个字符，唯一性约束是pg_class表的唯一索引pg_class_relname_nsp_index。若名称超过长度限制，表明的一些组成部分可能会被截断(未必报错)，就无法保障唯一性了。子表的名称格式如下：

<父表名称>_<分区层级>_prt_<分区名称>

例如：

sales_1_prt_jan08

对于未指定分区名称而自动产生的范围分区表来说可能是这样的：

sales_1_prt_1

子表的名称不能通过直接执行ALTER表名来实现。但修改顶级表的名称，该改变将会影响所有相关的分区表。例如：

```
ALTER TABLE sales RENAME TO globalsales;
```

该操作将会把相关的分区表名称改为如下的格式：

globalsales_1_prt_1

只修改分区名称的操作如下所示：

```
ALTER TABLE sales RENAME PARTITION FOR ('2008-01-01') TO jan08;
```

该操作将会把相关分区表的表名改为：

sales_1_prt_jan08

在使用ALTER TABLE命令修改分区时，使用的是分区名称(如jan8)，而不是分区表的名称全称(如sales_1_prt_jan08)。对于使用COLUMN值结合FOR()指定分区的格式，GPDB是使用FOR指定的COLUMN值来匹配分区表的CHECK约束，换言之，只要FOR()的COLUMN条件在分区条件内即可匹配。

注意：ALTER TABLE语句不能操作分区表名称。比如，ALTER TABLE sales...是正确的，而ALTER TABLE sales_1_part_jan08...是不允许的。

添加缺省分区

可以使用ALTER TABLE命令为现有分区表添加默认分区：

```
ALTER TABLE sales ADD DEFAULT PARTITION other;
```

如果是多级分区表，同一层次中的每个分区都需要一个默认分区。例如：

```
ALTER TABLE sales ALTER PARTITION FOR (RANK(1)) ADD DEFAULT PARTITION other;
ALTER TABLE sales ALTER PARTITION FOR (RANK(2)) ADD DEFAULT PARTITION other;
ALTER TABLE sales ALTER PARTITION FOR (RANK(3)) ADD DEFAULT PARTITION other;
```

RANK(partitionrank)指的是范围分区同一层级中的顺序。partitionrank可参见pg_partition表。若分区表未包含默认分区，无法匹配到分区表CHECK约束的记录将被拒绝，如果有默认分区，无法匹配到分区表CHECK约束的记录将进入默认分区。

删除分区

可以使用ALTER TABLE命令删除分区表中的分区。如果被删除的分区有子分区，其所有的子分区(包括所有的数据)会一起被删除。对于范围分区的表来说，在滚动数据时，通常是从范围中删除最老的数据。例如：

```
ALTER TABLE sales DROP PARTITION FOR (RANK(1));
```

注意：在将RANK(1)的分区删除后，其余分区的partitionrank值仍然是从1开始的连续编号。编号的顺序按照分区字段的值由小到大从1开始排序。不管分区是否连续(中间有值不匹配分区)，或者随意的修改分区定义。

清空分区数据

可以使用ALTER TABLE命令来清空分区。在清空一个包含子分区的分区时，其所有相关子分区的数据都自动被清空。命令如下：

```
ALTER TABLE sales TRUNCATE PARTITION FOR (RANK(1));
```

交换分区

交换分区是用一个普通的TABLE与现有的分区交换身份。使用ALTER TABLE命令来交换分区。另外只能交换最低层级的分区表(只有包含数据的分区可以交换)。交换分区对于数据加载是有帮助的。例如，先将数据装载到一个中间表，然后与目标分区进行交换。还可以使用交换分区将旧的分区储存为AO表。例如：

```
CREATE TABLE jan08 (LIKE sales) WITH (appendonly=true);
INSERT INTO jan08 SELECT * FROM sales_1_prt_1 ;
ALTER TABLE sales EXCHANGE PARTITION FOR (DATE '2008-01-01') WITH TABLE jan08;
```

注意：该例子涉及的是单层分区表sales。译者认为能使用单层分区的情况下，最好不要选择多层分区这种复杂的结构，避免维护管理时不必要的麻烦。

拆分分区

拆分分区是将现有的一个分区分成两个分区。使用ALTER TABLE命令来拆分分区。只能拆分最低层级的分区表(只有包含数据的分区可以拆分)。指定的分割值对应的数据将进入后面一个分区(就是STAER为INCLUSIVE)。

例如，将一个月分区数据拆分到一个1-15日的分区和另一个16-31日的分区：

```
ALTER TABLE sales SPLIT PARTITION FOR ('2008-01-01')
AT ('2008-01-16')
INTO (PARTITION jan081to15, PARTITION jan0816to31);
```

如果分区表有默认分区，要添加新的分区只能从默认分区拆分。而且只能从最低层级分区的默认分区拆分(只有包含数据的分区可以拆分)。在使用INTO子句时，第2个分区名称必须是已经存在的默认分区。例如，从默认的范围分区中拆分出一个新的月份分区：

```
ALTER TABLE sales SPLIT DEFAULT PARTITION
START ('2009-01-01') INCLUSIVE
END ('2009-02-01') EXCLUSIVE
INTO (PARTITION jan09, default partition);
```

注意：如何从多级分区表的一级默认分区拆分出新的分区？译者也很纳闷，既然多级分区只能拆分最低级的默认分区，有默认分区的一级分区又不能直接添加分区，这就是说，在有一级默认分区的多级分区表新增一级分区是不可能的。当然，可以先删掉该默认分区，添加之后再恢复默认分区。译者建议能使用单层分区的情况下，最好不要选择多层分区这种复杂的结构，避免维护管理时不必要的麻烦。

修改子分区模版

使用ALTER TABLE SET SUBPARTITION TEMPLATE命令来修改现有分区表的子分区模版。在修改了子分区模版之后添加的分区，其子分区将按照新的模版产生。已经存在的分区不会被修改。例如，修改”定义多级分区表”章节中二层分区表sals的分区表模版：

```
ALTER TABLE sales SET SUBPARTITION TEMPLATE
( SUBPARTITION usa VALUES ('usa'),
    SUBPARTITION asia VALUES ('asia'),
    SUBPARTITION europe VALUES ('europe'),
    SUBPARTITION africa VALUES ('africa')
```

```
    DEFAULT SUBPARTITION other );
```

在使用新的模版后为表sales新增一个分区时，其将包含Africa地区的子分区，下面的命令将创建子分区usa、asia、europe、africa和默认分区other：

```
ALTER TABLE sales ADD PARTITION sales_prt_3
    START ('2009-03-01') INCLUSIVE END ('2009-04-01') EXCLUSIVE;
```

注意：这个例子在一级分区有默认分区时是不能执行的，要查看效果，先删除默认分区。

要删除子分区模版，使用SET SUBPart TEMPLATE并使用空的参数来完成。例如，将上面例子中的sales表的子分区模版清空：

```
ALTER TABLE sales SET SUBPARTITION TEMPLATE ()
```

创建与使用序列

序列常用于在新增记录时自动生成唯一标识符。

创建序列

使用CREATE SEQUENCE命令来创建并初始化一个给定名称的单列序列表。序列的名字必须与其他的对象如SEQUENCE、TABLE、INDEX或VIEW都不同。例如：

```
CREATE SEQUENCE myserial START 101;
```

使用序列

使用CREATE SEQUENCE命令创建好序列生成器后，就可以使用nextval函数在序列上操作了。例如，获取序列的下一个值并插入表中：

```
INSERT INTO vendors VALUES (nextval('myserial'), 'acme');
```

可以使用setval函数重置一个序列计数器的值。例如：

```
SELECT setval('myserial', 201);
```

函数nextval是不回滚的。其只要获得值就被认为已经使用该值，即便是事务在nextval之后被中断。这就意味着中断事务会使得有空缺的序列没有被真正的使用。同样的setval函数也是不回滚的。

注意，如果启用了镜像功能，nextval函数不允许在UPDATE和DELETE语句中被使用，另外currval和lastval函数目前未被GPDB支持。

检查序列当前的计数设置，可以直接查询该序列表：

```
SELECT * FROM myserial;
```

修改序列

使用ALTER SEQUENCE命令修改已有的序列表。例如：

```
ALTER SEQUENCE myserial RESTART WITH 105;
```

使用ALTER SEQUENCE并不提供任何参数，这样的操作不会修改任何设置，

换言之没有意义的操作。

删除序列

使用DROP SEQUENCE命令删除已有的序列表。例如：

```
DROP SEQUENCE myserial;
```

在 GPDB 中使用索引

在大多数的OLTP数据库中，索引可以显著的改善数据访问性能。然而在分布式数据库如GP中，应保守使用索引。GPDB在顺序扫描(索引通过随机寻址在磁盘上定位数据记录)方面很快。与传统的OLTP数据库不同的是，数据是分布在多个Instance上的。这意味着每个Instance都扫描全部数据的一小部分来查找结果。如果使用了表分区，扫描的数据可能更少。通常，商业智能(BI)的查询工作负载需要返回大量的数据，这种情况下使用索引未必有效。

GP建议在没有添加索引的情况下先测试一下查询工作负载。索引更易于改善OLTP类型的工作负载，其返回很少量的数据。在返回一定量结果的情况下，索引同样可以改善压缩AO表上查询的性能，当情况合适时查询优化器会把索引作为获取数据的选择，而不是一味的全表扫描。对于压缩数据来说，索引访问数据的方法是解压需要的记录而不是全部解压。

值得注意的是，GPDB会自动为主键建立主键索引。与文档说的不同的是，译者认为，在分区表的顶级表上建立索引会自动在其相关的子表上也建立索引，分区索引的命名规则与分区表的命名规则类似，但是，修改父级分区的索引名称不会影响到子分区的索引名称，这与表名的修改不同。

添加索引会带来一些数据库开销 – 其必定占用相当的存储空间和表更新时的索引维护资源。需确保索引的创建在查询工作负载中真正被使用到。同时，需要检查索引的确对于查询性能有显著的改善(与顺序扫描的性能相比)。可以使用EXPLAIN查看查询计划来确认是否使用了索引。参考相关”查询分析”章节。

在创建索引时需要综合考虑的问题：

- **查询工作负载。**索引更易于改善OLTP类型的工作负载，其返回很少量的数据。通常，商业智能(BI)的查询工作负载需要返回大量的数据，这种情况下使用索引未必有效。对于这种工作负载，使用顺序扫描来定位大部分数据比使用索引扫描的随机寻址定位数据更有效。
- **压缩表。**在返回一定量结果的情况下，索引同样可以改善压缩AO表上查询的性能，当情况合适时查询优化器会把索引作为获取数据的选择，而不是一味的全表扫描。对于压缩数据来说，索引访问数据的方法是解压需要的记录而不是全部解压。
- **避免在频繁更新的列上使用索引。**在频繁更新的列上创建索引，当该列被更新时，需要消耗大量的写磁盘资源和CPU计算资源。
- **创建选择性B-tree索引。**选择性指的是列中DISTINCT值的数量除以表中的记录

数。例如，如果一张表中有1000行记录且有800个DISTINCT值，选择性指数为0.8，这被认为是良好的。唯一索引总是具备1.0的选择比，这是最好的情况。值得注意的是在GPDB中唯一索引必须包含所有的DK键。

- **低选择性列上使用位图索引。**GPDB还有一种索引叫位图(Bitmap)索引，其在标准PostgreSQL中是没有的。
- **索引列用于关联。**经常关联(JOIN)的COLUMN(比如外键)上建立索引或许可以改善JOIN的性能，因为其可以帮助查询规划器使用其他的关联方法。
- **索引列经常用在查询条件中。**对于大表来说，查询语句WHERE条件中经常用到的列，可以考虑使用索引。

在GPDB中使用聚集索引

对于大表来说，使用CLUSTER命令来排序物理记录以创建索引可能需要耗费极长的时间。要快速达到同样的效果，可以通过创建一张中间表的方式来手动排序数据。例如：

```
CREATE TABLE new_table (LIKE old_table) AS SELECT * FROM old_table ORDER BY myixcolumn;
DROP old_table;
ALTER TABLE new_table RENAME TO old_table;
CREATE INDEX myixcolumn_ix ON old_table;
VACUUM ANALYZE old_table;
```

注意：译者提醒，从语法上来说，其实GPDB就是不支持CLUSTER索引，这里只是给出一种实现类似效果的方法选择。

索引类型

GPDB提供了PostgreSQL的索引类型：B-tree和GiST(Hash索引与GIN索引在GPDB中不可用)。每种索引使用不同的算法以最优化适应不同类型的查询。缺省状态下CREATE INDEX命令将创建B-tree索引，其使用于大多数场合。

注意：GPDB在使用唯一索引时有特殊考虑。唯一索引必须包含所有的DK键。唯一索引不支持AO表。在分区表上，唯一索引不能跨越子分区起到限制作用 – 唯一索引的限制仅仅在独立的分区上。

关于位图索引

除了PostgreSQL提供的索引类型之外，GPDB提供了位图(Bitmap)索引类型。位图索引对于数据仓库应用的高维数据和决策支撑系统是一个很不错的策略。这些应用通常有大量的分析查询但却很少有数据操作处理。

索引是包含指向表中记录的索引。普通的索引每个键值对应一组数据表中相同值行的ID记录。而在Bitmap索引，每个位图对应一组数据表中相同值行的ID记录。

在一张大表上建立全表的B-tree索引可能需要庞大的空间，其可能是数据空间的好几倍。而Bitmap索引通常只是数据空间零点几。

位图的每一位对应源数据的标识符，被设置的位对应的记录包含该位图相同的

值。数据的实际位置可以通过映射函数得到，因此位图索引提供了普通索引相同的功能。位图索引以压缩的方式存储位图。当DISTINCT值很小时，位图索引在压缩空间上面比B-tree表现的更好。

在WHERE子句中包含多个条件的查询可能更适合位图索引。一部分而不是全部行被匹配，条件在访问表本身之前就被过滤了。这将经常不可思议的有效改善响应时间。

何时使用位图索引

Bitmap索引在DISTINCT值数量介于100到100000之间时可以有较好的表现。DISTINCT值数量少于100的COLUMN往往不适合使用任何类型的索引。比如，性别列仅有两种DISTINCT值：男、女，不适合使用索引。一个DISTINCT值数量超过100000的列，Bitmap索引的空间个性能效率都将下降。Bitmap索引的尺寸与表中记录数和DISTINCT值数量成正比。

在特征合适的列上使用Bitmap索引将好于B-tree索引，特别是该索引列结合其他索引列一起查询的时候。

位图索引在分析查询方便可以获得显著的性能改善。WHERE条件中的AND和OR操作可以快速的转换为相应的源数据的查询。如果查询结果的数据量很小，将可以快速的响应，而不是走全表扫描。

何时不宜使用位图索引

位图索引不适合用于唯一性列和DISTINCT值很高的列，比如客户名称、电话号码。位图索引在DISTINCT值超过100000后，其性能和储存空间方面的都是都大大降低，而不管表中的记录数是多少。

位图索引主要适用数据仓库应用 -- 大量的分析查询但却很少修改数据。不适合大量并发事务更新数据的OLTP类型应用。

和B-tree相比，Bitmap索引的使用应该更保守。建议在建立Bitmap索引之后做必要的测试以证明其可以对查询性能有改善(相对于做全表扫描查询)。

创建索引

使用CREATE TABLE命令在表上定义新的索引。缺省状态下，不明确指定索引类型时创建B-tree索引。例如，在表films的title列上创建B-tree索引：

```
CREATE INDEX title_idx ON films (title);
```

在表employee的gender列上创建位图索引：

```
CREATE INDEX gender_bmp_idx ON employee USING bitmap (gender);
```

检查索引使用

虽然在GPDB中索引不需要维护和调优，但检查在真正的查询工作负载中索引是

否被使用还是很重要的。可以使用EXPLAIN命令来检查独立的查询是否使用了索引。

查询计划显示出不同的执行步骤以及时间评估等信息。在EXPLAIN的输出中寻找下面的查询节点以确认索引的使用：

- **Index Scan –** 扫描索引
- **Bitmap Heap Scan –** 从BitmapAnd、BitmapOr或BitmapIndexScan中和数据文件检索相关的记录
- **Bitmap Index Scan –** 从索引的底层扫描那些与查询条件相匹配的位图索引
- **BitmapAnd or BitmapOr –** 将来自多个位图索引通过AND或者OR结合在一起，从而产生一个新的位图。

很难通过一个通用的程序来决定那些场景要使用索引。最好的方法还是进行大量必要的测试。

- 在创建和更新索引后运行ANALYZE。该命令将收集查询规划器需要的统计信息。在预测返回结果的数量时这些信息是必要的，对于评估每个查询计划真实的开销也是必要的。
- 使用真实数据测试。用测试数据测试将可以知道对于这些测试数据来说那些索引是必要的，但这对于真实数据来说是不够的。
- 使用很小的数据量来测试是致命的错误。在从1000000条记录中查询1000条记录时，可能很适合使用索引，而从100条记录中查询1条数据时就很不适合，因为这100条数据很有可能在磁盘上存储在一个磁盘页内，查询计划是不可能决定读取一个磁盘页的什么位置的。
- 要小心制造测试数据，通常应用处于非生产环境时不可避免。数值相似，完全随机，或者顺序存储等，这些都会与真是的数据不同。
- 当索引没有被使用，强制使用可能是有用的。有些运行时的参数可以关闭一些查询计划类型。例如，关闭顺序扫描(enable_seqscan)和打开嵌套关联(enable_nestloop)将使系统走不同的查询计划。使用EXPLAIN ANALYZE命令对使用索引前后进行计时比较会很有用。

管理索引

在某些情况下，性能变差可能需要通过REINDEX命令来重建索引。重建索引将使用存储在索引表中的数据建立一个新的索引取代旧的索引。也就是说，其比删除重新建立索引的效率是要高很多的。

更新和删除操作不更新位图索引。因此在删除或者更新了位图索引列之后，可能需要使用REINDEX命令重建索引。

重建表上的全部索引

```
REINDEX my_table;
```

重建特定的索引

```
REINDEX my_index;
```

删除索引

使用DROP INDEX命令来删除特定的索引。例如：

```
DROP INDEX title_idx;
```

在装载数据时，通常先删除索引、再装载数据、然后在重新创建索引，这样比直接装载数据要快很多。译者建议使用这样的操作，译者提醒，通常来说，删除分区表的顶级分区的索引时不会自动删除相关子表的索引，要删除子分区的索引，需要逐个手动删除，使用时需慎重。

创建和管理视图

对于那些使用频繁或者比较复杂的查询，通过创建视图(VIEW)可以把其当作访问一张表一样使用SELECT语句来访问。视图不会像表一样存在于物理介质上。每当视图被访问时，创建视图的查询语句作为子查询被执行。

创建视图

使用CREATE VIEW命令将查询语句定义为一个视图。例如：

```
CREATE VIEW comedies AS SELECT * FROM films WHERE kind = 'comedy';
```

注意，目前视图忽略ORDER BY或者SORT操作，虽然在定义视图的语句中可以显式的使用ORDER BY子句，但该子句不会得到执行，除非有LIMIT子句同时出现。

删除视图

使用DROP VIEW命令删除已有的视图。例如：

```
DROP VIEW topten;
```

第十章：管理数据

本章讲述关于管理数据和GPDB中的并发访问。包含如下内容：

- 关于GPDB的并发控制
- 插入新记录
- 更新记录
- 删除记录
- 使用事务
- 回收空间

关于 GPDB 的并发控制

与事务型数据库系统通过锁机制来控制并发访问的机制不同，GPDB(与PostgreSQL一样)使用多版本控制(Multiversion Concurrency Control/MVCC)保证数据一致性。这意味着在查询数据库时，每个事务看到的只是数据的快照，其确保当前的事务不会看到其他事务在相同记录上的修改。据此为数据库的每个事务提供事务隔离。

MVCC以避免给数据库事务显式锁定的方式，最大化减少锁争用以确保多用户环境下的性能。在并发控制方面，使用MVCC而不是使用锁机制的最大优势是，MVCC对查询(读)的锁与写的锁不存在冲突，并且读与写之间从不互相阻塞。

GPDB提供了各种锁机制来控制对表数据的并发访问。大多数GPDB的SQL命令可以自动获取适当模式的锁以确保在命令执行时相关的表不会被删除或者修改。对于不能适应MVCC锁的应用来说，可以使用LOCK命令来获得适当的锁。然而，恰当的使用MVCC比使用LOCK有更好的性能表现。

锁模式	相关 SQL 命令	冲突
ACCESS SHARE	SELECT	ACCESS EXCLUSIVE
ROW SHARE	SELECT FOR UPDATE SELECT FOR SHARE	EXCLUSIVE、ACCESS EXCLUSIVE
ROW EXCLUSIVE	INSERT、COPY	SHARE、SHARE ROW EXCLUSIVE EXCLUSIVE、ACCESS EXCLUSIVE
SHARE UPDATE EXCLUSIVE	VACUUM (without FULL) ANALYZE	SHARE UPDATE EXCLUSIVE、SHARE SHARE ROW EXCLUSIVE、EXCLUSIVE ACCESS EXCLUSIVE
SHARE	CREATE INDEX	ROW EXCLUSIVE、SHARE UPDATE EXCLUSIVE SHARE ROW EXCLUSIVE、EXCLUSIVE ACCESS EXCLUSIVE
SHARE ROW EXCLUSIVE		ROW EXCLUSIVE、SHARE UPDATE EXCLUSIVE SHARE、SHARE ROW EXCLUSIVE EXCLUSIVE、ACCESS EXCLUSIVE
EXCLUSIVE	DELETE, UPDATE	ROW SHARE、ROW EXCLUSIVE SHARE UPDATE EXCLUSIVE SHARE、SHARE ROW EXCLUSIVE EXCLUSIVE、ACCESS EXCLUSIVE

ACCESS EXCLUSIVE	ALTER TABL、DROP TABLE TRUNCATE、REINDEX CLUSTER、VACUUM FULL	ACCESS SHARE、ROW SHARE、ROW EXCLUSIVE SHARE UPDATE EXCLUSIVE、SHARE SHARE ROW EXCLUSIVE EXCLUSIVE、ACCESS EXCLUSIVE

插入新纪录

在表刚被创建时，是没有数据的。在数据库进行更多利用之前的第一步是插入数据。要插入新的记录，使用INSERT命令。该命令需要表名和该表每列的值。数据的值按照列在表中出现的顺序排列，使用逗号分割。例如：

```
INSERT INTO products VALUES (1, 'Cheese', 9.99);
```

若不知道列在表中的顺序，还可以将列显式的列出来。很多用户认为总是列出列名是一种比较好的习惯。例如：

```
INSERT INTO products (name, price, product_no) VALUES ('Cheese', 9.99, 1);
```

通常，数据值使用字面值(常量)，但标量表达式也是允许使用的。例如：

```
INSERT INTO films SELECT * FROM tmp_films WHERE date_prod < '2004-05-07';
```

还可以使用一个命令插入多条记录。例如：

```
INSERT INTO products (product_no, name, price) VALUES
    (1, 'Cheese', 9.99),
    (2, 'Bread', 1.99),
    (3, 'Milk', 2.99);
```

在同时插入大量数据时，应该考虑使用外部表(CREATE EXTERNAL TABLE)或者COPY命令。在装载大量记录时，这些装载机制比使用INSERT更高效。更多关于批量装载数据的信息参见”装载和卸载数据”相关章节。

AO表为批量装载做了优化。不建议在AO表上使用单条的INSERT语句。

更新记录

UPDATE意味着对数据库中现有的数据进行修改。可以修改表中单独的记录、全部的记录、全部记录的一部分。每个列都可以被单独的更新，但不会影响其他列。

要执行更新，需要如下3方面的信息：

1. 要被更新的表和列
2. 列的新值
3. 需要被更新的列必须匹配的条件

UPDATE命令更新表中的记录。例如，下面的命令更新products表中所有price为5的记录的price为10：

```
UPDATE products SET price = 10 WHERE price = 5;
```

在GPDB中使用UPDATE有如下的限制：

- GP的DK不可以被UPDATE
- 在GPDB中不支持RETURNING子句(参考相关SQL说明)

删除记录

使用DELETE命令从指定的表中删除符合WHERE条件的记录。如果没有使用WHERE子句，将会删除该表的所有记录。例如，从products表中删除所有price为10的记录：

```
DELETE FROM products WHERE price = 10;
```

或者删除表中所有记录：

```
DELETE FROM products;
```

在GPDB中使用DELETE操作的限制：

- 在GPDB中不支持RETURNING子句(参考相关SQL说明)

清空表

若想要快速删除所有记录，应该考虑使用TRUNCATE命令。例如：

```
TRUNCATE mytable;
```

该命令一次清空表中的全部记录。值得注意的是，TRUNCATE不扫描表，其继承者表不会被执行该操作，只是被TRUNCATE的表受到影响。分区表视作一个整体，在父级表上执行TRUNCATE操作会清空所有相关子表的数据。

使用事务

事务允许将多个SQL语句放在一起当作一个整体操作，所有SQL一起成功或失败。

在GPDB中用以执行事务的SQL命令为：

- 使用BEGIN或START TRANSACTION开始一个事务块
- 使用END或COMMIT提交事务块
- 使用ROLLBACK回滚事务而不提交任何修改
- 使用SAVEPOINT选择性的保存事务点，之后可以使用ROLLBACK TO SAVEPOINT回滚到之前保存的事务点，还可以使用RELEASE SAVEPOINT来释放之前保存的事务点，更多信息参考相关SQL说明，译者认为在GP中应用场景不多。

事务隔离级别

SQL标准定义了4个事务隔离级别。在GPDB中可以使用4个事务隔离级别的任何一个。但在内部，实际上只存在两个不同的隔离级别 – 已提交读(read commited)和序列化(serializable)：

- **已提交读** – 当事务使用该隔离级别，SELECT查询只能看到查询开始前的数据，其永远读不到SELECT查询期间其他并发事务未提交或已提交的修改。不过，SELECT语句可以看到当前事务之前所做的改变，虽然这些改变尚未提交。实际上SELECT语句看到的是该查询开始时数据库的一个镜像。需要注意的是，如果在一个事务中有两个SELECT语句，且在第一个SELECT语句期间有其他的事务提交了对数据的改变，两个SELECT语句可

以看到不同的数据。UPDATE和DELETE命令与SELECT在数据目标上有同样的行为，他们也只会找到命令开始前已经提交的数据。不过，在获取这些目标数据之前其可能已经被其他的事务更新(或者删除或者锁定)。已提交读事务隔离级别对于多数的应用已经足够，且该级别高效而易用。然而对于复杂的查询更新应用来说，获取比已提交读隔离级别更高的数据一致性保护还是很有必要的。

- **系列化** – 这是最严格的事务隔离级别。该级别要求事务被串行执行，就是说事务必须一个接着一个的执行而不能并发执行。应用程序在使用该级别时需要做好由于序列化失败而需要的重试操作。当一个序列化事务在执行时，SELECT查询只能看到该事务开始前提交的数据，其永远读不到该事务期间其他并发事务未提交或已提交的修改。不过，SELECT语句可以看到当前事务之前提交的改变，虽然这些改变尚未提交。同一事务中连续的SELECT命令始终看到相同的数据。UPDATE和DELETE命令与SELECT在数据目标上有同样的行为，他们也只会找到事务开始前已经提交的数据。不过，在获取这些目标数据之前其可能已经被其他的事务更新(或者删除或者锁定)。在这种情况下，系列化事务将会等待前面更新事务的提交或者回滚(若该事务仍在执行)。如果前面的更新事务回滚了，其影响被丢弃，序列化事务将处理更新之前的数据。而如果之前的更新事务提交了(发生了真实的数据更新删除而非仅仅的获取锁)，序列化事务将会被回滚。
- **未提交读** – 在GPDB中与已提交读等同。
- **可重复读** – 在GPDB中与序列化等同。

在GPDB中缺省的事务隔离级别是已提交读。要使用不同的事务隔离级别，可以在BEGIN事务时声明隔离界别，或者在事务开始之后使用SET TRANSACTION命令设置隔离级别。详见相关附录或手册。

回收空间

由于MVCC事务并发模型的原因，已经删除或者更新的记录仍然占据着磁盘空间，虽然其对于新的事务来说已经不可见。如果数据库有大量的更新和删除操作，其将会产生大量的过期记录。定期的运行VACUUM命令可以删除这些过期的记录。例如：

```
VACUUM mytable;
```

VACUUM命令还会收集表级别的统计信息，如记录数、占用磁盘页面数，所以在装载数据之后对全表执行VACUUM是有必要的，这同样适用AO表。关于VACUUM操作可以参考相关的”日常回收与分析”章节。译者认为该命令还需慎用，就译者的经验和理解来说，VACUUM未必比REORGANIZE好，ANALYZE未必需要针对全表。

配置子空间映射

过期的记录会被存在叫做自由空间映射的地方。自由空间映射的大小必须足够容纳数据库中的所有过期记录。如果尺寸不够大，超出自由映射空间的过期记录占用的空间将无法被VACUUM命令回收。

VACUUM FULL命令将回收所有过期记录，但这是一个很昂贵的操作并且其可能会花费无法接受的时间长度在一张大表上来完成操作。如果自由映射空间已经溢出，最好的做法是及时的使用CREATE TABLE AS命令来重建数据表并删除旧的表。在GPDB中不建议使用VACUUM FULL命令。

最好将自由映射空间设置为一个合适的值。自由空间映射由下面的参数来设置：

max_fsm_pages

max_fsm_relations

相关信息参考相关附录章节。

第十一章：查询数据

本章讲述在GPDB中使用SQL语言。通常可以使用标准的PostgreSQL交互命令psql来执行SQL命令，但也有其他类似功能的应用。

- 定义查询
- 使用函数和运算符
- 查询分析

定义查询

查询是一个查看、修改或者分析数据库中数据的命令。本节介绍如何在GPDB中构造SQL查询。

- SQL词典
- SQL值表达式

SQL 字典

SQL(结构化查询语言)是用来访问数据库的一种语言。SQL语言有特定的字典(单词、特征等)，据此构造数据库引擎可以理解的查询或命令。

SQL由一些列的命令组成。命令由一系列的符号组成并以分号(;)结尾。符号的有效性取决于特定的命令。语法规则参见相关的”SQL命令参考”附录。

GPDB基于PostgreSQL并遵守相同的SQL结构和语法(一些次要的例外)。大多情况下语法与PostgreSQL对等，不过在GPDB中有些命令可能会有增量或者语法限制。

关于PostgreSQL中SQL规则和概念的完整解释，参考相关”PostgreSQL SQL语法”文档。

SQL 值表达式

在多种情况下都用到值表达式，比如SELECT命令的目标列表、INSERT或UPDATE命令新的列值、查询条件中的各种命令。值表达式的结果通常被称为标量，以区别表的表达式结果(其标识一张表)。因此值表达式被成为标量表达式(或直白表达式)。表达式语法允许从原始值通过算数、逻辑、集合等操作计算出结果值。

一个值表达式是下列请款之一：

- 常数或字符值
- 列的引用
- 位置参数引用 – 比如在函数定义中
- 下标表达式
- 字段选择表达式
- 运算符调用

- 函数调用
- 聚合表达式
- 窗口表达式
- 类型转换
- 标量子查询
- 数组构造函数
- 行构造函数
- 括号中的值表达式 – 用于分组子表达式或设置优先级

在这之外，还有一些结构被当作表达式，但其不符合常规的语法规则。这些通常相当于函数或运算符，并在”使用函数和运算符”章节解释。

列的引用

列可以按照这种方式被引用：

```
correlation.columnname
```

这里correlation(相关名)可以是一张表(有时还需要模式名)，或者一个FROM子句中对表定义的别名，或者NEW、OLD关键字。(NEW和OLD只能在重写规则中出现，而其他相关名可以出现在任何的SQL语句。)当列名在当前查询的所有表中唯一时相关名和点号可以省略。

位置参数引用

位置参数引用用于标识从外部给一个 SQL 语句的一个参数。 参数用于 SQL 函数定义语句。在一些客户端库中还可以在SQL命令之外独立的指定数据值，这时的参数引用指的是命令之外的数据值(类似编程中常见的占位符)。参数引用的格式为：

```
$number
```

例如，dept函数的定义：

```
CREATE FUNCTION dept(text) RETURNS dept
AS $$
SELECT * FROM dept WHERE name = $1
$$
LANGUAGE sql;
```

这里，在函数被调用时$1就是函数的第一个参数值的引用。

下表表达式

若一个表达式产生了一个数据类型的值，那么该数组中特定元素的值可以这样被获取：

```
expression[subscript]
```

或者数据的多个相邻元素这样被获取：

```
expression[lower_subscript:upper_subscript]
```

这里，方括号中指定的是字面标量。每个下标都是表达式自身，其必须是自然数。通常数组表达式必须有括号，但当要被下标的对象仅有一个列或者定位参数时，括号可以被省略。例如：

```
mytable.arraycolumn[4]
mytable.two_d_column[17][34]
$1[10:42]
(arrayfunction(a,b))[42]
```

这里的括号都是必须的。

字段选择表达式

若一个表达式产生了一个符合类型(ROW类型)的值，那么其特定的属性可以这样被获取：

expression.fieldname

通常ROW表达式必须必须被括起来，但如果该表达式如果是一张表的引用或者位置参数引用，括号可以被省略。例如：

mytable.mycolumn

$1.somecolumn

(rowfunction(a,b)).col3

因此，列引用实际上就是一个特殊的字段选择语法。

运算符调用

运算符调用有3中可能的语法：

expression operator expression(二元中间运算符)

operator expression(一元前缀运算符)

expression operator(一元后缀运算符)

这里运算符是一个运算标记，可以是关键字AND、OR或NOT等，或者是如下的运算符名称：

OPERATOR(schema.operatorname)

存在哪些特定的运算符以及他们是一元还是二元取决于系统或用户的定义。参见”内置函数和运算符”相关章节了解更多内置运算符描述。

函数调用

调用函数的语法是函数名(有时还需要模式名)后跟着相关的参数，这些参数用括号括起来。

function ([expression [, expression ...]])

例如，下面的函数调用计算2平方根：

sqrt(2)

内置函数列表参见”内置函数和运算符”相关章节。其他函数可能是用户自己创建的。

聚合表达式

聚合表达式是将聚合函数作用在一些查询选出的行上面的处理。聚合函数将多个输入处理为一个输出值，比如求和、平均值。聚合表达式的语法为：

aggregate_name (expression [, ...]) [FILTER (WHERE condition)]

aggregate_name (ALL expression [, ...]) [FILTER (WHERE condition)]

aggregate_name (DISTINCT expression [, ...]) [FILTER (WHERE condition)]

aggregate_name (*) [FILTER (WHERE condition)]

这里的聚合名称(aggregate_name)是事先定义好的聚合函数(优势需要模式名)，而表达式(expression)是本身不是聚合表达式的任何值表达式。

第一种语法的聚合表达式将把所有输入的数据计算出一个非空值。第二种语法与第一种相同，因为ALL是缺省的。第三种语法是对该表达式找到的所有DISTINCT非空值调用聚集。最后一种语法为每一行调用聚集，而不管其为空还是非空，因为没有指定特定输入值，它通常用于count(*)这个聚集函数。

例如，count(*)计算出总记录数，count(f1)计算出非空记录数，count(distinct f1)

计算出DISTINCT非空记录数。这里f1为一个列明。

这里的FILTER子句允许用来为聚合函数指定特定的条件以过滤输入的记录。例如：

```
SELECT count(*) FILTER (WHERE gender='F') FROM employee;
```

这里FILTER子句的WHERE条件不能包含返回集合的函数、子查询、窗口函数或者外部引用。如果使用自定义的聚合函数，该函数的状态必须声明为严格模式(STRICT)(参见CREATE AGGREGATE)。

预定义的聚合函数信息参见”聚合函数”章节。其他聚合函数可能是用户自己定义的。

GPDB为逆分布函数提供了特殊的聚合表达式：

```
PERCENTILE_CONT(_percentage_) WITHIN GROUP (ORDER BY _expression_)
PERCENTILE_DISC(_percentage_) WITHIN GROUP (ORDER BY _expression_)
```

GP中仅有这两个表达式使用WITHIN GROUP关键字。GPDB还提供了MEDIAN聚合函数。该函数返回PERCENTILE_CONT结果的百分之50。

聚合表达式的限制

以下是一些聚合表达式的限制：

- 一些高级的聚集函数不能和ALL、DISTINCT、FILTER或OVER等关键字一起使用。参见”高级聚集函数”相关章节。
- 一些聚合表达式不能与分组规范一起使用：CUBE、ROLLUP和GROUPING SETS。
- 有些聚合函数值出现在结果列表或者HAVING子句值出现在SELECT命令中。这些是不允许出现在其他子句中的，比如WHERE，因为这些子句的评估逻辑在得到聚合结果之前。
- 当一个聚合表达式出现在子查询中，通常该聚合用于评估记录数量。但如果该聚集参数包含了外部的引用会出现异常。该子查询的聚合表达式的结果以一个常量的形势出现在外部相关的结果中。GPDB目前不支持多个输入表达式的DISTINCT。

窗口表达式

窗口函数允许应用开发人员使用标准SQL命令方便的构造复杂的在线分析处理(OLAP)查询。例如，计算不同时间段的平均值和总数，根据列值的变化得到聚合值和等级关系，简单的得到复杂的比率关系。

窗口表达式是在窗口架构(OVER子句)的基础上应用窗口函数。窗口分区是将一个数据集合分区为一个整体用窗口函数来处理。与聚集函数不同，聚集函数为每组记录返回一个结果，而窗口函数为每条记录返回一个结果，但那些计算只是在一个窗口分区数据内部。如果没有指定分区，窗口函数将在全部的中间结果集上进行计算。

窗口表达式的语法为：

```
window_function ( [expression [, ...]] ) OVER ( window_specification )
```

这里的window_function是”窗口函数”列表中的函数，expression是任何自身不包

含窗口表达式和窗口规范的值表达式：

```
[window_name]
[PARTITION BY expression [, ...]]
[[ORDER BY expression [ASC | DESC | USING operator] [, ...]
    [{RANGE | ROWS}
      { UNBOUNDED PRECEDING
      | expression PRECEDING
      | CURRENT ROW
      | BETWEEN window_frame_bound AND window_frame_bound }]]
```

这里的window_frame_bound可以是下面的一种：

```
UNBOUNDED PRECEDING
expression PRECEDING
CURRENT ROW
expression FOLLOWING
UNBOUNDED FOLLOWING
```

窗口表达式可能值出现在SELECT命令的列表中。例如：

```
SELECT count(*) OVER(PARTITION BY customer_id), * FROM sales;
```

OVER子句是窗口函数与其他聚集函数或报表函数的区别。OVER子句定义了窗口规范，其决定函数应用在什么窗口上。一个窗口规范包含如下特征：

- PARTITION BY子句，其决定了窗口函数应用在那个什么窗口上。如果缺失，整个结果集将被视作一个窗口。
- ORDER BY子句定义了在一个窗口内如何排序记录。值得注意的是，窗口规范中的ORDER BY与标准查询表达式中的ORDER BY是不一样的。对于计算等级的窗口函数来说ORDER BY子句是必须的，其可以得到等级值。对于OLAP聚合来说，在使用窗口框架(ROWS或RANGE子句)时必须有ORDER BY子句。

  注意：缺乏连贯排序的列，比如时间，不适合在窗口规则中的ORDER BY子句中使用。时间类型缺乏连贯排序的原因是加减法不能得到预期的效果。例如，下面的表达式并不总是为true：

  ```
  x::time < x::time + '2 hour'::interval
  ```

- ROWS/RANGE子句用以定义窗口框架。窗口框架定义了一个窗口分区内的集合。当窗口框架定以后，窗口函数将基于动态的框架计算而不是整个窗口部分进行计算。窗口框架可以是行导向(ROWS)或者值导向(RANGE)。

类型转换

类型转换指的是从一种数据类型转换为另外一种。GPDB(与PostgreSQL相同)使用两种等价的方式做类型转换：

```
CAST ( expression AS type )
expression::type
```

CAST语法与SQL标准一致，而::语法源于PostgreSQL的历史用法。

在CAST应用于一个已知类型的值表达式时，其表示运行时的类型转换。只当合适的类型转换函数已经被定义时转换才会成功。注意这与常量的转换略有差异。应用于普通字符常量的类型转换意味着其作为该类型的字符常量值，并且对于任何类型总是成功的(当然假设字符串的内容对于该类型来说可以接受)。

在一个值表达式中，如果其类型没有奇异的话，可以省去明确的类型转换(比如被指派到表的列)，此时系统会自动应用类型转换。然而，自动转换仅适用于那些在系统日志中标记为”可隐式使用”的类型转换。其他的转换必须使用显式的转换语法。该限制是防止发生意外的隐式转换。

标量子查询

标量子查询是一个在括号中的普通SELECT查询，其返回单行单列的结果。该SELECT查询被执行，其返回的但值用于周围的表达式。作为一个标量子查询，若其返回所行或者多列是错误的。如果标量子查询包含外部语句块的引用，该子查询称为关联标量子查询。

关联子查询

关联子查询提供了一种使用其他查询结果来组建结果的方法。GPDB支持关联子查询(Correlated Subqueries/CSQs)，其为很多已有的应用提供了兼容性。关联子查询是一个普通的SELECT查询，其WHERE子句或目标列表包含了外部子句的引用。关联子查询可以是标量子查询(例1)，也可以是表子查询(例2)，区别在于其返回的是单条记录还是多条记录。关联子查询不支持跨级的关联(比如包含外部的外部的子句的引用)。

关联子查询例子

例1 – 标量关联子查询

```
SELECT * FROM t1 WHERE t1.x > (SELECT MAX(t2.x) FROM t2 WHERE t2.y = t1.y);
```

例子2 – 是否存在的关联子查询

```
SELECT * FROM t1 WHERE EXISTS (SELECT 1 FROM t2 WHERE t2.x = t1.x);
```

GPDB执行关联子查询有两种方式，如下：

1. 关联子查询可以被拆解为关联(JOIN)操作，这是高效的
2. 关联子查询为外部的查询每行执行一次

大部分的查询属于第一种，包括所有TPC-H基准测试的查询。

在SELECT列表中或者使用OR连接的子句中出现的关联子查询属于第二种类型。例3和例4阐述一些这样的查询如何重写以改进性能。

例子3 – 在SELECT列表中的关联子查询

源语句：

```
SELECT T1.a,
    (SELECT COUNT(DISTINCT T2.z) FROM t2 WHERE t1.x = t2.y) dt2
FROM t1;
```

重写该查询，先与t1进行内链接，再与t1进行左连接。该重写仅是作用在等价关联条件上。

重写后语句：

```
SELECT t1.a, dt2 FROM t1
    LEFT JOIN
    (SELECT t2.y AS csq_y, COUNT(DISTINCT t2.z) AS dt2
        FROM t1, t2 WHERE t1.x = t2.y GROUP BY t1.x)
    ON (t1.x = csq_y);
```

例子4 – 使用OR连接的子句中的关联子查询

源语句：

```
SELECT * FROM t1
WHERE
x > (SELECT COUNT(*) FROM t2 WHERE t1.x = t2.x)
OR x < (SELECT COUNT(*) FROM t3 WHERE t1.y = t3.y)
```

将该查询分为两个部分并UNION OR条件。

重写后语句：

```
SELECT * FROM t1
WHERE x > (SELECT count(*) FROM t2 WHERE t1.x = t2.x)
UNION
SELECT * FROM t1
WHERE x < (SELECT count(*) FROM t3 WHERE t1.y = t3.y)
```

要确定一个关联子查询属于哪一种，使用EXPLAIN SELECT或者EXPLAIN ANALYZE SELECT语句查看查询计划。存在SubPlan节点的查询计划说明该查询为第二种类型。更多信息参考相关”查询分析”章节。

高级表函数

GPDB通过TABLE值表达式支持表函数。作为高级表函数输入值可以使用ORDER BY子句，并且可使用SCATTER BY子句执行列来分布数据。这与在创建表时使用DISTRIBUTED BY子句是相似的。然而重分布只在查询被执行是才发生。

下面的命令使用了SCATTER BY子句：

```
SELECT * FROM f( TABLE(SELECT * FROM input_table ORDER BY x SCATTER BY y) );
```

注意：表函数的参数表在集群的节点之间自动重分布数据。

数组构造函数

数据构造函数是一个从一系列值构造一个数组作为自身元素的表达式。简单的数组构造函数有关键字ARRAY、左方括号、数个(0或N)逗号分割的表达式、右方括号组成。例如：

```
SELECT ARRAY[1,2,3+4];
array
---------
{1,2,7}
```

数据元素的类型是成员表达式的基本类型，其与UNION或CASE使用相同的规则。

多维数据可以通过嵌套数据构造函数的方式构造。在构造函数内部，ARRAY关键字可以被省略。例如，这两个SELECT语句产生相同的结果：

```
SELECT ARRAY[ARRAY[1,2], ARRAY[3,4]];
SELECT ARRAY[[1,2],[3,4]];
array
---------------
{{1,2},{3,4}}
```

由于多维数据必须是规矩的，内部同意层级上的构造函数必须生成完全相同维度的子数组。

多维数据的构造函数可以是任何生成合适类型数据的东西，而不仅仅是子数据构造器。例如：

```
CREATE TABLE arr(f1 int[], f2 int[]);
INSERT INTO arr VALUES (ARRAY[[1,2],[3,4]], ARRAY[[5,6],[7,8]]);
SELECT ARRAY[f1, f2, '{{9,10},{11,12}}'::int[]] FROM arr;
array
------------------------------------------------
{{{1,2},{3,4}},{{5,6},{7,8}},{{9,10},{11,12}}}
```

还可以从一个子查询的结果集来构建一个数据。这种情况下，数组构造函数被写为关键字ARRAY和一个括号括起来的子查询。例如：

```
SELECT ARRAY(SELECT oid FROM pg_proc WHERE proname LIKE 'bytea%');
?column?
------------------------------------------------------------
{2011,1954,1948,1952,1951,1244,1950,2005,1949,1953,2006,31}
```

子查询必须返回单列的结果集。生成的单维数组将子查询得到的每行作为一个元素，元素的类型与子查询输出的列的类型一致。使用ARRAY建立数据的下标值始终从1开始。

行构造函数

行构造函数是一个从一系列的值构建一个作为自身行的值的表达式。行构造函数由关键字ROW、左括号、一系列(0或N)逗号分割的表达式、右括号组成。例如：

```
SELECT ROW(1,2.5,'this is a test');
```

行构造函数可以包含rowvalue.*语法，其将会被把该行的值展开为一个元素列表，就像在顶层的SELECT列表中的语法那样。例如，如果表t有列f1和f2，下面的语句是对等的：

```
SELECT ROW(t.*, 42) FROM t;
SELECT ROW(t.f1, t.f2, 42) FROM t;
```

缺省状态下，使用ROW表达式创建的值是匿名记录类型。如果有必要，其可以转换为命名的复合类型 – 或者一张表的行类型，或者一个通过CREATE TYPE AS创建的复合类型。有时为避免歧义可能需要明确的转换。例如：

```
CREATE TABLE mytable(f1 int, f2 float, f3 text);
CREATE FUNCTION getf1(mytable) RETURNS int AS 'SELECT $1.f1' LANGUAGE SQL;
--由于只有一个getf1()存在，所以不必转换：
SELECT getf1(ROW(1,2.5,'this is a test'));
getf1
-------
1
CREATE TYPE myrowtype AS (f1 int, f2 text, f3 numeric);
CREATE FUNCTION getf1(myrowtype) RETURNS int AS 'SELECT $1.f1' LANGUAGE SQL;
--现在需要转换以指定执行哪个函数：
SELECT getf1(ROW(1,2.5,'this is a test'));
ERROR: function getf1(record) is not unique
```

```
SELECT getf1(ROW(1,2.5,'this is a test')::mytable);
getf1
-------
1
SELECT getf1(CAST(ROW(11,'this is a test',2.5) AS myrowtype));
getf1
-------
11
```

行构造函数可在存储复合类型表(有复合类列类型)数据时用于构建复合类型数据，或用于给一个具有复合类型参数的函数传参。

表达式评估规则

表达式的评估顺序没有定义。通常，一个运算符或者函数的输入没必要按照固定的顺序从左到右评估。此外，如果一个表达式的结果可以通过只评估其一部分来确定，其它部分的自表达式可能完全没必要被评估。比如，一种写法为：

```
SELECT true OR somefunc();
```

函数somefunc()可能根本不会被调用。而另外一种写法为：

```
SELECT somefunc() OR true;
```

注意，这与一些程序语言布尔操作的从左到右”强制评估顺序”是不同的。

因此，在复杂的表达式中利用函数的副作用是极不明知的。而且在WHERE或者HAVING子句中依赖副作用或者评估顺序是特别危险的，因为这些子句很可能成为查询计划重新加工的部分。布尔表达式(AND/OR/NOT 或组合)在这些子句中可能会被以任何符合逻辑规则的方式重新组织。

如果一定要强制评估的顺序，可以选择CASE结构。例如，这样在WHERE子句中避免被0除是靠不住的：

```
SELECT ... WHERE x <> 0 AND y/x > 1.5;
```

但这样做是安全的：

```
SELECT ... WHERE CASE WHEN x <> 0 THEN y/x > 1.5 ELSE false END;
```

CASE结构会使得优化器不再尝试重新组织它，因此请在必要的时候这么做。

使用函数和运算符

- 在GPDB中使用函数
- 使用自定义函数
- 内置函数和运算符
- 窗口函数
- 高级分析函数

在 GPDB 中使用函数

函数有三种类型：不变型(IMMUTABLE)、稳定型(STABLE)、不稳定型(VOLATILE)。GPDB完全支持所有类型的不变型函数。不变型函数仅仅依赖直接传递的参数列表，而且在给定参数值的情况下总是得到相同的返回值。

GPDB支持大部分的稳定性函数。稳定型表明，在单个表上的扫描，使用相同的参数值函数将始终返回相同的结果，但该结果可能会因其他的SQL语句而发生改变。返回值依赖于数据库查询或者参数值的函数归类为稳定型。值得注意的是，current_timestamp一类的函数可作为稳定型，因为在一个事务中它们的值是不变的。

在GPDB中使用不稳定型函数是受限的。不稳定型表明，即便是单表的扫描，函数值也可能发生变化。相对来说，很少有数据库函数属于这种类型，一些例子如random()、currval()、timeofday()等。不过需要提醒的是，所有有副作用的函数都必须是不稳定型，即便其返回值是可预测的(比如setval())。

在GPDB中，数据分散存储在各Segment Instance上 – 每个Segment Instance是一个独立的PostgreSQL数据库。为了防止节点之间的数据出现不一致，任何含有SQL语句或者修改数据库的不稳定函数都不可以在Segment Instance级别执行。比如，函数random()或者timeofday()不允许在GPDB的分布式数据上执行，因为其可能会导致Segment Instance之间的数据不一致。

为了确保数据的一致性，不稳定型和稳定型函数可以安全的在语句中使用，因为该语句是在Master上被评估和执行。例如，下面的语句总是在Master上被执行(没有FROM子句的语句)：

```
SELECT setval('myseq', 201);
SELECT foo();
```

有时候，含有FROM子句的语句中含有分布式表，并且函数在FROM子句中被简单的用于返回记录集，这种情况可能是允许在Segment上执行的：

```
SELECT * from foo();
```

这种规则的一个例外是，函数返回一个表的引用，或者使用了refCursor数据类型。这些类型的函数不能使用在GPDB中(译者不明白这句话的含义，译者认为应该是指在pgsql中不能使用)。

自定义函数

GPDB像PostgreSQL一样支持自定义函数的使用。更过信息可以参看PostgreSQL文档的”扩展SQL”章节。可以使用CREATE FUNCTION命令注册自定义函数，就像”在GPDB中使用函数”章节描述的那样。缺省状态下，函数被声明为非稳定型，因此，如果自定义函数是不变型或者稳定型的，在注册函数时指定其稳定性是很重要的。

在创建自定义函数时，应避免使用致命错误或任何类型的破坏性调用。GPDB对于此类错误的响应可能会是突然关闭或重启。

注意，在GPDB中，自定义函数的共享库文件在每个GPDB主机(Master、Segment以及Mirror)上的库路径必须相同。

内置函数和运算符

下标列出了PostgreSQL支持的内置函数和运算符的类别。除了稳定型和非稳定型函数外，所有PostgreSQL的函数和运算符在GPDB中都支持，所其受限如”在GPDB中使用函数”中的描述。

关于内置函数和运算符的更多信息参照PostgreSQL文档的”内置函数和运算符”相关章节。

运算符/函数种类	不稳定函数	稳定函数
逻辑运算符		
比较运算符		
数学函数和运算符	random、setseed	
字符串函数和运算符	所有内置转换函数	convert、pg_client_encoding
二元字符串函数和运算符		
按位字符串函数和运算符		
模式匹配		
日期类型格式化函数		to_char、to_timestamp
日期/时间函数和运算符	timeofday	age、current_date、current_time current_timestamp、localtime localtimestamp、now
几何函数和运算符		
网络地址函数和运算符		
序列处理函数	currval、lastval、nextval、setval	
条件表达式		
数组函数和运算符		所有数据函数
聚合函数		
子查询表达式		
行比较和数组比较		
集合返回函数	generate_series	
系统信息函数		所有会话信息函数 所有访问权限查询函数 所有模式可见性查询函数 所有系统日志信息函数 所有说明信息函数
系统管理函数	set_config、pg_cancel_backend pg_reload_conf、pg_rotate_logfile pg_start_backup、pg_stop_backup pg_size_pretty、pg_ls_dir pg_read_file、pg_stat_file	current_setting 所有数据库对象尺寸函数
XML 函数		xmlagg(xml) xmlexists(text, xml) xml_is_well_formed(text) xml_is_well_formed_document(t

		ext) xml_is_well_formed_content(text) xpath(text, xml) xpath(text, xml, text[]) xpath_exists(text, xml) xpath_exists(text, xml, text[]) xml(text) text(xml) xmlcomment(xml) xmlconcat2(xml, xml)

窗口函数

GPDB有如下这些在PostgreSQL8.2中没有的内置窗口函数。窗口函数常用以构建复杂的OLAP(在线分析处理)查询。窗口函数在一个查询表达式中对分区结果集进行分区处理。一个窗口分区是一个查询返回结果集的子集，其由特殊的子句OVER()来定义。参见”窗口表达式”相关章节。所有的窗口函数都是不变型函数。

值得注意的是，任何一个聚合函数(如PostgreSQL的”聚合函数”相关章节的描述)都可以与OVER子句一起使用，从而使得其成为窗口聚合函数。窗口聚合函数为特定的窗口框架返回特定的值。如果OVER子句未指定排序(ORDER BY)或者窗口框架(ROWS或RANGE)，聚合值将基于整个窗口分区(窗口框架是窗口分区的条件子集)。在不指定排序和框架的情况下，如果聚合函数允许指定DISTINCT，其相当于窗口聚合函数。

函数	返回值类型	完整语法	描述
cume_dist()	double precision	CUME_DIST() OVER ([PARTITION BY expr] ORDER BY expr)	计算一组值的累计分布，相同值的记录总是统计在相同的累计分布中
dense_rank()	bigint	DENSE_RANK () OVER ([PARTITION BY expr] ORDER BY expr)	在分区中排序计算等级而不跳过等级的值。相同值的记录等级相同
first_value(expr)	与输入表达式类型相同	FIRST_VALUE(expr) OVER ([PARTITION BY expr] ORDER BY expr [ROWS|RANGE frame_expr])	返回一定顺序集合的第一个值
lag(expr [,offset] [,default])	与输入表达式类型相同	LAG(expr [,offset] [,default]) OVER ([PARTITION BY expr] ORDER BY expr)	一种跨行访问方式。按照排序的位置，LAG 将一行记录向后偏移。如果不指定 offset 缺省值为 1。如果不指定 default 缺省值为 null
last_value(expr)	与输入表达式类型相同	LAST_VALUE(expr) OVER ([PARTITION BY expr] ORDER BY expr [ROWS|RANGE frame_expr])	返回一定顺序集合的最后一个值

lead(expr [,offset] [,default])	与输入表达式类型相同	LEAD(expr [,offset] [,default]) OVER ([PARTITION BY expr] ORDER BY expr)	一种跨行访问方式。lead 将一行记录向前偏移。如果不指定 offset 缺省值为 1.如果不指定 default 缺省值为 null
ntile(expr)	bigint	NTILE(expr) OVER ([PARTITION BY expr] ORDER BY expr)	将排序的数据集合分为 expr 个小块，并在每块内标记顺序
percent_rank()	double precision	PERCENT_RANK () OVER ([PARTITION BY expr] ORDER BY expr)	计算记录的百分比等级。返回值大于 0 并小于或等于 1
rank()	bigint	RANK () OVER ([PARTITION BY expr] ORDER BY expr)	在分区中排序计算等级且可能跳过等级的值。相同值的记录等级相同且下一等级值会被跳过
row_number ()	bigint	ROW_NUMBER () OVER ([PARTITION BY expr] ORDER BY expr)	为排序的分区记录每行分配一个唯一的编号

高级分析函数

GPDB提供了如下这些在PostgreSQL中没有的内置高级分析函数。这些分析函数都是不变型函数。

函数	返回值类型	完整语法	描述
matrix_add(array[],array[])	smallint[]、int[] bigint[]、float[]	matrix_add(array[[1,1],[2,2]], array[[3,4],[5,6]])	将两个 2 维矩阵相加。2 个矩阵必须相一致
matrix_multiply(array[], array[])	smallint[]、int[] bigint[]、float[]	matrix_multiply(array[[2,0,0],[0,2,0],[0,0,2]], array[[3,0,3],[0,3,0],[0,0,3]])	将两个 2 维矩阵相乘。2 个矩阵必须相一致
matrix_multiply(array[], expr)	int[], float[]	matrix_multiply(array[[1,1,1], [2,2,2], [3,3,3]], 2)	将一个 2 维矩阵与一个标量相乘
matrix_transpose(array[])	与输入表达式类型相同	matrix_transpose(array [[1,1,1],[2,2,2]])	转置一个 2 维矩阵
pinv(array[])	smallint[]int[], bigint[], float[]	pinv(array[[2.5,0,0],[0,1,0],[0,0,.5]])	计算雅可比逆矩阵
unnest (array[])	任何元素集合	unnest(array['one', 'row', 'per', 'item'])	将 1 维数据转换为 ROW 记录。
MEDIAN (expr)	timestamp, timestampz, interval, float	SELECT department_id, MEDIAN(salary) FROM employees GROUP BY department_id;	在假设为连续分布的模型上执行反向分布函数。其需要一个可以转换为数字的参数表达式。其返回一个中间值或者内插值。空值被忽略。
PERCENTILE_CONT (expr) WITHIN GROUP (ORDER BY expr	timestamp, timestampz, interval, float	PERCENTILE_CONT(_percentage_) WITHIN GROUP (ORDER BY _expression_) **Example:** SELECT department_id,	在假设为连续分布的模型上执行反向分布函数。根据给定的百分比计算排序集合的线性内插值。

[DESC/ASC])		PERCENTILE_CONT (0.5) WITHIN GROUP (ORDER BY salary DESC) “Median_cont”; FROM employees GROUP BY department_id;	空值会被忽略。
PERCENTILE_DISC (expr) WITHIN GROUP (ORDER BY expr [DESC/ASC])	timestamp, timestampz, interval, float	PERCENTILE_DISC(_percentage_) WITHIN GROUP (ORDER BY _expression_) **Example:** SELECT department_id, PERCENTILE_DISC (0.5) WITHIN GROUP (ORDER BY salary DESC) “Median_desc”; FROM employees GROUP BY department_id;	在假设为连续分布的模型上执行反向分布函数。返回值是指定集合的元素而不会使用线性插值。没有命中时取之前最接近的值空值会被忽略。
sum(array[])	smallint[]int[], bigint[], float[]	sum(array[[1,2],[3,4]]) **Example:** CREATE TABLE mymatrix (myvalue int[]); INSERT INTO mymatrix VALUES (array[[1,2],[3,4]]); INSERT INTO mymatrix VALUES (array[[0,1],[1,0]]); SELECT sum(myvalue) FROM mymatrix; sum --------- {{1,3},{4,4}}	矩阵求和，可以使用二维数组作为矩阵作为输入参数
pivot_sum(label[], label, expr)	int[], bigint[], float[]	pivot_sum(array['A1','A2'], attr, value)	使用 sum 来解决重复记录的枢轴聚合
mregr_coef(expr, array[])	float[]	mregr_coef(y, array[1, x1, x2])	4 种 mregr_*使用最小二乘法计算线性回归。mregr_coef 计算回归系数。每个输入值在结果中体现为一个系数
mregr_r2(expr, array[])	float	mregr_r2(y, array[1, x1, x2])	计算回归平方差
mregr_pvalues(expr, array[])	float[]	mregr_pvalues(y, array[1, x1, x2])	计算线性回归值
mregr_tstats(expr, array[])	float[]	mregr_tstats(y, array[1, x1, x2])	计算统计数据回归值
nb_classify(text[], bigint, bigint[], bigint[])	text	nb_classify(classes, attr_count, class_count, class_total)	基于训练数据最大可能的预测新行的可能值，译者表示不太明白
nb_probabilities(text[], bigint, bigint[], bigint[])	text	nb_probabilities(classes, attr_count, class_count, class_total)	基于训练数据最大可能的预测新行的可能值，译者表示不太明白

高级分析函数示例

这些例子展示在简单的示例数据上使用高级分析函数。例子介绍了多元线性回归聚合函数和nb_classify。

线性回归聚合的例子

下面的例子在一条查询中在regr_example表上使用了4个线性回归聚合：mregr_coef、mregr_r2、mregr_pvalues和mregr_tstats。所有的聚合以因变量作为第一个参数和一系列独立变量作为第二个参数。

```
SELECT mregr_coef(y, array[1, x1, x2]),
    mregr_r2(y, array[1, x1, x2]),
    mregr_pvalues(y, array[1, x1, x2]),
    mregr_tstats(y, array[1, x1, x2])
from regr_example;
```

表regr_example为：

```
id    |y    |x1   |x2
------+-----+-----+---
1     |5    |2    |1
2     |10   |4    |2
3     |6    |3    |1
4     |8    |3    |1
```

执行该查询将产生下面的结果：

```
mregr_coef:
    {9.9475983006414e-14,1.99999999999996,1.00000000000006}
mregr_r2:
    0.864406779661098
mregr_pvalues:
    {0.99999999999998,0.454371051656925,0.783653104061144}
mregr_tstats:
    {3.14570678784565e-14,1.15470053837956,0.353553390593399}
```

如果聚合的结果是未定义的，NaN可能被返回。当数据非常小的时候可能会发生。

注意：该部分的例子较多，译者暂且只处理这些部分，留待补充或者自行查阅源文档或其他相关手册了解。该部分内容与GPDB的特性关系紧密性较低。

查询性能

GPDB支持动态分区消除和查询内存优化。GPDB在查询优化时为不同的操作动态的消除不相关分区和分配内存。该强化大大减少了查询对数据的扫描，显著加速查询处理，从而容许更大的并发。

- 动态分区消除

  在GPDB中，运行时的值只能用于在内部动态的减少分区。这样就提高了查询处理的速度。

  使用这个动态分区消除特性需要设置Server参数gp_dynamic_partition_pruning。

缺省设置为on。将该参数设置为off来关闭动态分区消除。这里译者不得不说，在多分区表做关联时该参数可能会导致完全不可预测的查询计划出现，需慎用。

- 内存优化
  GPDB对于不同用户的查询、急切释放、查询处理的不同阶段重新分配内存进行内存分配的优化。
  这些特征允许更多耗内存的查询，更快的查询和更多的并发。

查询分析

GPDB会为每个查询设计出一个查询计划。评判一个好的性能的绝对关键是为匹配的查询和数据结构选择正确的查询计划。查询计划决定了该查询将在GPDB的并行执行环境中如何被执行。通过检查糟糕性能查询的查询计划，你可能会找到有效的调优方案。

查询规划器使用数据库的统计信息来选择一个尽可能最低成本的查询计划。成本是对I/O和CPU消耗的衡量(获取磁盘页的数量)。优化的目标是最小化查询计划的执行成本。

可以使用EXPLAIN来查看查询计划。其展现了对该查询的查询计划的评估。例如：

```
EXPLAIN SELECT * FROM names WHERE id=22;
```

EXPLAIN ANALYZE会真正的执行语句，而不仅仅是计划。这对于查看规划器评估的是否接近实际情况比较有用。例如：

```
EXPLAIN ANALYZE SELECT * FROM names WHERE id=22;
```

查看 EXPLAIN 输出

查询计划是一个计划为节点组成的树。其每个节点表示一个独立的操作，比如表扫描、关联、聚合或排序。

计划应该从下向上读，每个节点得到的记录直接输入上面的节点。计划最底下的节点通常都是表扫描操作(顺序扫描、索引扫描或位图扫描 – 这里说的位图扫描与位图索引不同，可参见先关enable_bitmapscan参数)。如果查询有关联、聚合或者排序，在扫描节点之上会有额外的节点来执行这些操作。最顶端的计划节点通常是GPDB的移动节点(重分布、广播或者汇总移动)。这些操作意味着查询处理时在Segment Instance之间移动记录。

EXPLAIN的输出中每个几点都有一行，其显示基本的节点类型和规划器为该操作作出的成本评估：

- **cost** – 以获取的磁盘页数计算，就是说，1.0等于一个连续的磁盘页读取。第一个评估是开始成本(得到第一条记录时的成本)而第二个评估是总成本(得到全部记录时的成本)。需要注意的是，总成本是假设所有的记录被获取，当然可能并不会获取全部记录(比如使用了LIMIT)。
- **rows** – 该计划几点输出的记录数。通常该值小于真实处理或者扫描的数量，其会反映WHERE子句的条件对记录的过滤。顶级节点评估的数量理想状

态下与真实返回的、更新的或者删除的数据量接近。

- **width** – 所有通过该计划输出的总字节数。译者不太明白这个有什么用。

需要重点注意的是，一个上层节点的cost包含其所有子节点的cost。最顶层节点的cost包含了整个计划执行的总cost。这就是规划器要试图减小的数字。同样重要的是，要意识到的一点是，cost仅仅反映了查询规划器在意的东西。尤其特殊的是，cost总不包含结果集传输到客户端的时间。

EXPLAIN示例

要说明如何阅读EXPLAIN得到的查询计划，参考一下下面这个简单的例子：

```
EXPLAIN SELECT * FROM names WHERE name = 'Joelle';
QUERY PLAN
------------------------------------------------------------
Gather Motion 2:1 (slice1) (cost=0.00..20.88 rows=1 width=13)
    -> Seq Scan on 'names' (cost=0.00..20.88 rows=1 width=13)
        Filter: name::text ~~ 'Joelle'::text
```

从下往上阅读这个计划，执行规划器从顺序扫描names表开始。注意，WHERE子句被用作一个filter条件。这意味着，扫描操作将根据条件检查扫描的每一行，并只输出符合条件的记录。

扫描操作的结果配上传到一个汇总移动操作。在GPDB中，汇总移动就是Segment Instance向Master发送记录。该场景下，有2个Segment Instance向1个Master发送(2:1)。这个操作工作在并行查询计划的步骤1(slice1)上。在GPDB中，一个查询计划被分为多个步骤，因此查询计划可以在Segment Instance之间并行的工作。

评估的开始成本为00.00(无cost)且总成本为20.88个磁盘页面获取。规划器评估这个查询将返回一行记录。

查看 EXPLAIN ANALYZE 输出

EXPLAIN ANALYZE会真正的执行语句，而不仅仅是计划。EXPLAIN ANALYZE输出真实运行的评估结果。这对于查看规划器评估的是否接近实际情况比较有用。在输出EXPLAIN的信息之外，EXPLAIN ANALYZE还会输出如下的额外信息：

- 执行该查询花费的总时间(以毫秒计)。
- 参与一个借点计划操作的节点数(Segment Instance)，只有返回记录的Segment Instance被统计。
- 操作中Segment Instance返回的最大结果数量。如果多个Segment Instance产生相同数量的结果，最晚返回结果的被记录为最大返回数。
- 操作中返回最大结果数量的Segment Instance的ID。
- 返回最大结果数量的Segment Instance产生记录的开始时间和结束时间。如果和结束的时间一样，开始时间可以被省略(结束时间减去开始时间为花费总时间)。

EXPLAIN ANALYZE示例

为了例举如何阅读EXPLAIN ANALYZE的查询计划，这里使用”EXPLAIN示例”

相同的简单查询。需要注意的是一些额外的信息在EXPLAIN的计划中是没有的。且计划的粗体信息显示了真实运行的时间和每步结果集的数量：

```
EXPLAIN ANALYZE SELECT * FROM names WHERE name = 'Joelle';
QUERY PLAN
------------------------------------------------------------
  Gather Motion 2:1 (slice1) (cost=0.00..20.88 rows=1 width=13)
    recv: Total 1 rows with 0.305 ms to first row, 0.537 ms to end.
    -> Seq Scan on 'names' (cost=0.00..20.88 rows=1 width=13)
      Total 1 rows (seg0) with 0.255 ms to first row, 0.486 ms to end.
      Filter: name::text ~~ 'Joelle'::text
  22.548 ms elapsed
```

从下往上阅读，将看到每个计划节点操作的额外信息。花费的总时间为22.548毫秒。

该连续扫描操作仅获得一个Segment Instance返回的一条记录。花费了0.225毫秒找到第一条记录，找到全部记录花费了0.486毫秒。值得注意的是，其与计划评估结果相当，查询计划评估的返回结果集数量为1，实际执行就是1。汇总移动操作接收了1条记录(Segment Instance发送给Master)，该操作的总共花费的时间为0.537毫秒。

如何看查询计划

若一个查询表现出很差的性能，查看查询计划可能有助于找到问题点。下面是一些需要查看的东西：

- **计划中是否有一个操作花费时间超长？**查询计划中是否有一个操作花费了大部分的处理时间？例如，如果一个索引扫描比预期的时间超长，也许该索引已经处于过期状态，需要考虑重建索引。还可临时尝试使用enable_之类的参数查看是否可以强制选择不同的计划(可能会更好的效果)，这些参数可以设置特定的查询计划操作为开启或关闭状态。
- **规划器的评估是否接近实际情况？**执行EXPLAIN ANALYZE查看规划器评估的记录数与真实运行查询操作返回的记录数是否一致。如果差异巨大，可能需要在TABLE相关的COLUMN上收集更多的统计信息。相关信息可查看”维护数据库统计信息”章节。
- **选择性强的条件是否较早出现？**选择性强的条件应该被较早应用，从而使得在计划树中上传的记录更少。如果查询计划在选择性评估方面没有对查询条件作出正确的判断，可能需要在TABLE相关的COLUMN上收集更多的统计信息。相关信息可查看”维护数据库统计信息”章节。也可以尝试调整SQL语句WHERE子句的顺序。
- **规划器是否选择了最佳的关联顺序？**如查询使用多表关联，需要确保规划器选择了选择性最好的关联顺序。那些可以消除大量记录的关联应在更早的被执行，从而使得在计划树中上传的记录更少。如果规划器没有选择最佳的关联顺序，可以尝试设置join_collapse_limit=1并在SQL语句中构造特定的关联顺序，从而可以强制规划器选择指定的关联顺序。还可以尝试在TABLE相关的COLUMN上收集更多的统计信息。相关信息可查看”维护数

据库统计信息”章节。

- **规划器是否选择性的扫描分区表？**如果使用了分区，规划器是否值扫描了查询条件匹配的相关子表？父表的扫描返回0条记录(本该如此，因为父表不包含任何数据)。作为显示选择性扫描分区查询计划的例子，参见”验证分区策略”章节。
- **规划器是否合适的选择了HASH聚合与HASH关联操作？**HASH操作通常比其他类型的关联和聚合要快。记录在内存中的比较排序比磁盘快。要使用HASH操作，必须有足够的工作内存用以放置评估的记录。对于特定才查询可以尝试增加工作内存来查看是否能够获得更好的性能。如果可能，为该查询执行EXPLAIN ANALYZE，将可以得到哪些操作缓存到磁盘(由于工作内存不足导致)，多少的工作内存被使用，以及需要多少内存以保证不缓存到磁盘。例如：

  Work_mem used: 23430K bytes avg, 23430K bytes max (seg0).

  Work_mem wanted: 33649K bytes avg, 33649K bytes max (seg0) to lessen

  workfile I/O affecting 2 workers.

需要注意的是wanted信息只是一个提示，基于写出工作文件的量是不精确的。需要的最小work_mem可能会比提示的值或多或少一些。

第十二章：装载与卸载数据

本章的第一节结合例子讲述各种GPDB载入载出数据的方式。最后一节详细讲述如何格式化数据文件。

GP对大规模数据支持快速并行的数据装载和卸载，除了单文件以及小量数据的导入导出。本章包含以下内容：

- GPDB装载命令概述
- 装载数据到GPDB
- 定义外部表 – 示例
- 从GPDB卸载数据
- 转换XML数据
- 格式化数据文件

GPDB 装载命令概述

本节简要的概述GPDB提供的用于装载和卸载数据的命令。

- 关于外部表
- 关于gpload
- 关于COPY

关于外部表

外部表允许用户像访问标准数据库表一样访问外部文件。结合GP的并行文件分配程序(gpfdist)，外部表支持在装载和卸载数据时全并行化利用所有Segment Instance的资源。GPDB还可以利用Hadoop分布式文件系统的并行架构来访问其存储的文件。

GPDB提供两种类型的外部表 – 可读外部表用于数据装载、可写外部表用于数据卸载。外部表可以基于文件亦可基于WEB，这两种都可以是可读的或者可写的。

可读外部表为数据仓库常用的抽取、转换、装载(ETL)任务提供了简易的方式。外部表的数被GPDB的所有Segment Instance并行读取，因此大的装载操作可以被尽可能快的处理。一旦外部表被定义，就可以直接使用SQL命令并行对数据进行查询。可以对外部表进行比如查询、关联或者排序等操作。还可以为外部表创建视图。不过外部表不可以做修改。DML(UPDATE、INSERT、DELETE、TRUNCATE)操作是不允许的。

有两种可读外部表 – 常规的和WEB的。二者的主要区别在于他们的数据源。常规的外部表访问静态的平面文件，而WEB外部表访问动态数据源 – 既可以是一个使用http://协议访问的web服务，也可以是运行OS的命令或脚本。

当一个查询使用一个常规的外部表，该外部表被认为是可重读的，因为在该查

询期间数据是静态的。而对于WEB外部表，数据是不可重读的，因为在该查询的执行期间数据可能会发生变化。

可写外部表用以从数据库表中选择记录并输出到文件、命名管道或其他可执行程序。比如，可以从GPDB中卸载数据并发送到一个可执行程序，该程序连接到其他数据库或者ETL工具并装载数据到其他地方。可写外部表还可以用于输出到GPDB的并行MapReduce计算。

当一个可写外部表被定义后，数据即可从数据库表中被选择并插入到该可写外部表。可写外部表只允许INSERT操作 – SELECT、UPDATE、DELETE或TRUNCATE是不允许的。可写外部表输出数据到一个可执行程序，该程序要能够接受流输入数据。

关于 gpload

gpload是一个数据装载命令，其以GP外部表并行装载作为工作接口。通过一个按照YAML格式定义的装载说明控制文件，gpload调用GP的并行文件服务程序，创建基于定义好的数据源的外部表，然后执行INSERT、UPDATE或者MERGE操作，将源数据装载到目标数据库表中。

关于 copy

GPDB同样提供了对于标准PostgreSQL装载和卸载数据COPY命令的支持，其用于装载和卸载数据。COPY不具有并行装载/卸载的机制，这意味着数据的装载/卸载是通过GP的Master Instance单进程处理的。因此，若涉及的是小量数据，COPY提供了一种便捷的单事务方案来操作数据进出数据库，这样就不需要管理员事先设置好外部表。

装载数据到 GPDB

本节讲述如何使用并行装载操作(基于文件的外部表、WEB外部表和gpload)和非并行装载操作(COPY)把数据装载到GPDB中。本节包含如下内容：

- 基于文件的外部表
- 使用GP并行文件服务(gpfdist)
- 使用Hadoop分布式文件系统表
- 创建和使用WEB外部表
- 使用外部表装载数据
- 装载和卸载自定义数据
- 处理装载错误数据
- 使用gpload装载数据
- 使用COPY装载数据
- 数据装载性能技巧

基于文件的外部表

在创建一个外部表定义时，必须指定输入文件的格式个外部数据源的位置。更多关于输入文件格式，参照”格式化数据文件”章节。有3种可以用来访问外部表数据源的协议，但在使用CREATE EXTERNAL TABLE语句时不能混合这些协议。这些协议如下：

gpfdist

如果使用gpfdist://协议，在外部表指定的文件所在的主机上必须运行GP文件分发程序(gpfdist)。该程序指向一个给定的目录，并行的为GPDB的所有Segment Instance提供外部数据文件服务。如果文件使用了gzip或者bzip2压缩(文件扩展名分别为.gz和.bz2)，gpfdist会自动解压 这些文件(在$PATH中需提供gunzip和bunzip2)。

不管在定义外部表时指定了多少个URI，所有Primary Segment Instance并行访问外部文件。在使用CREATE EXTERNAL TABLE语句时可以使用多个gpfdist数据源来提升外部表的扫描性能。

在指定gpfdist获取哪些文件时，可以使用通配符或者C风格的模式匹配来执行多个文件。被指定的文件是假设以gpfdist运行的目录作为相对路径的(运行的目录在启动gpfdist程序时指定)。

gpfdist位于GPDB Master主机的$GPHOME/bin目录下。关如何同外部表一起使用文件分发程序的更多信息可以参考gpfdist的相关部分文档。

gpfdists

gpfdists是gpfdist的安全版本，其开启了加密通信并确保了文件服务与GPDB之间的安全认证，以防御类似窃听和通过中间件的攻击。

其协议实现了在客户端/服务端之间的SSL安全方案，具备了下列重要特征：

- 需要客户端证书
- 不支持多语言证书
- 不支持证书废止列表(CRL)
- TLSv1协议使用TLS_RSA_WITH_AES_128_CBC_SHA加密算法。SSL参数不能修改
- 支持SSL重新握手
- SSL忽略主机不匹配参数被设置为false
- 包含密码的私有密钥不受gpfdist文件服务支持，也不受GPDB支持
- 使用OS适用的证书是用户的责任。通常转换证书在 https://www.sslshopper.com/ssl-converter.html受到支持。

译者注：这部分内容译者认为不应作为学习重点，如确需这方面的使用说明，请自行参考原官方文档。通常情况下，GPDB作为内部系统，无需使用加密的

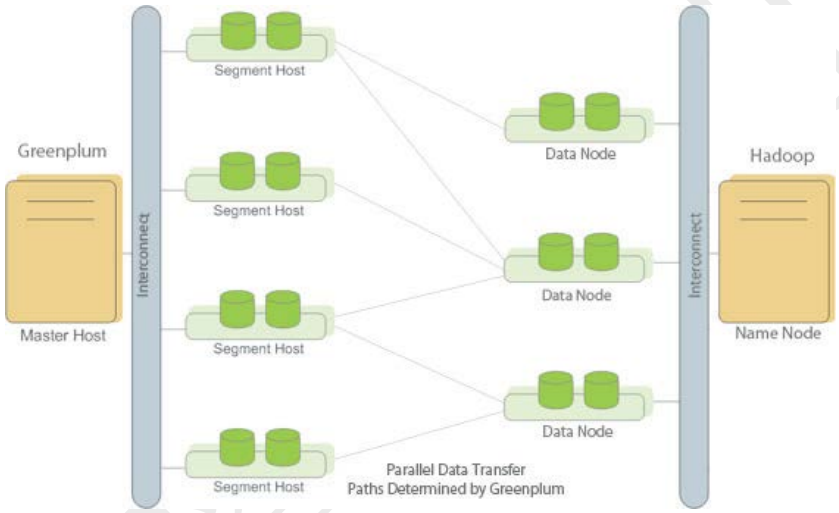
gpfdist协议。这仅代表译者观点，具体怎么使用与译者无关。

gphdfs

该协议指定了一个可以在HDFS上包含通配符的路径。TEXT和自定义格式允许在HDFS上使用。

在GP连接到HDFS文件时，所有数据将从HDFS数据节点被并行读取到GP的Segment Instance以快速处理。GPDB来确定Segment Instance与HDFS Data Node之间的连接。

每个GP Segment Instance只读取一组Hadoop数据块。对于写来说，每个GP Segment Instance值写该Instance包含的数据。



FORMAT子句用来指定外部表文件是什么格式。有效的文件格式是，划界的text(TEXT)对于所有协议有效，逗号分割的CSV格式对于gpfdist和file协议有效，格式的选项与PostgreSQL的COPY命令类似，还有适合gphdfs协议的TEXT和自定义格式。如果数据文件没有使用缺省的列分隔符、转义字符、空替换字符等，需要指定额外的格式选项以确保外部文件可以被GPDB正确的读取。gpfdist协议和gphdfs协议要求一次性完成设置。

外部表中的错误数据

缺省状态下，如果外部表包含错误数据，整个命令会失败且没有数据被装载到目标数据库表中。为了在装载正确格式的记录时隔离错误数据，需在定义外部表时使用单条记录出错处理。查看”处理装载错误数据”章节。

有一个系统视图pg_max_external_files用来确定在每个外部表中允许多少个外部文件。该值列出了每个Segment主机可用的文件数(如使用了file://协议)。例如：

```
SELECT * FROM pg_max_external_files;
```

如果使用gpfdist://协议，下面这个服务器参数可用以设置最大多少Segment Instance访问同一个gpfdist文件分发程序以并发获取外部表数据。缺省为64个Segment Instance：

```
gp_external_max_segs = <integer>
```

gpfdist程序通过HTTP协议提供文件服务。带有LIMIT子句的外部表查询将在获取相应行数之后中断HTTP连接，这会产生一个HTTP嵌套字异常。如果在查询使用了gpfdist://协议或者http://协议的外部表时使用了LIMIT子句，其忽视来自gpfdist或者web服务器的HTTP 嵌套字异常是安全的 – 数据已然正确的返回到DB客户端。

在备份/恢复操作中，仅仅外部表或者WEB外部表的定义会被备份或恢复。并不包含其相关的数据源。

使用 GP 并行文件服务(gpfdist)

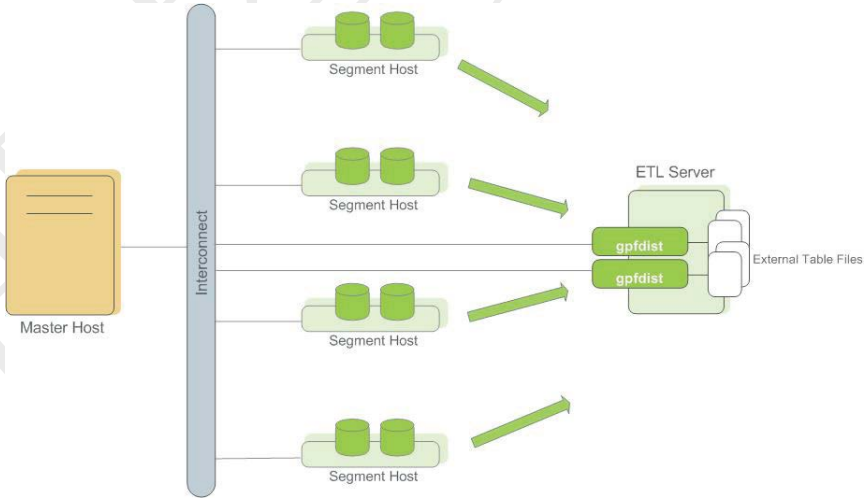
为了获取最佳的性能和便捷的管理，gpfdist协议应作为首选。使用gpfdist的优势在于其可以确保在读取外部表的文件时，GPDB系统的所有Segment Instance可以完全被利用起来。本节结合外部表讲述设置和管理GP并行文件服务程序。

- 关于gpfdist的设置与性能
- 控制节点并行度
- 安装gpfdist
- 启动和停止gpfdist
- gpfdist故障诊断

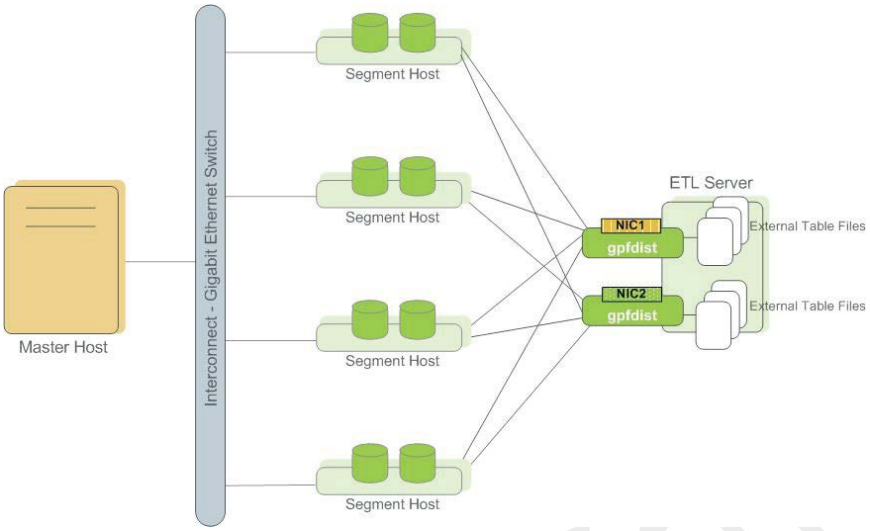
关于gpfdist的设置与性能

为了最大化ETL系统带宽而运行gpfdist时应该考虑下面几点因素：

- 如果ETL机器配备了多个网口，在ETL主机上运行一个gpfdist服务，那么在定义外部表时，应将所有网口对应的HostName在LOCATION子句中声明。这样将允许GP Segment主机同时使用全部网口与ETL主机进行网络通信。



- 在ETL主机上运行多个gpfdist并将外部数据均匀的分拆到各gpfdist服务。例如，ETL系统有两个网口，可以运行两个gpfdist服务以最大化装载性能。这时需要将外部表数据文件均匀的分割到两个gpfdist程序。



注意：GP建议，在为gpfdist准备文件时，最好使用竖线(|)分割而不是逗号。因为在使用逗号分割时，字符串爆赞单引号或双引号中。gpfdist需要将双引号剥离。如果使用竖线分割，gpfdist不需要剥离字符串，这样将表现出更快的性能。

控制节点并行度

可以使用下面的服务器参数来控制有多少Segment Instance同时访问一个独立的gpfdist程序。缺省值为64。这样可以控制处理外部表文件的Segment Instance数量，以保留一些Segment Instance做其他的数据库处理。该参数可以在Master Instance的postgresql.conf文件中设置：

gp_external_max_segs

安装gpfdist

gpfdist程序安装在GPDB的Master主机的$GPHOME/bin目录下。最可能的是在GP Master之外的机器运行gpfdist，比如部署了ETL的机器。可以这样在其他机器上安装gpfdist：

1.从GP安装机器的$GPHOME/bin位置拷贝gpfdist到远程机器

2.将gpfdist添加到$PATH.

启动和停止gpfdist

要启动gpfdist，必须指定其提供文件服务的目录以及运行的端口(缺省为HTTP端口8080)。

在后台启动gpfdist(日志信息和出错信息输出到日志文件)：

$ gpfdist -d /var/load_files -p 8081 -l /home/gpadmin/log &

要在同一个ETL主机启动多个gpfdist服务，为每个服务指定不同的目录和端口。例如：

$ gpfdist -d /var/load_files1 -p 8081 -l /home/gpadmin/log1 &

$ gpfdist -d /var/load_files2 -p 8082 -l /home/gpadmin/log2 &

停掉一个正在后台运行的gpfdist服务：

--第一步找到其进程号

$ ps ax | grep gpfdist (Linux)

$ ps –ef | grep gpfdist (Solaris)

--第二部杀掉该进程，例如：

```
$ pg_cancel_backend(3456)
$ gpkill 3456
```

gpfdist故障诊断

需要注意的是，gpfdist在运行时由Segment Instance访问。因此，必须确保GP Segment可以具有访问gpfdist的网络。gpfdist服务是简单的web服务，所以可以供GP集群(Segment或Master)使用下面的命令来测试连接性：

```
$ wget http://gpfdist_hostname:port/filename
```

另外需要确保CREATE EXTERNAL TABLE定义了正确的Host Name，Port以及gpfdist的文件名(文件名和路径需要使用gpfdist启动的相对路径)。参见”定义外部表 – 示例”相关章节。

使用 Hadoop 分布式文件系统表

GPDB使用gphdfs协议支持并行架构Hadoop分布式文件系统数据的读写。使用HDFS包含三个步骤，如下所示：

- 安装设置
- HDFS协议授权
- 在外部表定义中指定HDFS数据

译者提示：由于该部分内容译者目前还未触及，暂不翻译，待译者掌握后再补充。

创建和使用 WEB 外部表

使用CREATE EXTERNAL WEB TABLE可创建GPDB的WEB表。WEB表是一种外部表，其可以像使用常规的数据库表一样用以访问动态的数据源。WEB表的数据是动态的(意味着在执行查询期间数据可能会发生变化)。因此，执行规划器选择的计划不允许重复扫描WEB表的数据。

WEB外部表的定义有两种形式。可以使用CREATE EXTERNAL WEB TABLE命令创建任何一种形式，但不能混合使用这两种形式。

- **WEB URL。**使用http://协议指定WEB服务器上文件的LOCATION。该WEB数据文件必须在GP Segment可以访问的WEB服务上。URL的数量将对应并行访问WEB表的Segment Instance数量。比如，GPDB系统有8个Primary Segment Instance，而指定了2个外部文件，将仅有2个Segment Instance同时并行访问WEB表的数据。
- **OS命令。**在一个或数个Segment上指定执行SHELL命令或者脚本，这些命令或脚本的输出作为WEB表访问时的数据。使用EXECUTE子句定义的外部表，将在指定的数个Segment Host上执行指定的OS SHELL命令或脚本。缺省状态下，命令将被所有Segment Hosts上所有活动的Segment Instance执行。例如，如果每个Segment Host有4个Primary Segment Instance在运行，命令将在每个Segment Host上执行4次。不过，可以选择限制执行WEB表命令的Segment Instance数量。

  WEB表的数据由执行WEB表语句时命令的输出结果构成。WEB表定义中

所有Segment Instance(通过ON子句指定那些实例)并行执行该命令。

定义命令型WEB外部表

在外部表定义中指定的命令或程序必须放置到所有Segment Host上(如果指定由所有Segment Host来执行的话)。如果在外部表中用到了环境变量(比如$PATH)，需要注意的是，命令是从数据库执行而不是从登录的SHELL。因此，当前用户的.bashrc或者.profile文件不会被装载。不过，在外部表定义的EXECUTE子句中，可以根据需要设置环境变量，比如：

```
=# CREATE EXTERNAL WEB TABLE output (output text)
   EXECUTE 'PATH=/home/gpadmin/programs; export PATH; myprogram.sh'
   FORMAT 'TEXT';
```

当然，任何需要被执行的脚本必须放置在Segment Host上的相同目录下，并确保gpadmin用户有可执行权限。

例如，下面的命令定义了一个外部表，每个Segment Instance都执行一次脚本：

```
=# CREATE EXTERNAL WEB TABLE log_output
   (linenum int, message text)
   EXECUTE '/var/load_scripts/get_log_data.sh' ON HOST
   FORMAT 'TEXT' (DELIMITER '|');
```

定义URL型WEB外部表

一个URL型的WEB外部表类似与文件型的外部表，区别是，URL型通过HTTP协议从一个WEB服务获取数据。WEB外部表的数据是动态的，这与文件型外部表不同，数据不可以被重复扫描。

例如，下面的的命令定义了各个外部表，从Server的不同URL获取数据：

```
=# CREATE EXTERNAL WEB TABLE ext_expenses (name text,
   date date, amount float4, category text, description text)
   LOCATION ( 'http://intranet.company.com/expenses/sales/file.csv',
   'http://intranet.company.com/expenses/exec/file.csv',
   'http://intranet.company.com/expenses/finance/file.csv',
   'http://intranet.company.com/expenses/ops/file.csv',
   'http://intranet.company.com/expenses/marketing/file.csv',
   'http://intranet.company.com/expenses/eng/file.csv' )
FORMAT 'CSV' ( HEADER );
```

使用外部表装载数据

一旦定义了外部表，并且在正确的目录(如果使用gpfdist协议的话已经启动了GP文件服务)放有必要的数据，即可像普通数据库表一样使用SELECT FROM(这里说的是可读外部表)来访问外部表。例如，想将一部分数据装载到一张数据库表中，可以像这样操作：

```
=# INSERT INTO expenses_travel
   SELECT * from ext_expenses where category='travel';
```

或者想要快速装载全部数据到一个新的数据库表中：

```
=# CREATE TABLE expenses AS SELECT * from ext_expenses;
```

译者注：使用CREATE AS比使用INSERT INTO效率要高，尤其在参数checkpoint_segments设置的值较小时更为显著。CREATE AS为一次刷入，而INSERT INTO每checkpoint_segments个检查点一次刷入。

装载和卸载自定义数据

在GPDB中有两种自定义方案可用于装载和卸载数据。下面介绍这两种自定义方案：

- 使用自定义格式
- 使用自定义协议

使用自定义格式

自定义格式用于导入导出TEXT和CSV两种格式之外的数据。通常，使用自定义格式有如下3个特别的步骤：

1. 编写输入输出函数并编译到共享库中。
2. 在GPDB中通过CREATE FUNCTION指定共享库函数。
3. 将这些函数与CREATE EXTERNAL TABLE的Formatter子句关联。

对于固定宽度数据来说，前两步在GPDB中已经完成。函数名称分别为fixedwith_in和fixedwidth_out。

导入导出固定宽度数据

在GPDB中固定宽度数据的函数可以直接使用。需要做的是指定自定义的格式和在formatter参数中提供函数名称，下面的例子，简单的展示了读取全部数据。

```
CREATE READABLE EXTERNAL TABLE students (
   name varchar(20), address varchar(30), age int)
LOCATION ('file://<host>/file/path/')
FORMAT 'CUSTOM' (formatter=fixedwidth_in,
   name='20', address='30', age='4');
```

对于导入固定宽度数据还有一些其他的选项。

- 读取全部数据

  要装载固定宽度数据一行的全部字段，必须按照他们的物理顺序装载。必须指定列的长度，且不能指定开始于结束位置。指定宽度参数中的字段名称必须与命令开始处CREATE…TABLE指定的顺序一致。

- 设置空白和空值字符

  补尾的空白缺省被修剪掉。要保留补尾的空白，使用preserve_blanks=on选项。使用null='null_string_value'选项指定空值字符。

  --如果指定了选项preserve_blanks=on，必须指定控制字符，否则将产生错误。

  --如果指定了选项preserve_blanks=off，并且列中仅包含空白，如果没有定义空值字符，该字段将会被作为空值写到表中。如果空值定义了，将写入一个空字符到表中。

- 指定行的结尾

  使用参数line_delim='line_ending'指定行的结尾字符。下面的列举的可以覆盖绝大部分场景。E指定该字符串为转义字符。

  ```
  line_delim=E'\n'
  ```

line_delim=E'\r'

line_delim=E'\r\n'

line_delim='' (这是设置没有行分隔符)

注意：如果在数据库中已经存在line_delim，将不能在CREATE READABLE EXTERNAL TABLE语句中指定该参数。

示例

下面的例子展示了读入固宽度度数据。

例1 – 加载一张包含所有字段定义的表

```
CREATE READABLE EXTERNAL TABLE students (
    name varchar(20), address varchar(30), age int)
LOCATION ('file://<host>/file/path/')
FORMAT 'CUSTOM' (formatter=fixedwidth_in,
    name=20, address=30, age=4);
```

例2 – 加载一张保留补尾空白的表

```
CREATE READABLE EXTERNAL TABLE students (
    name varchar(20), address varchar(30), age int)
LOCATION ('gpfdist://<host>:<portNum>/file/path/')
FORMAT 'CUSTOM' (formatter=fixedwidth_in,
    name=20, address=30, age=4, preserve_blanks='on',null='NULL');
```

例3 – 加载一张没有行分隔符的表

```
CREATE READABLE EXTERNAL TABLE students (
    name varchar(20), address varchar(30), age int)
LOCATION ('file://<host>/file/path/')
FORMAT 'CUSTOM' (formatter=fixedwidth_in, name=20, address=30, age=4, line_delim='')
```

例4 – 加载一张以\r\n为行分隔符的表

```
CREATE WRITABLE EXTERNAL TABLE students_out (
    name varchar(20), address varchar(30), age int)
LOCATION ('gpfdist://<host>:<portNum>/file/path/')
FORMAT 'CUSTOM' (formatter=fixedwidth_out,
    name=20, address=30, age=4, line_delim=E'\r\n');
```

使用自定义协议

如果现有的协议(gpfdist、http、file等)不能够很好的用于访问数据，可以编写自定义的协议。自定义协议不指定数据格式。自定义协议可以使用标准数据格式如TEXT、CSV以及自定义格式(如固定宽度格式)。

在创建自定义协议时，通过CREATE TRUSTED PROTOCOL允许Owner之外的用户访问它。如果协议是不可信的，将不能赋予任何其他用户权限用以读写数据。在可信协议创建之后，即可通过GRANT命令指定那些用户能够使用该协议。

- 允许特定用户使用可信协议创建可读外部表

  GRANT SELECT ON PROTOCOL <protocol name> TO <user name>

- 允许特定用户使用可信协议创建可写外部表
  GRANT INSERT ON PROTOCOL <protocol name> TO <user name>
- 允许特定用户使用可信协议创建可读以及可写外部表
  GRANT ALL ON PROTOCOL <protocol name> TO <user name>

下面是使用自定义协议概要性的步骤。

1. 使用预先定义的API，用C编写发送和接受函数。可以选择编写一个验证函数。这些函数要被编辑并注册到GPDB中。
   下面的例子是使用编译后的导入导出代码。

```
CREATE FUNCTION myread() RETURNS integer
    as '$libdir/gpextprotocol.so', 'myprot_import'
LANGUAGE C STABLE;
CREATE FUNCTION mywrite() RETURNS integer
    as '$libdir/gpextprotocol.so', 'myprot_export'
LANGUAGE C STABLE;
```

2. 创建一个访问这些函数的协议

```
CREATE TRUSTED PROTOCOL myprot(readfunc='myread', writefunc='mywrite');
```

3. 将协议授权给必要的用户

```
GRANT ALL ON PROTOCOL myprot TO otheruser
```

4. 在可读或者可写外部表中使用该协议

```
CREATE WRITABLE EXTERNAL TABLE ext_sales(LIKE sales)
LOCATION ('myprot://<meta>/<meta>/…')
FORMAT 'TEXT';

CREATE READABLE EXTERNAL TABLE ext_sales(LIKE sales)
LOCATION('myprot://<meta>/<meta>/…')
FORMAT 'TEXT';
```

译者注：关于自定义协议，译者也不曾用过，无法给出更多解释，一般来说不太需要用到这个功能，作为学习来说该部分可以忽略。

处理装载错误数据

对于可读外部表来说，最常见的应用就是SELECT数据并装载到常规数据库表。代表性的操作为CREATE TABLE AS SELECT或INSERt INTO SELECT命令，这里的SELECT语句查询外部表。缺省情况下，如果外部表包含错误数据，整个命令失败，且没有数据被装载到目标数据库表。为了将正确的数据装载并隔离错误的数据，可以在定义可读外部表时使用CREATE EXTERNAL TABLE命令结合使用SEGMENT REJECT LIMIT子句。

- 拒绝限制count参数可用于指定记录数(缺省)，或者使用PERCENT指定记录百分比。
- 保存错误记录以备将来的检查，使用LOG ERRORS INTO子句指定错误记录日志表。

定义一个带有单条记录错误隔离的外部表

可以使用SEGMENT REJECT LIMIT指定每个Segment Instance可接受的错误记录数，可以按照行数设置或者百分比设置。如果错误记录达到了SEGMENT

REJECT LIMIT的设置，整个外部表操作失败，且没有数据被处理。需要注意的是，错误限制是Segment Instance的设置，为不是整个集群的设置。如果SEGMENT REJECT LIMIT没有达到，所有正确的记录会被处理，而错误的记录被隔离到错误日志表中(如果指定了错误日志表)。下面的例子使用了LOG ERRORS INTO子句，因此，任何包含错误格式的记录都会被记录到错误日志表err_expenses中。

例如：

```
=# CREATE EXTERNAL TABLE ext_expenses ( name text,
    date date, amount float4, category text, desc text )
LOCATION ('gpfdist://etlhost-1:8081/*',
          'gpfdist://etlhost-2:8082/*')
FORMAT 'TEXT' (DELIMITER '|')
LOG ERRORS INTO err_expenses SEGMENT REJECT LIMIT 10 ROWS;
```

在使用了SEGMENT REJECT LIMIT后，外部表将以单条记录错误隔离模式扫描数据。这对于使用CREATE TABLE AS SELECT和INSERT INTO SELECT装载数据时隔离错误记录是很有帮助的。单条记录错误隔离模式仅适用于外部表记录格式错误 – 例如，多余或缺失列、错误数据类型的列、错误的客户端编码。约束错误不会被检查，不过在使用外部表装载数据时可以通过限制SELECT外部表的数据来过滤约束错误。例如，要排除重复记录错误：

```
=# INSERT INTO table_with_pkeys
    SELECT DISTINCT * FROM external_table;
```

查看错误日志表中出错记录

若使用了单条记录错误隔离模式，任何格式错误的记录会被隔离记录到一个错误日志表中，该错误日志表包含如下的列属性：

字段名	类型	描述
cmdtime	timestampz	错误出现的时间
relname	text	扫描的外部表名称或者 COPY 命令的目标表名称
filename	text	出现该错误的被装载文件名
linenum	int	使用 COPY 命令时装载文件的行号，对于 file://协议和 gpfdist://协议的外部表，对应出错文件的行号
bytenum	int	如果使用的是 gpfdist://协议，记录的是错误发生时的 byte 偏移量。由于 TEXT 文件使用块处理方式，无法记录行号。CSV 是按行处理的，所以其可以记录行号
errmsg	text	出错描述信息
rawdata	text	被拒绝记录的原始数据
rawbytes	bytea	如果是数据库的编码与客户端编码无法转换，将无法记录 rawdata。取而代之的是记录这些字符的 8 进制 ASCII 编码

可以使用SQL命令查询错误日志表的记录。例如：

```
=# SELECT * from err_expenses;
```

在错误日志表数据中识别无效CSV文件

如果逗号分割的(CSV)文件存在无效格式，错误日志表的rawdata列中可能会包含多个组合的记录行。比如，某个字段的关闭引号缺失，接下来的一行也会被当作当前行的记录。如果发生了这种情况，GP会在字符解析达到64K时自动停止，并将这些数据记录到错误日志表中作为一条记录，重置引号标识，然后继

续解析。如果装载处理中发生了3次，该文件将被作为无效文件处理，并且整个装载失败，获得一个提示信息”rejected N or more rows”。查看相关的”CSV格式文件转义”章节获取更过关于在CSV文件中正确使用引号的信息。

在表之间移动数据

可以使用CREATE TABLE AS或者INSERT…SELECT来装载外部表或者WEB外部表到另一个常规数据库表，且数据被外部表或者WEB外部表以并行(如果WEB外部表也定义了并行性)的方式被装载。

如果外部表文件或者WEB外部表数据源包含错误，任何读取操作都会失败。类似与COPY，从外部表或者WEB外部表装载的整体操作要么全部成功要么整体失败。

使用 gpload 装载数据

gpload是GP使用可读外部表和GP并行文件服务gpfdist装载数据的一个命令包装。其允许通过使用配置文件的方式设置数据格式等来创建外部表定义。

使用gpload

1. 首先、需确保环境安装了gpload相关环境。需要相关的依赖文件(在GPDB的安装目录中存在)如gpfdist和Python，还需要访问GP Segment Host能够访问的网络。
2. 创建装载控制文件。该文件为YAML格式，其指定了GPDB的连接信息，gpfdist配置信息，外部表选项，以及数据格式。例如：

```
---
VERSION: 1.0.0.1
DATABASE: ops
USER: gpadmin
HOST: mdw-1
PORT: 5432
GPLOAD:
    INPUT:
      - SOURCE:
          LOCAL_HOSTNAME:
                - etl1-1
                - etl1-2
                - etl1-3
                - etl1-4
          PORT: 8081
          FILE:
                - /var/load/data/*
      - COLUMNS:
          - name: text
          - amount: float4
          - category: text
```

```
            - desc: text
            - date: date
        - FORMAT: text
        - DELIMITER: '|'
        - ERROR_LIMIT: 25
        - ERROR_TABLE: payables.err_expenses
    OUTPUT:
        - TABLE: payables.expenses
        - MODE: INSERT
    SQL:
    - BEFORE: "INSERT INTO audit VALUES('start', current_timestamp)"
    - AFTER: "INSERT INTO audit VALUES('end', current_timestamp)"
```

3. 通过装载控制文件运行gpload命令。例如：

```
gpload -f my_load.yml
```

使用 gphdfs 协议装载数据

如使用INSERT INTO将gphdfs协议外部表定义的HDFS数据插入GP表中，数据将被并行拷贝。例如：

```
INSERT INTO gpdb_table (select * from hdfs_ext_table);
```

使用 COPY 装载数据

命令COPY FROM把数据从一个文件(或者标准输入流)拷贝到一个表中(不管表中已存在什么数据，只是追加)。如果从文件拷贝数据，该文件必须是Master主机可以访问的位置，并且，文件名必须是以Master主机作为视角。当指定了STDIN或者STDOUT，数据将在连接着的客户端与Master Server之间传输。COPY是非并行的，这意味着，数据是通过GP Master Instance的独立进程进行装载。

为了最大化COPY的性能和吞吐量，可以考虑使用多个COPY命令在不同的SESSION并行执行不同数据部分的装载，将数据尽量分布在所有的并发进程。要最大化吞吐量，可以考虑每个CPU执行一个并发的COPY。

使用单条记录错误隔离模式运行COPY

缺省状态，COPY操作将会在首个错误时终止，这意味着如果被装载的数据包含哪怕一个错误，这个操作失败并不会有数据被装载。作为可选项，COPY FROM命令可以使用单条记录隔离模式。在这种模式下，错误的记录将被忽略，而所有正确格式的数据依然被装载，隔离的错误记录包括缺少列、错误数据类型的列、无效的编码格式等。在目前版本，单条记录错误隔离模式仅对错误的格式有效，约束类的错误如NOT NULL、UNIQUE等仍属于”要么成功要么全部失败”的输入模式，需要注意的是，这与外部表的隔离模式是不同的，外部表加载时会忽略这些约束。

当在COPY FROM使用SEGMENT REJECT LIMIT子句，该操作将使用单条记录错误隔离模式。用户可以指定可接受的错误行数(按照每个Segment Instance计算)，在达到了这个限制之后操作将失败并且不会有数据被装载。需要注意的是，错误记录是按照每个Segment Instance计算，而不是按照整个操作计算。如果每个Segment Instance拒绝的限制没有达到，所有不包含错误的记录都将被装载，而错误的记录被丢弃。若希望将错误记录留存用以检查，可以选择LOG ERRORS INTO子句指定一个错误记录日志表。任何包含错误格式的记录都将被保存在指定的错误记录日志表中。例如：

```
=> COPY country FROM '/data/gpdb/country_data'
     WITH DELIMITER '|' LOG ERRORS INTO err_country
     SEGMENT REJECT LIMIT 10 ROWS;
```

关于错误记录日志表的信息，可查看”查看错误日志表中出错记录”相关章节。

数据装载性能技巧

在装载前删除索引 – 若装载一张新创建的表，最快的方式是先创建表，再装载数据，然后再创建任何需要的索引。在已存在的数据上创建索引比不断的递增索引要快。若向一个已有的表添加大量数据，更快的方式可能是，先删除索引，然后装载数据，然后再重新创建索引。临时的增加maintenance_work_mem服务器参数可以提升CREATE INDEX名的速度，不过，这可能对于装载数据本身没有性能提升。应该考虑在没有系统用户在用时进行删除并重新建立索引操作。

在装载之后运行ANALYZE -- 在修改了表中的大部分数据之后，强烈建议执行ANALYZE操作。执行ANALYZE(或者VACUUM ANALYZE)以确保查询规划器拥有最新的统计信息。如果没有统计信息或者统计信息过时了，规划器可能会根据过期的统计信息或不存在的统计信息选择一个较差的查询计划，以至导致较差的性能。

在装载出错后执行VACUUM – 如果运行的不是单条记录错误隔离模式，装载操作将在首次发生错误时终止。而目标表已经收到了错误发生前的记录。这些记录是无法被访问的，但它们仍占据磁盘空间。若这种情况发生在大的装载操作上，会导致大量的磁盘空间浪费。执行VACUUM命令可以回收这些浪费的空间。

定义外部表 – 示例

使用CREATE EXTERNAL TABLE命令定义一个外部表，结合LOCATION参数和FORMAT参数指定外部文件的位置和格式。该命令不会装载数据到表中。接下来的例子展示如何使用不同的协议连接外部文件。每个CREATE EXTERNAL TABLE命令只能包含一种协议。每个协议的细节内容见接下来的例子。

注意：如果使用IPv6，数字IP地址需要使用方括号括起来。

例1 – 启动GP文件服务(gpfdist)

在创建一个使用gpfdist协议的外部表之前，文件服务程序gpfdist比如处于运行状态。下面的命令是在后台启动gpfdist文件服务程序，其端口为8081，服务文件目录为/var/data/staging：

```
gpfdist -p 8081 -d /var/data/staging -l /home/gpadmin/log &
```

例2 – 多网口主机上的单个GP文件服务(gpfdist)

创建一个使用gpfdist协议名为ext_expenses的可读外部表。文件的格式为竖线(|)分割。

```
=# CREATE EXTERNAL TABLE ext_expenses ( name text,
date date, amount float4, category text, desc1 text )
LOCATION ('gpfdist://etlhost-1:8081/*',
          'gpfdist://etlhost-2:8081/*')
FORMAT 'TEXT' (DELIMITER '|');
```

例3 – 多个GP文件服务(gpfdist)

创建一个使用gpfdist协议名为ext_expenses的可读外部表，其匹配所有txt扩展名的文件。这些文件的格式为竖线(|)分割，空白字符当作null。

```
=# CREATE EXTERNAL TABLE ext_expenses ( name text,
    date date, amount float4, category text, desc1 text )
LOCATION ('gpfdist://etlhost-1:8081/*.txt',
          'gpfdist://etlhost-2:8082/*.txt')
FORMAT 'TEXT' ( DELIMITER '|' NULL ' ') ;
```

例4 – 多个安全GP文件服务(gpfdists)

创建一个使用gpfdists协议名为ext_expenses的可读外部表，其匹配所有txt扩展名的文件。这些文件的格式为竖线(|)分割，空白字符当作null。

第一步，使用—ssl参数开启gpfdists。然后执行如下命令。

```
=# CREATE EXTERNAL TABLE ext_expenses ( name text,
    date date, amount float4, category text, desc1 text )
LOCATION ('gpfdists://etlhost-1:8081/*.txt',
          'gpfdists://etlhost-2:8082/*.txt')
FORMAT 'TEXT' ( DELIMITER '|' NULL ' ') ;
```

例5 – 带有错误数据日志的单个GP文件服务(gpfdist)

创建一个使用gpfdist协议名为ext_expenses的可读外部表，其匹配所有txt扩展名的文件。这些文件的格式为竖线(|)分割，空白字符当作null。

该外部表为单条记录错误隔离模式。一个出错表(err_customer)配指定。任何格式错误的数据将带着出错描述被丢弃到err_customer表中。err_customer表稍后可以查询以确定错误的原因并在定位错误之后可以重新加载被拒绝的数据。如果任何Segment Instance获得的错误格式的数据记录大于5(SEGMENT REJECT LIMIT value指定)，这个外部表操作将被终止，并且没有记录被处理。

```
=# CREATE EXTERNAL TABLE ext_expenses ( name text,
    date date, amount float4, category text, desc1 text )
LOCATION ('gpfdist://etlhost-1:8081/*.txt',
          'gpfdist://etlhost-2:8082/*.txt')
FORMAT 'TEXT' ( DELIMITER '|' NULL ' ')
```

```
LOG ERRORS INTO err_customer SEGMENT REJECT LIMIT 5;
```

创建一个与上面的定义一样的外部表，但使用CSV格式文件：

```
=# CREATE EXTERNAL TABLE ext_expenses ( name text,
    date date, amount float4, category text, desc1 text )
LOCATION ('gpfdist://etlhost-1:8081/*.txt',
          'gpfdist://etlhost-2:8082/*.txt')
FORMAT 'CSV' ( DELIMITER ',' )
LOG ERRORS INTO err_customer SEGMENT REJECT LIMIT 5;
```

例6 – HDFS上的TEXT格式

创建一个使用gphdfs协议名为ext_expenses的可读外部表，其匹配所有txt扩展名的文件。这些文件的格式为竖线(|)分割。

```
=# CREATE EXTERNAL TABLE ext_expenses ( name text,
    date date, amount float4, category text, desc1 text )
LOCATION ('gphdfs://hdfshost-1:8081/data/filename.txt')
FORMAT 'TEXT' (DELIMITER '|');
```

注意：使用gphdfs时只允许使用一个数据路径。

例7 – 带有头信息行CSV格式的多文件协议

创建一个使用file协议名为ext_expenses的可读外部表，所有文件的通配符不一样。这些文件为带有头信息行的CSV格式。

```
=# CREATE EXTERNAL TABLE ext_expenses ( name text,
    date date, amount float4, category text, desc1 text )
LOCATION ('file://filehost:5432/data/international/*',
          'file://filehost:5432/data/regional/*'
          'file://filehost:5432/data/supplement/*.csv')
FORMAT 'CSV' (HEADER);
```

例8 – 基于脚本的可读WEB外部表

创建一个在每个Segment主机上运行脚本的可读WEB外部表。

```
CREATE EXTERNAL WEB TABLE log_output (linenum int, message text)
    EXECUTE '/var/load_scripts/get_log_data.sh' ON HOST FORMAT 'TEXT' (DELIMITER '|');
```

例9 – 写到文件的可写外部表

创建一个使用gpfdist协议名为sales_out的可写外部表，将数据输出到名为sales.out的文件。该文件的格式为竖线(|)分割，空白字符当作null。

```
CREATE WRITABLE EXTERNAL TABLE sales_out (LIKE sales)
LOCATION ('gpfdist://etl1:8081/sales.out')
FORMAT 'TEXT' ( DELIMITER '|' NULL ' ')
DISTRIBUTED BY (txn_id);
```

例10 – 基于脚本的可写WEB外部表

创建一个名为campaign_out的可写WEB外部表，结束数据的是每个节点上执行名为to_adreport_etl.sh的脚本：

```
CREATE WRITABLE EXTERNAL WEB TABLE campaign_out
(LIKE campaign)
EXECUTE '/var/unload_scripts/to_adreport_etl.sh'
FORMAT 'TEXT' (DELIMITER '|');
```

使用上面定义的可写外部表卸载查询出的数据：

```
INSERT INTO campaign_out SELECT * FROM campaign WHERE customer_id=123;
```

例子11 – 带有XML转换的可读和可写外部表

目前GPDB可以使用gpfdist协议的外部表读取和写XML数据。关于设置XML转换的更多信息查看”转换XML数据”相关章节。下面是读取XML数据。

```
CREATE READABLE EXTERNAL TABLE prices_readable (LIKE prices)
LOCATION ('gpfdist://127.0.0.1:8080/data/
    \ prices.xml#transform=prices_input') FORMAT 'text' (delimiter '|')
LOG ERRORS INTO prices_errortable SEGMENT REJECT LIMIT 10;
```

下面是创建可写外部表将GPDB中的数据转换为XML。

```
CREATE WRITABLE EXTERNAL TABLE prices_readable (LIKE prices)
LOCATION ('gpfdist://127.0.0.1:8080/data/
    \ prices.xml#transform=prices_input')
FORMAT 'text' (delimiter '|');
```

从 GPDB 中卸载数据

可写外部表允许查询其他数据库表的数据并输出到文件、命名管道或其他可执行程序。可写外部表还可用于为GP并行MapReduce计算数据数据。有2种方式的可写外部表，基于文件和基于WEB。

本节讲述如何使用两种方式卸载GPDB的数据，两种方式为并行卸载(可写外部表)和非并行卸载(COPY)。本节包含如下内容：

- 定义基于文件的可写外部表
- 定义基于命令可写WEB外部表
- 使用可写外部表卸载数据
- 使用COPY卸载数据

定义基于文件的可写外部表

可写外部表使用GP并行文件服务(gpfdist)或HDFS接口(gphdfs)输出数据到文件，这与基于文件的可读外部表是类似的。

使用CREATE WRITABLE EXTERNAL TABLE命令定义外部表并指定输出文件的位置和格式。

- 使用gpfdist协议的可写外部表，GP Segment Instance将数据发送给gpfdist进程，该进程将数据写到指定名称的文件。在卸载文件的主机上必须运行着GP文件分布程序gpfdist，且所有GP Segment Instance可以访问其网络。该程序接收来自GP Segment Instance的数据并写到指定位置的文件中。若希望输出的数据分割到多个文件，可以在可写外部表的定义中执行多个gpfdist的URI选项。
- 使用可写WEB外部表输出数据到可执行程序。例如，从GPDB中卸载数据并发送给一个可执行程序，该执行程序连接到另外一个DB或者ETL并装载数据。WEB表使用EXECUTE子句在指定主机上执行SHELL命令、脚本或者可执行程序。对于可写WEB表，这些命令必须准备着接收数据输入流。

与可读外部表不同的是，可写外部表还可以有一个分布策略的选项。缺省情况下，可写外部表使用随机分布策略。如果需要被导出数的表使用了HASH分布策略，在可写外部表中定义相同的分部键可以改善卸载的性能，其可有效消除一些在网络上的数据移动。若从特定的表中卸载数据，可以使用LIKE子句拷贝原表列的定义。

例1 – GP文件服务(gpfdist)

```
=# CREATE WRITABLE EXTERNAL TABLE unload_expenses ( LIKE expenses )
LOCATION ('gpfdist://etlhost-1:8081/expenses1.out',
          'gpfdist://etlhost-2:8081/expenses2.out')
FORMAT 'TEXT' (DELIMITER ',')
DISTRIBUTED BY (exp_id);
```

例2 – Hadoop文件服务(gphdfs)

```
=# CREATE WRITABLE EXTERNAL TABLE unload_expenses ( LIKE expenses )
LOCATION ('gphdfs://hdfslhost-1:8081/path')
FORMAT 'TEXT' (DELIMITER ',')
DISTRIBUTED BY (exp_id);
```

对于gphdfs协议基于文件的可写外部表还有两个额外的限制，如下：

- 仅可以为使用了gphdfs协议的可写外部表指定一个路径。(对于可读外部表来说也只能指定一个路径)
- TEXT是仅有被支持的格式。

注意：缺省的端口号是9000。

定义基于命令的可写外部表

就像可读WEB外部表执行命令或程序，可写WEB外部表可以被定义为发送数据到一个可执行程序或脚本。在可写外部表定义中的可执行程序必须准备这接收一个输入流，其必须被放置在所有GP Segment主机的相同位置，还必须确保gpadmin用户具备可执行权限。在可写外部表的定义中指定的命令将被所有Segment Instance执行，不管该Segment Instance有没有记录需要被输出。

使用CREATE WRITABLE EXTERNAL WEB TABLE命令定义外部表并指定可执行命令或者程序，其在所有Segment Instance上运行。如果在外部表中用到了环境变量(比如$PATH)，需要注意的是，命令是从数据库执行而不是从登录的SHELL。因此，当前用户的.bashrc或者.profile文件不会被装载。不过，在外部表定义的EXECUTE子句中，可以根据需要设置环境变量，比如：

```
=# CREATE WRITABLE EXTERNAL WEB TABLE output (output text)
   EXECUTE 'export PATH=$PATH:/home/gpadmin/programs; myprogram.sh'
   FORMAT 'TEXT'
   DISTRIBUTED RANDOMLY;
```

WEB外部表和可写外部表的可执行性

外部表执行OS命令或者脚本存在一定的安全风险。有些DB管理员可能不希望他们的GPDB系统暴露这些功能。在这种情况下，可以禁止在WEB表定义中使

用EXECUTE，这通过下面的服务参数设置，该参数在Master的postgresql.conf文件：

```
gp_external_enable_exec = off
```

在EXECUTE命令中使用环境变量

如果在外部表中用到了环境变量(比如$PATH)，需要注意的是，命令是从数据库执行而不是从登录的SHELL。因此，当前用户的.bashrc或者.profile文件不会被装载。不过，在外部表定义的EXECUTE子句中，可以根据需要设置环境变量，比如：

```
CREATE EXTERNAL WEB TABLE test (column1 text) EXECUTE
    'MAKETEXT=text-text-text; export MAKETEXT; echo $MAKETEXT'
FORMAT 'TEXT';
SELECT * FROM test;
column1
----------------
text-text-text
text-text-text
(2 rows)
```

下列额外的GPDB变量在WEB外部表或者可写外部表的OS命令中可以使用。这些变量在执行命令时被设置为环境变量。这可以被外部表用来在GPDB集群中辨认不同的Segment Instance。

变量	描述
$GP_CID	执行外部表时的命令会话的计数
$GP_DATABASE	外部表定义所在的数据库
$GP_DATE	外部表执行的日期
$GP_MASTER_HOST	GP Master 的 hostname
$GP_MASTER_PORT	GP MasterInstance 的端口号
$GP_SEG_DATADIR	执行外部表命令时的当前 Segment Instance 数据路径
$GP_SEG_PG_CONF	执行外部表命令时的当前 Segment Instance 配置文件 postgresql.conf 路径
$GP_SEG_PORT	执行外部表命令时的当前 Segment Instance 端口号
$GP_SEGMENT_COUNT	GPDB 系统中 Primary Segment Instance 总数
$GP_SEGMENT_ID	执行外部表命令时的当前 Segment Instance ID(与系统日至信息表 gp_segment_configuration 中的 dbid 相同)
$GP_SESSION_ID	外部表执行时的数据库 SESSION ID 编号
$GP_SN	外部表语句执行时其查询计划中外部表扫描节点的编号
$GP_TIME	外部表执行的时间
$GP_USER	执行外部表语句的 DB User
$GP_XID	执行外部表语句的事务 ID

使用可写外部表卸载数据

由于可写外部表只允许INSERT操作，不是Owner或者SUPERUSER的用户必须被授权INSERT权限才可以使用这些表。例如：

```
GRANT INSERT ON writable_ext_table TO admin;
```

要使用可写外部表卸载数据，使用SELECT语句从源表查询数据并使用INSERT语句插入到可写外部表中。由SELECT语句查询得到的记录将输出到可写外部表中。例如：

```
INSERT INTO writable_ext_table SELECT * FROM regular_table;
```

使用 COPY 卸载数据

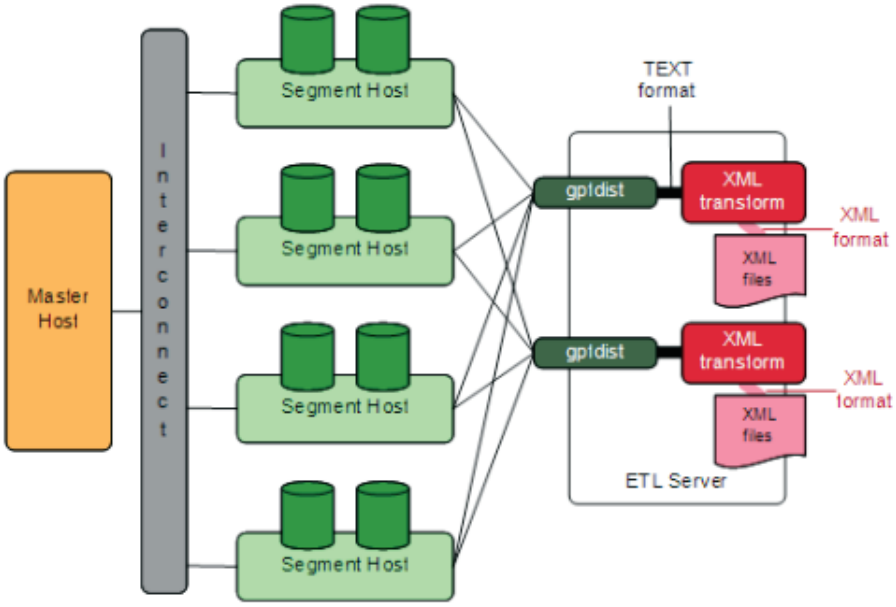
在GP Master主机上使用COPY TO语句从数据库表拷贝数据到文件(或标准输入)。COPY是非并行的，这意味着，数据是通过GP Master Instance的独立进程进行卸载。可以选择使用COPY输出一个表中的全部内容，或者通过SELECT语句选择过滤输出的内容。例如：

```
COPY (SELECT * FROM country WHERE country_name LIKE 'A%') TO
    '/home/gpadmin/a_list_countries.out';
```

译者注：通常来说，使用SELECT的COPY效率比直接使用表名COPY效率低很多，对于数据量稍大的表，可以考虑使用中间表结合COPY的方式来卸载数据。

转换 XML 数据

除了装载TEXT和CSV格式的数据，GPDB数据装载命令gpfdist还提供了从XML文件装载数据以及写数据到XML文件时对XML数据的转换功能。下图展示了gpfdist对XML的转换。



装载或抽取XML数据的最佳实践包括下列的5步过程。

- 确定转换模式
- 编写转换
- 编写gpfdist配置
- 装载数据
- 传输和存储数据

前3步组成了大部分的开发工作。最后两步很简单且属于可重复工作。

确定转换模式

在创建读取XML数据的转换前，需要一些准备工作。首先需要确定项目的目标。可能包含索引数据，分析数据，组合数据等。

接下来，检查XML文件，需注意文件的结构和节点名称。选择要导入的数据以及决定是否还有其他适当的限制。

例如下面这个prices.xml为例的XML文件，其中包含price的记录。每个price记录包含两个字段，产品编号和价格。

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<prices>
    <pricerecord>
        <itemnumber>708421</itemnumber>
        <price>19.99</price>
    </pricerecord>
    <pricerecord>
        <itemnumber>708466</itemnumber>
        <price>59.25</price>
    </pricerecord>
    <pricerecord>
        <itemnumber>711121</itemnumber>
        <price>24.99</price>
    </pricerecord>
</prices>
```

目标是将所有的数据导入到GPDB表中，itemnumber列使用integer类型，price列使用decimal类型。

编写转换

转换用以指定从数据中抽取哪些东西。可以使用任何的编程语言环境来编写。对于XML转换，GP建议选择合适的技术，如XSLT、Joost(STX)、Java、Python或Perl等。在这个price例子中，下一步是转换XML数据到简单的两列分割格式。

```
708421|19.99
708466|59.25
711121|24.99
```

下面的STX转换(input_transform.stx)完成了数据转换。

```
<?xml version="1.0"?>
<stx:transform version="1.0"
    xmlns:stx="http://stx.sourceforge.net/2002/ns"
    pass-through="none">
    <!-- declare variables -->
    <stx:variable name="itemnumber"/>
    <stx:variable name="price"/>
    <!-- match and output prices as columns delimited by | -->
    <stx:template match="/prices/pricerecord">
```

```
            <stx:process-children/>
            <stx:value-of select="$itemnumber"/>
            <stx:text>|</stx:text>
            <stx:value-of select="$price"/>
            <stx:text>
            </stx:text>
        </stx:template>
        <stx:template match="itemnumber">
            <stx:assign name="itemnumber" select="."/>
        </stx:template>
        <stx:template match="price">
            <stx:assign name="price" select="."/>
        </stx:template>
</stx:transform>
```

1. 该STX转换声明了2个临时变量itemnumber和price，以及下面的3个规则。当元素满足XPath表达式/prices/pricerecord，检查其子元素并生成输出，输出包含itemnumber变量的值，竖线(|)，price变量的值，以及行的结尾。

2. 在发现<itemnumber>元素时，将其内容存储到itemnumber变量中。

3. 在发现<price>元素时，将其内容存储到price变量中。

编写gpfdist配置

通过指定一个YAML1.1文档配置gpfdist配置。其指定了gpfdist在装载或卸载数据时选择转换的规则。

除了YAML规则外，如文档必须以三个连接号(---)开始，gpfdist配置还需要遵守如下的约束：

1. VERSION设置必须为1.0.0.1。

2. TRANSFORMATIONS设置必须指定，且必须包含一个或多个映射。

3. TRANSFORMATION中的映射必须包含下面两项

- TYPE设置，其值为input或者output
- COMMAND设置，指明如何运行转换

4. TRANSFORMATION中的每个映射可以包含CONTENT、SAFE以及STDERR设置。

下面的gpfdist配置适用prices示例。注意，每一行的缩进都是很重要的，其反映了规范的层次结构。

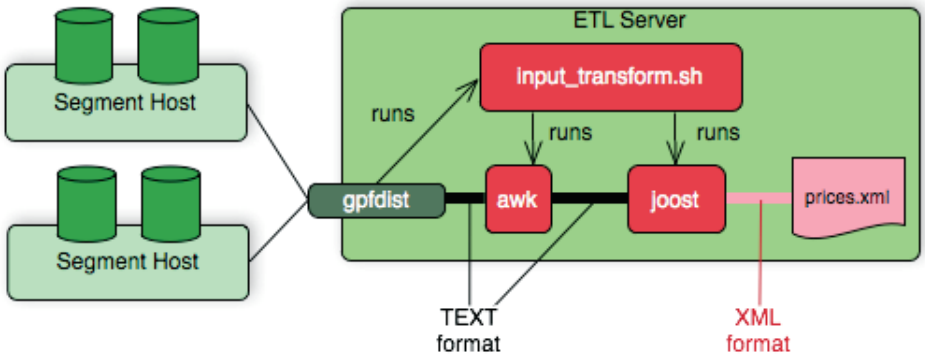
```
---
VERSION: 1.0.0.1
TRANSFORMATIONS:
    prices_input:
        TYPE: input
        COMMAND: /bin/bash input_transform.sh %filename%
```

COMMAND使用一个名为input_transform.sh的脚本，并使用占位符%filename%作为参数。在gpfdist运行prices_input转换时，会调用使用/bin/bash调用input_transform.sh，而占位符%filename%将被需要转换的输入文件路径取代。input_transform.sh脚本包

含了对STX转换的逻辑包装和调用并返回输出。

```
#!/bin/bash
# input_transform.sh - sample input transformation, # demonstrating use of Java and Joost STX to
convert XML into# text to load into Greenplum Database.
# java arguments:
# -jar joost.jar joost STX engine
# -nodecl don't generate a <?xml?> declaration
# $1 filename to process
# input_transform.stx the STX transformation
#
# the AWK step eliminates a blank line joost emits at the end
java -jar joost.jar -nodecl $1 input_transform.stx | awk 'NF>0
```

input_transform.sh文件使用Joost STX引擎并结合了AWK解释器。下图展示了gpfdist执行该转换时的处理流程。



总结一下，这个gpfdist配置的例子包含了下面3项：

- config.yaml文件定义了TRANSFORMATIONS
- input_transform.sh脚本文件，其在config.yaml文件中被用到
- input_transform.stx转换，其在input_transform.sh文件中被调用

装载数据

在适当的模式下使用SQL语句创建一张表。对于用以保存装载数据的GPDB表来说没有特殊要求。例如，下面的命令将创建一个合适的表。

```
CREATE TABLE prices (
        itemnumber integer,
        price decimal
) DISTRIBUTED BY (itemnumber);
```

传输和存储数据

有两种使用gpfdist转换数据的方法。

- GPLOAD
  在很多场景下GPLOAD实现起来更便捷，但其只支持输入转换。
- INSERT INTO SELECT FROM
  INSERT INTO SELECT FROM既支持输入转换也支持输出转换，其还可以提供更多细节。

译者注：关于XML转换的更多细节与示例这里暂时不再讲述，通常很少遇到这种情况，这种数据通常还可以使用如JAVA等编程语言预先处理为CSV格式或者TEXT格式会更便于使用。

格式化数据文件

在使用各种GP命令装载或卸载数据时，需要指定数据如何格式化。COPY、CREATE EXTERNAL TABLE和gpload有子句用以指定数据的格式。数据可以是分割文本(TEXT)或者逗号分割格式(CSV)。在GPDB读取时，外部数据必须被正确的格式化。该节讲述GPDB预期的数据文件格式。

- 格式化行
- 格式化列
- 空值字符
- 转义
- 字符编码

格式化行

GPDB预期是以LF字符(Line Feed/换行符/0x0A)、CR(Carriage Return/回车/0x0D)或者CR加LF(CR+LF/回车换行/0x0A 0x0D)作为一行的分割。LF是标准UNIX或类UNIX操作系统的标准换行标识符。其他操作系统(如Windows、Mac OS 9)可能是CR或者CR+LF。所有这些换行标识符在GPDB中都被支持作为行分隔符。

格式化列

对于TEXT文件来说缺省的列分隔符是TAB字符(0x09)，而对于CSV文件来说缺省的列分隔符是逗号(0x2C)。不过在使用COPY、CREATE EXTERNAL TABLE时或者使用gpload定义数据格式时都可以使用DELIMITER子句执行其他的单字符分隔符。分隔符只能出现在两个列之间。不要将分隔符放置在行首或者行尾。例如，使用竖线(|)作为分隔符：

```
data value 1|data value 2|data value 3
```

空值字符

NULL(空值)用于表示字段中未知的数据块。可以指定一个字符作为空值。TEXT模式的缺省字符为\N，而CSV模式为空字符。可以为COPY、CREATE EXTERNAL TABLE或gpload的数据格式定义不同的空值字符。例如，若不需要区分空值与空字符，可以指定空字符为空值。在使用GPDB装载命令时，任何匹配设定的空值的字符将被认作为空值。

译者注：假如指定NULL AS ’1’，那么在数据文件中列的值为’1’的字段将被当作NULL。

转义

数据文件中有两个保留字符，其在GPDB中有特殊含义：

- 指定的分隔符，其用于在数据文件中分割数据列。
- 换行符，用于在数据文件中标识新的一行。

如果在数据中含有这些字符，必须转义它们，这样GP才能把其作为数据而不是列分隔符或者新行。缺省情况下TEXT格式文件转义字符为反斜杠(\)，CSV格式文件的转义字符为双引号(“)。

文本格式文件的转义

缺省情况下TEXT格式文件转义字符为反斜杠(\)。如果想要指定不同的转义字符，在COPY、CREATE EXTERNAL TABLE或者gpload中使用ESCAPE子句指定其它的转义字符。在某些情况下，若转义字出现在数据中，转义字符可以转义自己。

例如，若有一个表有3个列，而要加载的数据列的情况如下所示：

```
backslash = \
vertical bar = |
exclamation point = !
```

指定的分隔符为竖线(|)，转义符为反斜杠(\)。那么数据文件应该是这种表示：

```
backslash = \\ | vertical bar = \| | exclamation point = !
```

注意当反斜杠作为数据的一部分的时候是如何使用另外一个反斜杠转义的，而竖线作为数据的一部分的时候是如何使用反斜杠转义的。

转义字符还可以用来转义8进制和16进制序列。在这种情况下，转义的值将作为等价字符被装载到GPDB。例如，要装载&符号，可以使用转义字符转义为等价的16进制(\0x26)或8进制(\046)来标识。

如果在装载的TEXT格式文件中不需要转义字符，可以使用ESCAPE子句在COPY、CREATE EXTERNAL TABLE或gpload中禁用转义字符：

```
ESCAPE 'OFF'
```

这对于数据本身包含大量反斜杠的输入数据可能是有用的(比如WEB日志数据)。

CSV格式文件的转义

缺省情况下CSV格式文件转义字符为双引号(“)。如果想要指定不同的转义字符，在COPY、CREATE EXTERNAL TABLE或者gpload中使用ESCAPE子句指定其它的转义字符。在某些情况下，若转义字出现在数据中，转义字符可以转义自己。

例如，若有一个表有3个列，而要加载的数据列的情况如下所示：

```
Free trip to A,B
5.89
Special rate "1.79"
```

指定的分隔符为逗号(,)，转义符为双引号(“)。那么数据文件应该是这种表示：

"Free trip to A,B","5.89","Special rate ""1.79"""

注意当逗号作为数据的一部分时，整个数据是被双引号包含的，而双引号作为数据的一部分的时候，即便该数据值是被双引号包含，仍需要使用双引号转义。

包含在一对双引号中的数据，其前后的空白字符将被保留作为数据的一部分：

"Free trip to A,B ","5.89 ","Special rate ""1.79"" "

注意：在CSV模式中，所有字符都是有意义的。双引号括起来的值其前后的空白或其他非分隔符的字符都会作为数据的一部分。对于一些系统产生的有补尾空白的数据文件来说，装载时可能会存在错误。如果发生了这种情况，在装载到GPDB之前，可能需要对该CSV文件做预处理以去掉这些补尾空白。

字符编码

字符编码系统包含一组编码，其对于每个给定的条目使用其他的如一串数字或者八进制串作为标记来传输或者存储数据。GPDB对字符集的支持允许以不同的字符集存储数据，包括单字节字符集如ISO 8859、多字节字符集如EUC(扩展UNIX编码)以及UTF-8等。所有支持的字符集都可以通过客户端使用，但有些不支持在服务器内部使用(作为服务器端编码)。

数据文件必须使用GPDB能够辨认的字符编码。可参考”字符集支持”相关章节查看支持的字符集。包含无效或者不支持的编码的数据文件在装载到GPDB时会发生错误。

注意：装载由Windows操作系统产生的数据文件时，在装载到GPDB之前尝试执行dos2unix系统命令删除任何Windows独有的字符。

第十三章：安装 Greenplum

由于Greenplum的性能与硬件环境有很大的相关性，所以本章从硬件环境开始讲解。本章包含如下内容：

- 硬件评估
- 操作系统安装
- 配置操作系统
- 确认操作系统
- 初始化GPDB系统

硬件评估

GP是一个分布式数据库软件，整体数据库的性能依赖于硬件的性能和各种硬件资源的均衡。如果过分突出某一方面硬件资源会造成大量的资源浪费，同时也是投资的浪费。对于OLAP应用来说，最大的瓶颈是磁盘性能(而不是磁盘容量)，因此，所有其他资源都应该围绕磁盘性能来均衡配置。这些资源包括CPU主频与核数、内存容量、网络带宽、RAID性能等。

CPU 主频与核数

CPU的主频理论上是越高越好，因为任何一个SQL语句在特定的Segment Instance上执行时只能利用一个CPU核资源进行计算。所以对于特定数量Segment Instance场景来说，CPU的主频越高，计算速度越快。不过也不必过分追求过高的主频，需衡量性能与价格和散热之间的平衡。

CPU核数最好参考主机磁盘数量，假如选择的Segment主机具有12块SAS磁盘，建议CPU核数与磁盘数的比例在12:12以上，如24:12甚至36:12，通常，考虑到生产环境需要考虑Segment Mirror和并发的因素，最佳的比例为24:12及以上。

内存容量

Segment Instance执行SQL语句时可能需要大量的内存放置计算数据，如果内存的容量无法满足计算的需要，GP可能会选择将数据缓存到本地磁盘以帮助保存数据。为每个Segment Instance配备足够的内存是必要的。通常，每个Segment Instance配备8GB以上的内存为最佳，4GB尚可接受，但最好不要低于2GB，除非这个GPDB系统只是做简单的功能测试，不然可能经常遭遇内存不足的问题。

在任务复杂度高，有并发任务执行的环境下，足够的内存容量更是必要，当然可以通过限制并发度来缓解这种压力。如果不能够限制并发度，可能会出现有查询语句失败的情况，通常从这类失败信息中可以发现内存不足的提示。

网络带宽

因为GP的架构特征，对网络层的依赖很严重，尤其在执行查询语句时出现数据

重分布的情况下特征尤为明显。假设单个Segment主机的磁盘读取能力为1000MB/S，在出现数据重分布操作时，每个Segment主机每秒有大约1000MB的数据需要被重分布到其他Segment主机。此时，如果网络吞吐能力低于1000MB/S，磁盘读取数据的速度将会被网络堵塞。若单个Segment主机的网络总带宽为400MB/S，重分布速度的理论上限即为400MB/S，由于数据重分布是有进有出，即便开启双通道模式，实际速度还会大大低于理论上限。

大多数情况下，发生数据重分布时需要分布的数据量都较大，网络将成为主要瓶颈。在允许的情况下，最好配置网络吞吐能力为磁盘读取性能的2倍左右。

注意：若最常用的场景确定不会出现数据重分布，可以考虑使用较低的网络带宽。

RAID 性能

若直接访问磁盘，将直接冲击磁盘的性能极限。大多数RAID卡都具备缓存特性。可在操作系统和磁盘之间充当缓冲作用。另外，商业的RAID卡还有一些预读和回写等高级特性，对磁盘性能有很大的提升。配置RAID时需要注意以下几点：

1. **条带宽度的设置** – 虽然OLAP应用以连续大数据读取为主要应用场景，但也有随机读取的场景存在，而一味的提高条带宽度是不正确的，千万不可听信一些所谓的专家的非专业建议。对于磁盘来说，连续读写能力是远高于随机读写能力的，但条带宽度需要选择一个适中的值，这个值可以通过测试获得，若条带宽度为64KB与1MB的连续读写速度相当，建议选择64KB的条带宽度，因为小的条带宽度随机访问能力更高。需要注意的是，GP默认的块大小为32KB，条带宽度×有效磁盘数不可小于数据库表的块大小。
2. **回写特性** – RAID的回写机制是将OS的操作缓存并批量提交到磁盘，将零散的操作连续化，从而最大化降低随机访问的消耗。

RAID卡的性能很重要，除了配置方面需要注意以外，RAID卡本身的特性在选购时特别需要留意，比如，支持的RAID类型，Cache的大小，是否支持回写与预读模式等，这些都会直接影响GP访问磁盘的效率。关于RAID卡参数的更多信息可参考其他相关文档。

总之，RAID卡的选择以完全发挥出磁盘性能为根本参考。

操作系统安装

GP支持的操作系统为：

SUSE Linux SLES 10.2 or higher
CentOS 5.0 or higher
RedHat Enterprise Linux 5.0 or higher
Oracle Unbreakable Linux 5.5
Solaris x86 v10 update 7

GP要求的文件系统为：

在SUSE或者RedHat上使用xfs(操作系统使用ext3)

在Solaris上使用zfs(操作系统使用ufs)

GP对软件工具的要求

bash shell

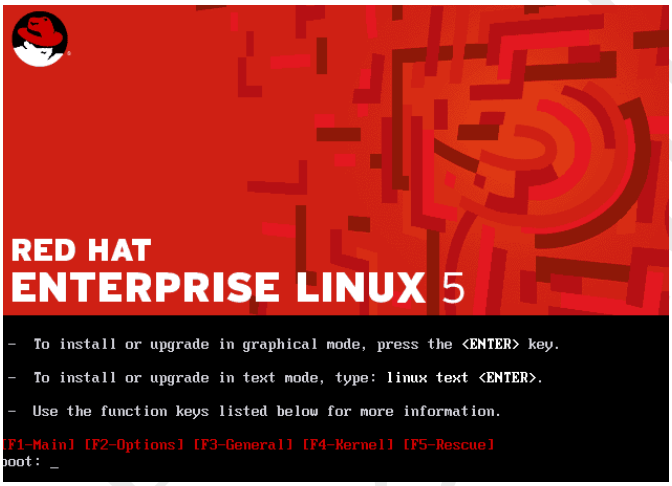
GNU tar

GNU zip

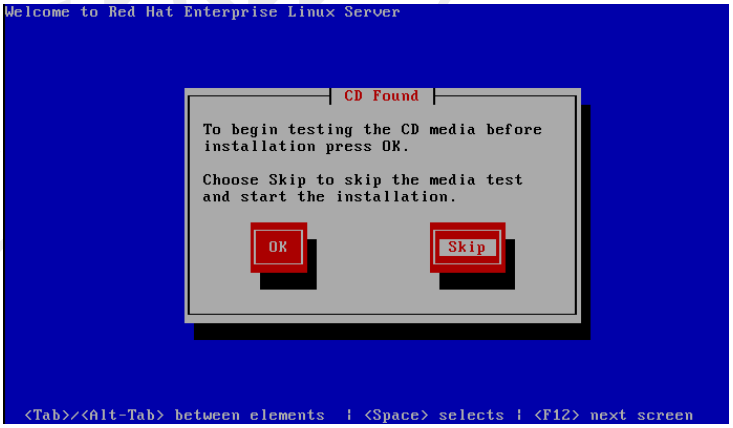
GNU readline (Solaris only)

下面以RedHat5.5为例说明建议的安装选项。

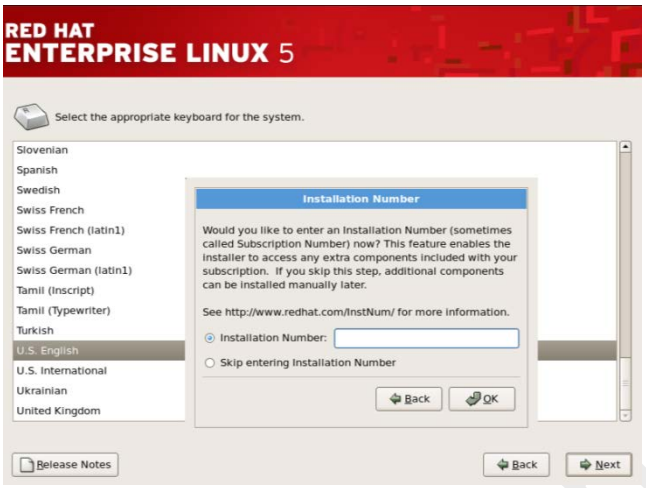
1. 出现如下界面时键入回车，进入安装界面。



2. 出现如下界面时选择【Skip】跳过磁盘检查。

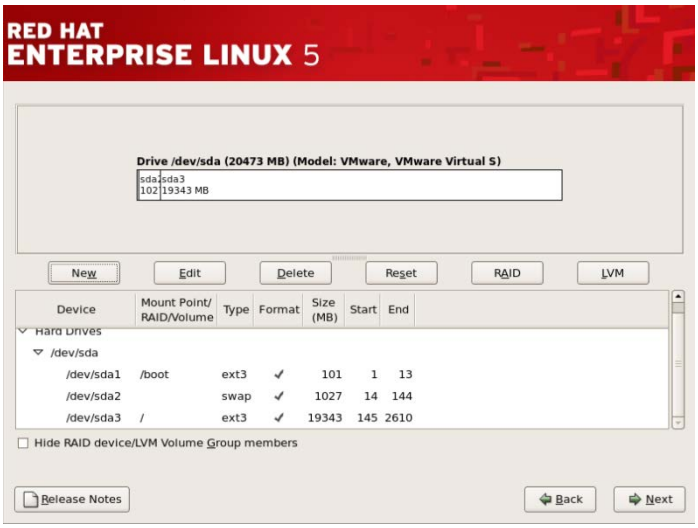


3. 一直点击【Next】直到出现如下界面，选择【Skip…】-》【OK】-》【Skip】。



4. 如出现警告，选择【Yes】，出现如下界面，如图打勾后选择【Next】。

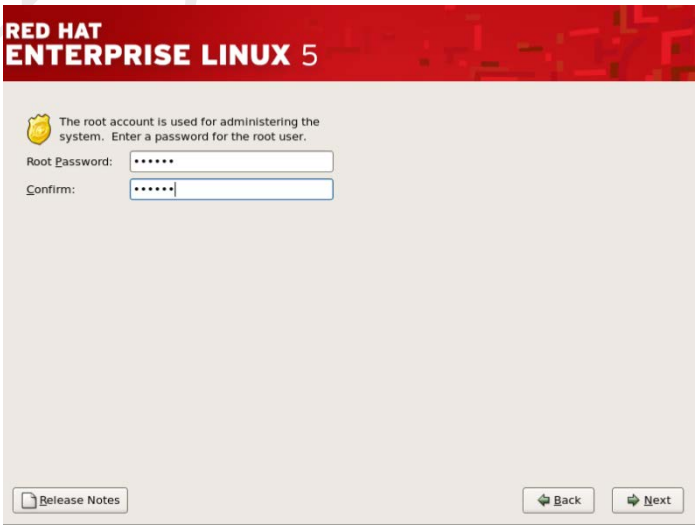
5. 如出现警告，选择【Yes】，出现如下界面，删除所有默认分区方式，按照计划新建分区，并确保SWAP大小不小于内存容量。对于数据RAID保留不动，该示例为虚拟机示例，不可作为生产机划分尺寸参考。



6. 依次选择【Next】到如下界面，设置合适的hostname，选择【Next】。

7. 出现如下界面，选择合适的时区，选择【Next】。

8. 出现如下界面，设置Root密码，选择【Next】。



9. 出现如下界面，勾选如图部分，选择【Next】。

10. 出现如下界面，按照下面的说明进行勾选，之后一直【Next】到开始安装。

--》【Desktop Environments】全置空

--》【Applications】中【Editors】和【Text-based Internet】保持不动，其他置空

--》【Development】中【Development Libraries】和【Development Tools】全选
其他置空

--》【Servers】全部置空

--》【Base System】置空【X Window System】

11. 下面是安装过程中的界面。

12. 安装完成后选择【Reboot】重启，进入【Firewall Configuration】配置，关闭【Security Level】，关闭【SELinux】。

注意：对于RedHat6.x系统来说，没有重启后的配置画面，缺省状态下SELINUX和IPTABLES都是开启状态。在登录系统后还需要进行如下操作：

- **关闭SELINUX** – 使用getenforce命令检查SELINUX状态，若结果不是”Disabled”，可使用setenforce 0命令临时关闭SELINUX。要永久关闭SELINUX，需修改/etc/selinux/config配置文件，修改配置为SELINUX=disabled。
- **关闭IPTABLES** – 使用service iptables status命令检查IPTABLES状态，若结果不是”Firewall is stopped”，可使用service iptables stop命令关闭IPTABLES。要永久关闭IPTABLES，使用chkconfig iptables off命令。

配置操作系统

GP要求在所有GPDB系统主机(Master和Segment)上设置某些OS参数。

- Linux系统设置
- Solaris系统设置
- Mac OS X系统设置

通常下面这类系统参数需要修改：

- **共享内存** – GPDB要求系统内核的内存设置符合适当的大小，否则将无法启动数据库实例。多数OS缺省安装时的共享内存设置对于GPDB来说过低。在Linux

系统，还必须禁用OOM限制器(又称OOM杀手)，不过译者使用的RedHat5.5未遭遇这类问题。

- **网络** -- 在大容量的GPDB系统中，需要一定的网络参数调优以最优化设置GP内部连接使用的网络设备。
- **用户限制** – 用户限制是对用户SHELL启动进程的可用资源限制。GPDB要求单个处理可以打开文件句柄的限制数很高。缺省的设置可能会导致一些GPDB查询失败，因为一些查询需要打开文件句柄的数量会超过系统限制。

Linux 系统设置

- 将/etc/sysctl.conf文件的内容修改为如下内容，重启生效(或执行sysctl -p生效)：

```
kernel.shmmax = 5000000000
kernel.shmmni = 4096
kernel.shmall = 40000000000
kernel.sem = 250 5120000 100 20480
#SEMMSL SEMMNS SEMOPM SEMMNI
kernel.sysrq = 1
kernel.core_uses_pid  = 1
kernel.msgmnb = 65536
kernel.msgmax = 65536
kernel.msgmni = 2048
net.ipv4.tcp_syncookies = 1
net.ipv4.ip_forward = 0
net.ipv4.conf.default.accept_source_route = 0
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_max_syn_backlog = 4096
net.ipv4.conf.default.rp_filter = 1
net.ipv4.conf.default.arp_filter = 1
net.ipv4.conf.all.arp_filter = 1
net.ipv4.ip_local_port_range = 1025 65535
net.core.netdev_max_backlog = 10000
vm.overcommit_memory = 2
```

该配置与官方文档推荐参数有调整，增大了共享内存和信号量的配置，适应更多场景。

- 在/etc/security/limits.conf配置文件末尾处增加如下内容：

```
* soft    nofile 65536
* hard    nofile 65536
* soft    nproc 131072
* hard    nproc 131072
* soft    core unlimited
```

注意：对于RedHat6.x系统，还需要将/etc/security/limits.d/90-nproc.conf文件中的1024修改为131072。

- 在Linux平台，推荐使用XFS文件系统。GP建议使用下面的挂载参数：

  rw,noatime,inode64,allocsize=16m

  比如，挂载XFS格式的设备/dev/sdb到目录/data1，/etc/fstab中的配置如下：

  /dev/sdb /data1 xfs rw,noatime,inode64,allocsize=16m 1 1

  使用XFS文件系统，需安装相应的rpm软件包，并对磁盘设备进行格式化：

  # rpm -ivh xfsprogs-2.9.4-4.el5.x86_64.rpm

  # mkfs.xfs -f /dev/sdb

- Linux磁盘I/O调度器对磁盘的访问支持不同的策略，默认的为CFQ，GP建议设置为deadline。

  要查看某驱动器的I/O调度策略，可通过如下命令查看，下面示例的为正确的配置：

  # cat /sys/block/{devname}/queue/scheduler

  noop anticipatory [deadline] cfq

  修改磁盘I/O调度策略的方法为，修改/boot/grub/menu.lst文件的启动参数，在kernel一行的最后追加”elevator=deadline”，如下为正确配置的示例：

  # grub.conf generated by anaconda

  #

  # Note that you do not have to rerun grub after making changes to this file

  # NOTICE: You have a /boot partition. This means that

  # all kernel and initrd paths are relative to /boot/, eg.

  # root (hd0,0)

  # kernel /vmlinuz-version ro root=/dev/sda3

  # initrd /initrd-version.img

  #boot=/dev/sda

  default=0

  timeout=5

  splashimage=(hd0,0)/grub/splash.xpm.gz

  hiddenmenu

  title Red Hat Enterprise Linux Server (2.6.18-194.el5)

  root (hd0,0)

  kernel /vmlinuz-2.6.18-194.el5 ro root=LABEL=/ **elevator=deadline**

  initrd /initrd-2.6.18-194.el5.img

  注意：修改该配置文件需谨慎，错误的修改会导致重启操作系统失败。

- 每个磁盘设备文件需要配置read-ahead(blockdev)值为65536。官方文档的推荐值为16384，但译者认为应该为65536更合理，该值设置的是预读扇区数，实际上预读的字节数是blockdev设置除以2，而GP缺省的blocksize为32KB，刚好与65536(32768B/32KB)对应。

  检查某块磁盘的read-ahead设置：

  # blockdev --getra devname

  例如：

  # blockdev --getra /dev/sda

  65536

  修改系统的read-ahead设置，可通过/etc/rc.d/rc.local来修改，在文件尾部追

加如下代码：

blockdev --setra 65536 /dev/sd*

如需临时修改read-ahead设置，可通过执行下面的命令来实现：

blockdev --setra bytes devname

例如：

blockdev --setra 65536 /dev/sda

- 编辑/etc/hosts文件，确保其包含所有主机名称以及所有网口信息，当然这里说的所有指的是GPDB系统涉及的所有机器。

Solaris 系统设置

- 在/etc/system文件设置如下参数：

  设置 rlim_fd_cur=65536

  设置zfs:zfs_arc_max=0x600000000

  设置cplusmp:apic_panic_on_nmi=1

  设置nopanicdebug=1

- 修改/etc/project文件中的参数。

  修改

  default:3::::

  为

  default:3:default

  project:::project.max-sem-ids=(priv,1024,deny);

  process.max-file-descriptor=(priv,252144,deny)

- 见下面的设置添加到/etc/user_attr文件：

  gpadmin::::defaultpriv=basic,dtrace_user,dtrace_proc

- 编辑/etc/hosts文件，确保其包含所有主机名称以及所有网口信息，当然这里说的所有指的是GPDB系统涉及的所有机器。

Mac OS X 系统设置

注意：Mac OSX仅仅用于开发和评估平台。不建议用于生产平台。

- 将下面的设置添加到/etc/sysctl.conf文件

  kern.sysv.shmmax=2147483648

  kern.sysv.shmmin=1

  kern.sysv.shmmni=64

  kern.sysv.shmseg=16

  kern.sysv.shmall=524288

  kern.maxfiles=65535

  kern.maxfilesperproc=65535

  net.inet.tcp.msl=60

将下面的设置添加到/etc/hostconfig文件

HOSTNAME="your_hostname"

注意：在系统设置完成后，建议重启操作系统，确保配置无误并生效。

运行 GP 安装程序

要配置GPDB系统，必须在安装的$GPHOME/bin目录下存在必要的实用程序。在选定作为Master主机的机器上使用root登录，运行GP安装程序。

在Master主机上安装GP二进制文件

1. 下载或拷贝安装文件到GPDB的Master主机。安装程序可以从EMC获得，支持的平台包括RedHat(32位和64位)、64位Solaris以及64位SuSe。
2. 解压安装文件，安装文件名中包含平台特性，如RHEL5-i386(32位RedHat)、RHEL5-x86_64(64位RedHat)、SOL-x86_64(64位Solaris)、SuSE10-x86_64(64位SuSe)。例如：

   ```
   # unzip greenplum-db-4.2.x.x-PLATFORM.zip
   ```
3. 使用bash命令启动安装程序。例如：

   ```
   # /bin/bash greenplum-db-4.2.x.x-PLATFORM.bin
   ```
4. 安装程序会提示接受GPDB许可协议。输入yes以接受许可协议。
5. 安装程序会提示输入安装路径。直接回车接受缺省安装路径(/usr/local/greenplum-db-4.2.x.x)，或者输入一个绝对安装路径。对于指定的路径，必须具有写权限。
6. 作为可选项，安装程序会提示输入一个以前安装的GPDB路径。例如：/usr/local/greenplum-db-4.2.x.x。这一步的安装会将以前安装的GPDB的附加模块(postgis、pgcrypto等)迁移到当前正在安装的版本。这一步是可选的，而且在安装之后可以使用gppkg命令结合--migrate参数重新执行。可参见相关的”gppkg”章节查看更多信息。
   直接回车跳过该步。
7. 安装程序将完成GP软件的安装，同时在带有版本信息的GP安装目录所在同级目录会创建一个名为greenplum-db的软连接。软连接有助于DB的管理维护。安装的位置被称为$GPHOME。
8. 需要更多的系统配置才能进行其他主机的GPDB安装，下一个任务就是其他主机的配置安装。

关于GPDB安装

- **greenplum_path.sh** – 该文件包含GPDB环境变量。可参考”设置GPDB环境变量”相关章节。
- **GPDB-LICENSE.txt** – GP授权许可。
- **bin** – 这个目录包含GPDB个管理命令。还包含PostgreSQL客户端和程序，大部分在GPDB中都有被用到。
- **demo** – 这个目录包含了GPDB的示例程序。
- **docs** – GPDB文档(译者不得不说，4.2.2.x版本中已经没有文档了)。
- **etc** – OpenSSL的配置示例。
- **ext** – GPDB命令使用到的依赖程序(如Python)。
- **include** – GPDB需要的C头文件。
- **lib** – GP与PostgreSQL的库文件。
- **sbin** – 支持的或内部的脚本和程序。

- **share** – GPDB的共享文件。

在所有主机上安装配置 GP

以root身份运行gpseginstall从当前主机拷贝GPDB的安装到指定的主机列表，同时，创建GP系统用户(gpadmin)，设置GP系统用户密码(缺省为changeme)，设置GPDB安装目录的所有权，在所有指定主机之间交换SSH密钥(包括root和指定的GP系统用户)。

关于gpadmin

在GPDB系统被初始化时，系统包含了预定义的SUPERUSER角色(与系统用户相关)，gpadmin。该用户拥有并管理GPDB。

注意：如果是安装单节点系统，依然可以使用gpseginstall来完成必要的系统配置任务。这时候，hostfile_exkeys将仅包含当前主机名称。

在所有指定主机上安装配置GPDB

1. 以root身份登录Master主句：

   $ su -

2. 从Master主机的GPDB安装目录加载路径文件：

   # source /usr/local/greenplum-db/greenplum_path.sh

3. 创建一个名为host_file的文件，包含GP系统的所有主机名(Master、Standby以及所有的Segment)。确保没有空行和多余的空格。例如，有1个Master，一个Standby，3个Segment，这个文件可以像这样：

   mdw

   smdw

   sdw1

   sdw2

   sdw3

注意：请确保文件中的hostname与/etc/hosts中一致。官方文档建议host_file文件包含所有对应不同网口的主机名，译者建议只包含主机名即可，因为安装操作中会涉及删除操作，为避免并行操作之间的冲突，这里不建议按照官方文档中示例的方式操作。

4. 使用刚刚创建的host_file文件运行gpdeginstall命令。使用-u和-p参数在所有主机上创建GP系统用户(gpadmin)并设置该用户的密码。例如：

   # gpseginstall -f host_file -u gpadmin -p gpadmin

安全方面的建议：

- 不要在生产环境中使用缺省密码。
- 在安装之后立即修改密码。

确认安装

要确认GP软件已经正确的安装配置，从GP的Master主机运行下面的确认步骤。如果有必要，在进行后面的任务之前解决所有的问题。

1. 在Master主机以gpadmin用户登录：
   $ su - gpadmin
2. 加载GPDB安装目录下的路径文件：
   # source /usr/local/greenplum-db/greenplum_path.sh
3. 使用gpssh命令确认是否可以在不提示输入密码的情况下登录到所有安装了GP软件的主机。使用hostfile_exkeys文件。该文件需包含所有主机的所有网口对应的主机名。例如：
   $ gpssh -f hostfile_exkeys -e ls -l $GPHOME

如果成功登录到所有主机并且未提示输入密码，安装没有问题。所有主机在安装路径显示相同的内容，且目录的所有权为gpadmin用户。

如果提示输入密码，执行下面的命令重新交换SSH密钥：

$ gpssh-exkeys -f hostfile_exkeys

安装 Oracle 兼容函数

作为可选项，许多的Oracle兼容函数在GPDb中是可用的。

在使用Oracle兼容函数之前，需要为没有数据库运行一次安装脚本：
$GPHOME/share/postgresql/contrib/orafunc.sql
例如，在testdb数据库中安装这些函数，使用命令:
$ psql –d testdb –f \$GPHOME/share/postgresql/contrib/orafunc.sql
要卸载Oracle兼容函数，视角如下脚本：
$GPHOME/share/postgresql/contrib/uninstall_orafunc.sql
注意：下面这些函数缺省可以使用，而不需要运行Oracle兼容安装：
sinh, tanh, cosh and decode
更多关于Oracle兼容函数的信息，可参考”Oracle兼容函数”相关章节。

创建数据存储区域

每个GPDB的Master和Segment实例都有一个称为data_directory位置的存储区域。这些文件系统的位置将创建用于存储Segment Instance数据的目录。Master主机需要用于存储Master数据目录的存储位置。每个Segment主机需要用于存储Primary Segment数据目录以及Mirror Segment数据目录的存储位置。

在Master主机上创建数据目录位置

Master的数据目录位置与Segment是不同的。Master不存储任何的用户数据，只是存储系统日志表和系统元数据。因此不需要为Master指定和Segment一样大小的空间。

1. 创建或者选择一个用于Master的数据存储区域。该目录需要有足够的磁盘空间并确保归属于gpadmin用户和组。例如，以root身份执行下面的命令：
   # mkdir /data/master
2. 修改目录的所有权信息。例如：
   # chown gpadmin:gpadmin /data/master
3. 使用gpssh命令在Standby上创建与和Master相同的数据存储位置。例如：

```
# source /usr/local/greenplum-db-4.2/greenplum_path.sh
# gpssh -h smdw -e 'mkdir /data/master'
# gpssh -h smdw -e 'chown gpadmin:gpadmin /data/master'
```

在所有Segment主机上创建数据目录位置

1. 以root身份登录Master主机：

   ```
   # su
   ```

2. 创建一个名为hostfile_segonly的文件。该文件包含所有Segment Host的主机名，且每个主机只出现一次。例如，有3个Segment主机：

   ```
   sdw1
   sdw2
   sdw3
   ```

3. 使用gpssh命令，在所有Segment主机上创建Primary和Mirror的数据存储位置。使用刚刚创建的hostfile_segonly文件指定Segment主机列表。例如：

   ```
   # source /usr/local/greenplum-db-4.2/greenplum_path.sh
   # gpssh -f hostfile _segonly -e 'mkdir /data/primary'
   # gpssh -f hostfile _segonly -e 'mkdir /data/mirror'
   # gpssh -f hostfile _segonly -e 'chown gpadmin /data/primary'
   # gpssh -f hostfile _segonly -e 'chown gpadmin /data/mirror'
   ```

同步系统时钟

GP建议使用NTP(网络时间协议)来同步GPDB系统中所有主机的系统时钟。

在Segment 主机上，NTP应该配置Master 主机作为主时间源，而Standby作为备选时间源。在Master和Standby上配置NTP到首选的时间源(如果没有更好的选择可以选择Master自身作为最上端的事件源)。

配置NTP

1. 在Master主机，以root登录编辑/etc/ntp.conf文件。设置server参数指向数据中心的NTP时间服务器。例如(假如10.6.220.20是数据中心NTP服务器的IP地址)：

   ```
   server 10.6.220.20
   ```

2. 在每个Segment主机，以root登录编辑/etc/ntp.conf文件。设置第一个server参数指向Master主机，第二个server参数指向Standby主机。例如：

   ```
   server mdw prefer
   server smdw
   ```

3. 在Standby主机，以root登录编辑/etc/ntp.conf文件。设置第一个server参数指向Master主机，第二个参数指向数据中心的时间服务器。例如：

   ```
   server mdw prefer
   server 10.6.220.20
   ```

4. 在Master主机，使用NTP守护进程同步所有Segment主机的系统时钟。例如，使用gpssh来完成：

   ```
   # gpssh -f hostfile_ allhosts -v -e 'ntpd'
   ```

5. 要配置集群自动同步系统时钟，应开启ntpd服务，并设置为开机时自动运行：

   ```
   # /etc/init.d/ntpd start
   ```

```
# chkconfig --level 35 ntpd on
```

检查系统环境

GP提供了下面的命令用以检查系统的配置和性能：

- gpcheck
- gpcheckperf

这些命令可以在GP安装的$GPHOME/bin目录下找到。
在初始化GPDB系统之前，应该执行下面的测试。

- 检查操作系统配置
- 检查硬件性能

检查操作系统配置

GP提供的gpcheck命令可以用来检查用作GPDB系统生产运行的主机OS配置是否符合要求。运行gpcheck命令：

1. 以gpadmin用户登录Master主机。
2. 加载GP安装目录的greenplum_path.sh文件。例如：
   ```
   $ source /usr/local/greenplum-db/greenplum_path.sh
   ```
3. 创建一个名为hostfile_gpcheck的文件，包含所有GP主机的主机名，每个名称一行。确保没有空行和多余的空白。例如：
   ```
   mdw
   smdw
   sdw1
   sdw2
   sdw3
   ```
4. 使用刚刚创建的主机名文件运行gpcheck命令。例如：
   ```
   $ gpcheck -f hostfile_gpcheck -m mdw -s smdw
   ```
5. 在gpcheck完成检查所有主机的OS参数之后，可能会给出对OS参数的修改意见，当然这些修改应该在GPDB系统初始化之前完成。

注意：刻意修改的参数不在修改考虑范围内。

检查硬件性能

GP提供的gpcheckperf命令可用来在GPDB集群主机上检查硬件和系统级别的问题。gpcheckper会在指定的主机上启动一个会话并执行下面的性能测试：

- 检查网络性能
- 检查磁盘I/O性能
- 检查内存带宽

在使用gpcheckperf之前，必须已经在所有相关需要做性能测试的主机之间建立了互信。如果还没有做这件事，可以使用gpssh-exkeys命令来建立互信。
gpcheckperf会调用gpssh和gpscp，所以这些命令必须已经存在$PATH中。

检查网络性能

要检查网络性能，结合网络测试选项执行gpcheckperf命令，网络测试选项包括：并行测试(-r N)、串行测试(-r n)、矩阵测试(-r M)。测试时运行一个网络测试程序从当前主机向远程主机传输5秒钟的数据流。缺省时，数据并行传输到每个远程主机，报告出传输的最小、最大、平均和中值速率，单位为MB/S。如果主体的传输速率低于预期(小于100MB/S)，可以使用-r n参数运行串行的网络测试以得到每个主机的结果。要运行矩阵测试，指定-r M参数，使得每个主机发送接收指定的所有其他主机的数据，这个测试可以验证网络层能否承受全矩阵工作负载。

在GPDB集群中多数系统都配置了多网口环境，每个网口有自己的子网。在测试网络性能时，独立的测试每个子网是很重要的。例如，考虑如下的多网口环境配置：

GP 主机	子网 1	子网 2
Segment1	sdw1-1	sdw1-2
Segment2	sdw2-1	sdw2-2
Segment3	sdw3-1	sdw3-2

应该为gpcheckperf创建不同的host文件用以做网络测试：

hostfile_gpchecknet_ic1	hostfile_gpchecknet_ic2
sdw1-1	sdw1-2
sdw2-1	sdw2-2
sdw3-1	sdw3-2

为每个子网运行一次gpcheckperf。例如(如果测试偶数个主机，使用并行测试)：

```
$ gpcheckperf -f hostfile_gpchecknet_ic1 -r N -d /tmp > subnet1.out
$ gpcheckperf -f hostfile_gpchecknet_ic2 -r N -d /tmp > subnet2.out
```

如果测试奇数个主机，可以使用串行测试模式(-r n)。

检查磁盘I/O和内存带宽

要测试磁盘I/O和内存带宽，使用磁盘和流测试参数(-r ds)执行gpcheckperf。磁盘测试使用dd命令(标准的UNIX命令)来测试逻辑磁盘或者文件系统的连续吞吐性能。内存测试使用流测试程序衡量内存带宽。结果以MB/S为单位给出报告。

1. 使用gpadmin用户登录Master主机。
2. 加载GP安装目录的greenplum_path.sh文件。例如：

   ```
   $ source /usr/local/greenplum-db/greenplum_path.sh
   ```

3. 创建一个名为hostfile_gpcheckperf的文件，包含所有Segment主机名，每个名称一行。不要包含Master主机名。例如：

   ```
   sdw1
   sdw2
   sdw3
   ```

4. 使用刚刚创建的主机名文件运行gpcheck命令。使用-d参数指定在每个主机上想要测试的文件系统(必须具备这些目录的写访问权限)。可能需要测试全部的Primary Instance和Mirror Instance数据目录位置。例如：

   ```
   $ gpcheckperf -f hostfile_gpcheckperf -r ds -D \
   -d /data1/primary -d /data2/primary \
   -d /data1/mirror -d /data2/mirror
   ```

5. 该测试可能需要较长的时间，因为其需要复制很大尺寸的文件。在测试完成后将可以看到磁盘写、磁盘读和流测试的结果摘要。

初始化 GPDB 系统

本节叙述如何初始化一个GPDB系统。本节的叙述假设已经根据本章之前的说明在所有相关的主机上安装了GPDB软件，并正确做好了相关的配置。
本机包括下面内容：

- 概要
- 初始化GPDB系统
- 设置GP环境变量

概要

因为GPDB是分布式的，对GPDB的初始化过程涉及到初始化一系列独立的PostgreSQL数据库实例(在GP中称为Segment Instance)。

系统中所有主机上的数据库实例都必须被初始化，这样他们才能一起作为一个整体的数据库协同工作。GP提供了自己的initdb版本，称为gpinitsystem，能够在Master和每个Segment Instance上按照正确的顺序完成初始化和启动操作。

在GPDB系统初始化并启动之后，就可以像使用常规的PostgreSQL数据库一样，通过连接到GP Master来创建和管理数据库。

初始化 GPDB 系统

对于初始化GPDB来说，下面是一些高级任务：

1. 确保已经完成了本章之前说明的安装配置任务。
2. 创建一个host文件，包含Segment的主机地址。
3. 创建GPDB系统的配置文件。
4. 缺省状态下，GPDB将使用Master主机的本地化设置进行初始化。确认本地化设置是否正确，一些本地化参数在初始化之后是不可修改的。
5. 在Master主机上运行GPDB的初始化命令。

创建一个初始化host文件

gpinitsystem命令需要一个host文件，包含每个Segment主机的地址。初始化命令根据指定的gpinitsystem_conf文件中列出的数据目录位置的数量决定Segment Instance的数量。

host文件应该仅包含Segment主机地址(不包含Master或Standby)。对于多网口主机，文件中应该为每个网口列为一行记录。

1. 以gpadmin用户登录：
   $ su - gpadmin
2. 创建名为hostfile_gpinitsystem的文件。在文件中，将所有Segment的每个网

口对应的HostName为一行的方式登记。没有多余的空行或者空格。例如，有3个Segment主机，每个主机有2个网口：

```
sdw1-1
sdw1-2
sdw2-1
sdw2-2
sdw3-1
sdw3-2
```

3. 保存关闭该文件。

注意：如果不确定主机名以及每个网口的地址，查看/etc/hosts文件获取帮助。

创建GPDB配置文件

GPDB配置文件会告诉gpinitsystem命令按照什么样的方式配置GPDB系统。在GP软件的安装目录的$GPHOME/docs/cli_help/gpconfigs/gpinitsystem_config文件可以作为配置的例子参考。

1. 以gpadmin用户登录：

```
$ su - gpadmin
```

2. 拷贝一个gpinitsystem_config示例文件作为入口。例如：

```
$ cp $GPHOME/docs/cli_help/gpconfigs/gpinitsystem_config
/home/gpadmin/gpinitsystem_config
```

3. 打开刚拷贝的文件并编辑。

DATA_DIRECTORY参数指定每个Segment主机配置多少个Instance。如果在host文件中为每个Segment主机列出了多个网口，这些Instance将平均分布到所有列出的网口上。

下面是一个gpinitsystem_config文件的例子：

```
ARRAY_NAME="EMC Greenplum DW"
SEG_PREFIX=gpseg
PORT_BASE=40000
declare -a DATA_DIRECTORY=(/data1/primary /data1/primary /data1/primary
    /data2/primary /data2/primary /data2/primary)
MASTER_HOSTNAME=mdw
MASTER_DIRECTORY=/data/master
MASTER_PORT=5432
TRUSTED SHELL=ssh
CHECK_POINT_SEGMENT=256
ENCODING=UNICODE
```

关于以上这些参数除了DATA_DIRECTORY和CHECK_POINT_SEGMENT外相对都比较容易根据英文意思理解。DATA_DIRECTORY决定了每个Segment主机上配置多少个Primary Instance，而CHECK_POINT_SEGMENT设置的是检查点段的大小，较大的检查点段可以改善大数据量装载的性能，同时会加长灾难事务恢复的时间。更多信息可参考相关文档。缺省值为8，若为保守起见，建议配置为缺省值。

4. 作为可选项，可以配置Mirror Segment Instance，取消文件中的注释并根据

环境情况配置参数。下面是gpinitsystem_config文件中可选配置Mirror的例子：

```
MIRROR_PORT_BASE=50000
REPLICATION_PORT_BASE=41000
MIRROR_REPLICATION_PORT_BASE=51000
declare -a MIRROR_DATA_DIRECTORY=(/data1/mirror /data1/mirror /data1/mirror
    /data2/mirror /data2/mirror /data2/mirror)
```

注意：可以在初始化时值配置Primary Segment Instance，而在之后使用gpaddmirrors命令部署Mirror Segment Instance。

5. 保存关闭该文件。

运行初始化命令

运行gpinitsystem命令将根据指定的配置文件创建一个GPDB系统。

1. 假设之前配置的两个配置文件hostfile_gpinitsystem和gpinitsystem_config在gpadmin用户的根目录下，使用这两个文件指定gpinitsystem命令。例如：

   ```
   $ cd ~
   $ gpinitsystem -c gpinitsystem_config -h hostfile_gpinitsystem
   ```

   对于一个完全冗余系统(带有Standby以及Spread的Mirror配置)，可使用-s和-S选项。例如：

   ```
   $ gpinitsystem -c gpinitsystem_config -h hostfile_gpinitsystem -s standby_master_hostname -S
   ```

2. 命令将自动检查安装信息，确认可以连接到每个HostName，可以访问配置文件中指定的每个目录。如果检查都通过了，将会提示确认配置信息。例如：

   ```
   => Continue with Greenplum creation? Yy/Nn
   ```

3. 输入y以开始初始化。
4. 该命令将开始安装初始化Master Instance和系统中的每个Segment Instance。每个Segment Instance的安装是串行的(原文说并行，但译者的经验证明不是并行)。Segment Instance的数量将决定了这个过程的时间长度。
5. 在成功安装初始化结束时，命令将会启动GPDB系统，并可以看到如下提示信息：

   ```
   => Greenplum Database instance successfully created.
   ```

初始化故障排除

如果在安装Instance时发生错误，整个处理将失败，并且可能会留下创建了一部分的系统。相关的错误信息和日志可以帮助诊断导致失败的原因以及在哪里发生的失败。日志文件放置在~/gpAdminLogs目录下。

根据错误发生的时间，可能在重新尝试gpinitsystem命令之前需要进行必要的清理工作。比如，一些Instance创建了而其他的失败，可能需要停掉postgres进程并删除由命令创建的数据目录(在数据存储区域内)。如果有必要，一个撤销脚本会被创建以帮助清理。

使用撤销脚本

一旦gpinitsystem命令执行失败，如过该命令留下了一个安装不完整的系统，将

会生成如下格式的撤销脚本：

~/gpAdminLogs/backout_gpinitsystem_<user>_<timestamp>

可以使用该脚本来清理已经创建的不完整的GPDB系统。该撤销脚本将会删除其创建的数据目录、postgres进程以及日志文件。执行了撤销脚本，并在解决导致gpinitsystem失败的原因之后，可以重新尝试初始化GPDB集群。

下面的例子展示了如何运行撤销脚本：

$ sh backout_gpinitsystem_gpadmin_20121106_212538

设置 GP 环境变量

必须在GPDB Master(包括Standby)设置相应的环境变量。在$GPHOME目录下的greenplum_path.sh文件提供了诸多针对GPDB的环境变量。可以在用户的启动脚本(比如.bashrc)中加载这个文件。

GPDB的管理命令还需要设置MASTER_DATA_DIRECTORY环境变量。这个变量应该指向gpinitsystem命令初始化时指定的Master数据目录位置(可参考初始化配置文件gpinitsystem_config中MASTER_DIRECTORY参数)。

1. 以gpadmin用户登录：

   $ su - gpadmin

2. 打开用户启动文件(如.bashrc)。比如：

   $ vi ~/.bashrc

3. 添加新行用以加载greenplum_path.sh文件和设置MASTER_DATA_DIRECTORY环境变量。例如：

   source /usr/local/greenplum-db/greenplum_path.sh

   export MASTER_DATA_DIRECTORY=/data/master/gpseg-1

4. 作为可选项，还可以设置一些客户端会话的环境变量，如PGPORT、PGUSER和PGDATABASE以便于使用客户端命令。例如：

   export PGPORT=5432

   export PGUSER=gpadmin

   export PGDATABASE=default_login_database_name

5. 保存关闭该文件。
6. 在编辑完用户启动文件后，加载该文件使得修改生效。例如：

   $ source ~/.bashrc

7. 如果有Standby，应该将环境变量的文件拷贝到Standby。例如：

   $ cd ~

   $ scp .bashrc standby_hostname

注意：.bashrc文件不应该产生任何的输出。如果想在用户登录的时候显示一些信息，可以使用.profile替代实现。

第十四章：启动与停止 GP

本章讲述如何启动、停止、重启GPDB系统。本章包含如下内容：

- 概述
- 启动GPDB
- 停止GPDB

概述

由于GPDB系统是分布在多个机器上，启动与停止GPDB的程序不同于常规的PostgreSQL。对于常规的PostgreSQL数据库，调用pg_ctl命令来启动、停止或重启数据库服务进程(postgres)。pg_ctl还负责重定向日志和恰当的将终端与进程组分离。

在GPDB中，每个数据库服务实例(Master和所有Segment)必须一起被启动和停止，这样它们才可以作为一个整体的数据库系统协同工作。

GP提供了类似pg_ctl功能的gpstart命令和gpstop命令，位于GPDB Master主机安装位置的$GPHOME/bin目录下。

重要提示：不要使用KILL命令来终止postgres进程。应该使用数据库命令gpkill或者pg_cancel_backend()。译者认为，有时还是不得不用，只要保证整个数据库系统被杀死后还有能力启动起来。

启动 GPDB

GP的初始化程序在成功完成初始化之后会自动启动GPDB系统。不过，有时还是需要重新启动系统，比如，要重新设置配置参数或解决一个Segment Instance失败故障。

使用gpstart命令来启动GPDB系统。这个命令会启动系统中的所有postgres数据库(包括Master和Segment的所有Instance)的监听进程。gpstart总是运行在Master主机上。

所有Instance是并行启动的(缺省的并行度为64，如果并行度超过了系统容忍度，会导致启动失败，此时需要降低并行度，参见相关章节或获取gpstart命令帮助信息)。

启动GPDB

```
$gpstart
```

重启 GPDB

gpstop命令有一个重启参数可以在成功完成关闭GPDB系统之后重新启动GPDB系统。

重启GPDB

$ gpstop -r

生效配置文件的修改

gpstop命令有一个参数用来在运行时生效修改的配置文件，包括pg_hba.conf以及postgresql.conf文件中的运行时参数，这样的生效是不需要重启服务的。需要注意的是，任何正在活动的会话不会受到这些变化的影响，而且很多服务参数要求系统完全重启(gpstop -r)才能生效。

生效配置文件的修改而不重启

$ gpstop -u

维护模式启动 Master

有些场景下需要只启动Master。这称为维护模式。在该模式下，只能使用工具模式连接到数据库，并只能对系统日志进行操作，而无法对Segment Instance上的用户数据进行操作。关于系统日志表，查看”系统日志参考”获得更多信息。

维护模式启动Master

1. 以-m参数执行gpstart命令：

   $ gpstart -m
2. 以工具模式连接Master做日志维护。例如：

   $ PGOPTIONS='-c gp_session_role=utility' psql template1
3. 在完成管理任务后，在以生产模式重新启动之前，需要以维护模式停止Master。

   $ gpstop -m

警告：错误的使用维护模式连接数据库可能会导致系统变为不一致的状态。这种操作仅应该在技术支持情况下被使用。

停止 GPDB

使用gpstop命令停止或重启GPDB系统。该命令会停止系统中的所有postgres进程(包括Master和所有Segment Instance)。gpstop总是运行在Master主机上。

所有Instance是并行停止的(缺省的并行度为64)。缺省时，系统在关闭前会等待正在活动的事务，等它们完成，如果有任何活动的客户端连接存在，系统将不会关闭。

停止GPDB

$ gpstop

快速模式停止GPDB

所有活动的事务会被中断并回滚，所有活动的客户端会话会被取消。

$ gpstop -M fast

第十五章：配置 GP 系统

有很多服务器配置参数影响着GPDB系统的行为。大多数的参数与常规的PostgreSQL一样，但还有一些GP特有的配置参数。

- 关于GP的Master参数与本地化参数
- 设置配置参数
- 配置参数种类

关于 GP 的 Master 参数与本地化参数

在多数的数据库系统中，都会有一个服务器配置文件用以配置服务器的各方面设置。在GPDB(与PostgreSQL一样)中，这个文件为postgresql.conf。该文件位于数据库实例的数据目录下。

在GPDB中，Master和每个Segment Instance都有自己的postgresql.conf文件。一些参数为本地化参数，意味着每个Segment Instance都根据自己的postgresql.conf文件来获取参数的值。对于本地化参数来说，必须在系统中的每个Instance(Master和Segment)配置。

另外一些参数是Master参数。Master参数仅需要在Master进行设置，这些参数的值在查询运行时传递(有些请款被忽略)到Segment Instance。

设置配置参数

很多的配置参数对于谁可以修改，什么地方或者什么时候可以修改是有限制的。例如，有些参数必须由GPDB的SUPERUSER才可以修改。而有些参数只能在系统级别(在postgresql.conf文件中)修改或者需要完全的重启系统才可以是修改生效。

有些参数属于回话级别的参数。会话级别的参数可以在系统级别、数据库级别、角色级别或者会话级别设置。大多数的会话参数可以在数据库User所在的会话中设置，但有些少量的参数需要SUPERUSER权限。

设置本地化配置参数

要设置本地化配置参数，必须修改所有的postgresql.conf文件才能使得修改生效。多数情况下，这意味着需要在Master和每个Segment(Primary和Mirror)上进行修改。要在GPDB系统所有的postgresql.conf文件修改参数，可以使用gpconfig命令。例如：

```
$ gpconfig -c gp_vmem_protect_limit -v 4096MB
```

然后重启GPDB以确保修改的配置生效：

```
$ gpstop –r
```

设置 Master 配置参数

如果参数是Master级别的，仅需要在GP Master Instance设置即可。如果参数是会话级别的，可以灵活的在特定的数据库、角色或者会话来设置。若某个参数在多个级别设置，越细粒度的级别优先级越高。例如，会话复写角色，角色复写数据库，数据库复写系统。

在系统级别设置参数

在Master的postgresql.conf文件设置参数以设定新的系统层面的缺省值。

1. 编辑$MASTER_DATA_DIRECTORY/postgresql.conf文件。
2. 找到需要修改的参数，去除注释(删除行首的井号#字符)，设置新的值。
3. 保存关闭该文件。
4. 对于会话级别的参数，不需要重启服务，使用下面的命令使得修改生效：

   $ gpstop -u
5. 对于需要重启服务的参数，使用下面的命令重启GPDB：

   $ gpstop -r

在数据库级别设置参数

在数据库设置会话级别参数，每个连接到该数据库的会话将使用这些参数设置。在数据库级别的设置复写系统级别的设置。使用ALTER DATABASE命令在数据库级别设置参数。例如：

```
=# ALTER DATABASE mydatabase SET search_path TO myschema;
```

在角色级别设置参数

在角色设置会话级别的参数，每个使用该角色的会话将使用这些参数设置。在角色级别的设置复写数据库级别的设置。使用ALTER ROLE命令在角色级别设置参数。例如：

```
=# ALTER ROLE bob SET search_path TO bobschema;
```

在会话级别设置参数

任何绘会话级别的参数都可以在活动的数据库会话中使用SET命令设置。设置对于当前会话后面的查询有效(直到使用RESET命令恢复缺省值)。在会话级别的设置复写在角色级别的设置。例如：

```
=# SET work_mem TO '200MB';
=# RESET work_mem;
```

查看配置参数设置

使用SQL命令SHOW查看GPDB系统使用的服务器配置参数的设置。例如，要查看所有参数：

```
$ psql -c 'SHOW ALL;'
```

使用SHOW命令只能查看Master Instance的设置。若要查看整个系统(Master和Segment)的特定参数的设置，可以使用gpconfig命令。例如：

```
$ gpconfig --show max_connections
```

配置参数种类

有很多服务器配置参数影响着GPDB系统的行为。本节讲述这些配置参数的分类。关于这些特定参数的细节，查看”服务器配置参数”相关章节。

- 连接与认证参数
- 系统资源消耗参数
- 查询调优参数
- 错误报告和日志参数
- 系统监测参数
- 运行时统计信息收集参数
- 统计信息自动收集参数
- 客户端连接缺省参数
- 锁管理参数
- 工作负载管理参数
- 外部表参数
- 旧的PostgreSQL版本兼容参数
- GP集群配置参数

连接与认证参数

这些参数控制着客户端如何连接到GPDB与认证。

连接参数

- gp_vmem_idle_resource_timeout
- listen_addresses
- max_connections
- max_prepared_transactions
- superuser_reserved_connections
- tcp_keepalives_count
- tcp_keepalives_idle
- tcp_keepalives_interval
- unix_socket_directory
- unix_socket_group
- unix_socket_permissions

安全与认证参数

- db_user_namespace
- krb_caseins_users
- krb_server_keyfile
- krb_srvname
- password_encryption
- ssl
- ssl_ciphers

系统资源消耗参数

内存消耗参数

这些参数空值着系统内存的使用。可以通过调整gp_vmem_protect_limit避免运行查询处理时Segment主机出现内存溢出。

- gp_vmem_idle_resource_timeout
- gp_vmem_protect_limit
- gp_vmem_protect_segworker_cache_limit
- max_appendonly_tables
- max_prepared_transactions
- max_stack_depth
- shared_buffers
- temp_buffers

自由空间映射参数

这些参数控制着自由空间映射的尺寸，自由空间映射存储着过期的记录。由VACUUM命令回收的磁盘空间存储在自由空间映射中。

- max_fsm_pages
- max_fsm_relations

操作系统资源参数

- max_files_per_process
- shared_preload_libraries

基于成本的延迟回收参数

标准PostgreSQL有一些参与用于配置VACUUM和ANALYZE的执行成本。这些特性的目的是允许管理员在数据库并发活动时降低相关命令的I/O冲击。当这些操作消耗的累计I/O成本超过了设定的限制，操作的执行将会被延迟。然后重置计数器并重新执行。

警告：基于成本的回收延迟不建议在GPDB中使用，因为在Segment Instance之间是异步运行的。回收成本的限制和延迟是基于Segment本地调用，而不是从GP集群全局的状态来考虑。

- vacuum_cost_delay
- vacuum_cost_limit
- vacuum_cost_page_dirty
- vacuum_cost_page_hit
- vacuum_cost_page_miss

查询调优参数

查询计划控制参数

下面的参数控制着查询规划器选择什么样的计划操作类型。启用或禁用特定的计划操作是一种强制规划器选择不同计划的方式。这对于使用不同的计划类型来测试查询以选择最优性能是很有帮助的。

- enable_bitmapscan

- enable_groupagg
- enable_hashagg
- enable_hashjoin
- enable_indexscan
- enable_mergejoin
- enable_nestloop
- enable_seqscan
- enable_sort
- enable_tidscan
- gp_enable_adaptive_nestloop
- gp_enable_agg_distinct
- gp_enable_agg_distinct_pruning
- gp_enable_direct_dispatch
- gp_enable_fallback_plan
- gp_enable_fast_sri
- gp_enable_groupext_distinct_gather
- gp_enable_groupext_distinct_pruning
- gp_enable_multiphase_agg
- gp_enable_predicate_propagation
- gp_enable_preunique
- gp_enable_sequential_window_plans
- gp_enable_sort_distinct
- gp_enable_sort_limit

查询规划成本估算

警告：GP建议不要调整这些查询成本估算参数。这些参数已经调整到适合GPDB硬件配置和工作负载特征的状态。这些参数是相关的。修改一个而不修改其他的参数可能会产生相反的性能影响。

- cpu_index_tuple_cost
- cpu_operator_cost
- cpu_tuple_cost
- cursor_tuple_fraction
- effective_cache_size
- gp_motion_cost_per_row
- gp_segments_for_planner
- random_page_cost
- seq_page_cost

数据库统计抽样参数

这些参数调整ANALYZE操作数据抽样的数量。调整这些参数会影响系统级别的统计信息收集。可以在特定的表和列配置统计信息手机参数，使用ALTER TABLE SET STATISTICS命令。

- default_statistics_target
- gp_analyze_relative_error

排序操作配置参数

- gp_enable_sort_distinct

- gp_enable_sort_limit

聚合操作配置参数

- gp_enable_agg_distinct
- gp_enable_agg_distinct_pruning
- gp_enable_multiphase_agg
- gp_enable_preunique
- gp_enable_groupext_distinct_gather
- gp_enable_groupext_distinct_pruning
- gp_workfile_compress_algorithm

关联操作配置参数

- join_collapse_limit
- gp_adjust_selectivity_for_outerjoins
- gp_hashjoin_tuples_per_bucket
- gp_statistics_use_fkeys
- gp_workfile_compress_algorithm

其他查计划配置参数

- from_collapse_limit
- gp_enable_predicate_propagation
- gp_statistics_pullup_from_child_partition

错误报告和日志参数

日志滚动

- log_rotation_age
- log_rotation_size
- log_truncate_on_rotation

日志级别

- client_min_messages
- log_error_verbosity
- log_min_duration_statement
- log_min_error_statement
- log_min_messages

日志内容

- debug_pretty_print
- debug_print_parse
- debug_print_plan
- debug_print_prelim_plan
- debug_print_rewritten
- debug_print_slice_table
- log_autostats
- log_connections
- log_disconnections
- log_dispatch_stats
- log_duration

- log_executor_stats
- log_hostname
- log_parser_stats
- log_planner_stats
- log_statement
- log_statement_stats
- log_timezone
- gp_debug_linger
- gp_log_format
- gp_max_csv_line_length
- gp_reraise_signal

系统监测参数

SNMP提醒

下面的参数用来在GPDB系统发生事件时发送SNMP通知。

- gp_snmp_community
- gp_snmp_monitor_address
- gp_snmp_use_inform_or_trap

邮件提醒

下面的参数用来配置系统发送致命错误事件的邮件提醒。比如，Segment失败或者服务崩溃重启。

- gp_email_from
- gp_email_smtp_password
- gp_email_smtp_server
- gp_email_smtp_userid
- gp_email_to

GP命令中心(CC)代理

下面的参数用以为GPCC配置数据手机代理。

- gp_enable_gpperfmon
- gp_gpperfmon_send_interval
- gpperfmon_port

运行时统计信息收集参数

这些参数可以空值PostgreSQL服务的统计信息收集功能。当统计信息收集被开启，产生的数据可以通过pg_stat和pg_statio系统日志视图获取。

- stats_queue_level
- track_activities
- track_counts
- update_process_title

统计信息自动收集参数

当自动统计信息收集被开启，在INSERT、UPDAT、DELETE、COPY和CREATE TABLE…AS SELECT语句被执行时，如果影响是数量达到指定门槛(on_change)或者现有系统中没有统计信息(on_no_stats)，ANALYZE会自动被运行。要开启这个功能，在GP Master的postgresql.conf文件中设置下面的服务器配置参数，并需要重启GPDB系统：

- gp_autostats_mode
- log_autostats

警告：根据特定的数据库操作，自动统计信息手机可能会产生负面的性能影响。需谨慎评估缺省设置on_no_stats是否适合当前的系统。译者认为，很多时候是真的不适合，如果可以人为在代码中控制，那是最理想的情况。

客户端连接缺省参数

语句行为参数

- check_function_bodies
- default_tablespace
- default_transaction_isolation
- default_transaction_read_only
- search_path
- statement_timeout
- vacuum_freeze_min_age

本地化和格式化参数

- DateStyle
- extra_float_digits
- IntervalStyle
- lc_collate
- lc_ctype
- lc_messages
- lc_monetary
- lc_numeric
- lc_time
- TimeZone

其他客户端缺省参数

- dynamic_library_path
- explain_pretty_print
- local_preload_libraries

锁管理参数

- deadlock_timeout
- max_locks_per_transaction

工作负载管理参数

下面的参数用来配置GPDB工作负载管理特征(资源队列)，查询优先级，内存利用和并发空值。

- gp_resqueue_priority
- gp_resqueue_priority_cpucores_per_segment
- gp_resqueue_priority_sweeper_interval
- gp_vmem_idle_resource_timeout
- gp_vmem_protect_limit
- gp_vmem_protect_segworker_cache_limit
- max_resource_queues
- max_resource_portals_per_transaction
- resource_cleanup_gangs_on_wait
- resource_select_only
- stats_queue_level

外部表参数

下面的参数用来配置GPDB的外部表特征。

- gp_external_enable_exec
- gp_external_grant_privileges
- gp_external_max_segs
- gp_reject_percent_threshold

只追加表参数

下面的参数用来配置GPDB的只追加表特征。

- max_appendonly_tables

数据库和表空间/文件空间参数

下面的参数用来配置系统中数据库、表空间和文件空间的最大值限制。

- gp_max_tablespaces
- gp_max_filespaces
- gp_max_databases

旧的 PostgreSQL 版本兼容参数

下面是与旧的PostgreSQL版本兼容性参数。通常，不需要在GPDB中修改这些参数。

- add_missing_from
- array_nulls
- backslash_quote

- escape_string_warning
- regex_flavor
- standard_conforming_strings
- transform_null_equals

GP 集群配置参数

下面的是控制GPDB集群和各组建(Segment、Master分布式事务管理以及互联网络)的配置参数。

互联网络配置参数

- gp_interconnect_hash_multiplier
- gp_interconnect_queue_depth
- gp_interconnect_setup_timeout
- gp_interconnect_type
- gp_max_packet_size

分派配置参数

- gp_cached_segworkers_threshold
- gp_connections_per_thread
- gp_enable_direct_dispatch
- gp_segment_connect_timeout
- gp_set_proc_affinity

故障操作参数

- gp_set_read_only
- gp_fts_probe_interval
- gp_fts_probe_threadcount

分布式事务管理参数

- gp_max_local_distributed_cache

只读参数

- gp_command_count
- gp_content
- gp_dbid
- gp_num_contents_in_cluster
- gp_role
- gp_session_id

第十六章：开启高可用特性

本章讲述GPDB的高可用特性，解释Segment或Master的恢复。本章包含如下内容：

- GPDB的高可用概述
- 开启GPDB的Mirror
- 获知Segment何时失败
- 恢复失败的Segment
- 恢复失败的Master

GPDB 的高可用概述

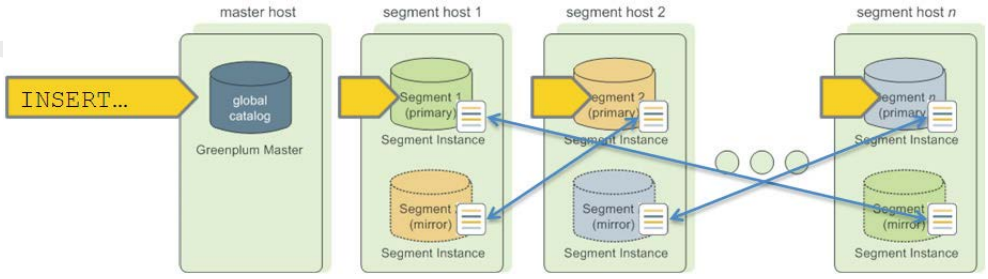
GP提供了几种可选特性来确保最大可能的工作连续性和系统的高可用。本节包含下面内容的概述：

- Segment Mirror概述
- Master Mirror概述
- 故障检测与恢复概述

Segment Mirror 概述

Mirror允许数据库查询在Primary不可用时故障切换到备份Segment Instance。要配置Mirror，GPDB系统中必须有足够多的节点才能保证Mirror总是位于不同于对应的Primary所在的主机。只有Mirror与Primary位于不同主机时才能充分保证有主机不可用时，数据库的中可用的Instance仍可以保证数据的完整性。然而，一定要配置Mirror与Primary重叠也不是不可以的，只是这样就失去了高可用特征。

Mirror位于其Primary主机之外。在数据库运行时，只有Primary是活动的。Primary的变化通过文件块同步程序复制到其Mirror。除非Primary出现故障，Mirror总是处于非活动状态 – 仅有同步程序在运行。



在Primary失效时，文件同步程序会停止，Mirror会自动唤醒替代Primary处于活动状态。所有的数据库操作将继续使用Mirror。在Mirror活动期间，所有对数据库的修改将被记录日志。此时的系统状态为修改跟踪(Change Tracking)模式。当失效的Segment被唤醒为在线状态，管理员可以使用恢复程序将其恢复到活动状态(或原始状态)。恢复程序只同步拷贝Primary失效期间Mirror发生变化的部分。此时的系统状态为重新同步(Resynchronizing)状态。一旦所有的Mirror与它们的Primary都处于同步(synchronized)状态，系统状态将变为同步(synchronized)状态。

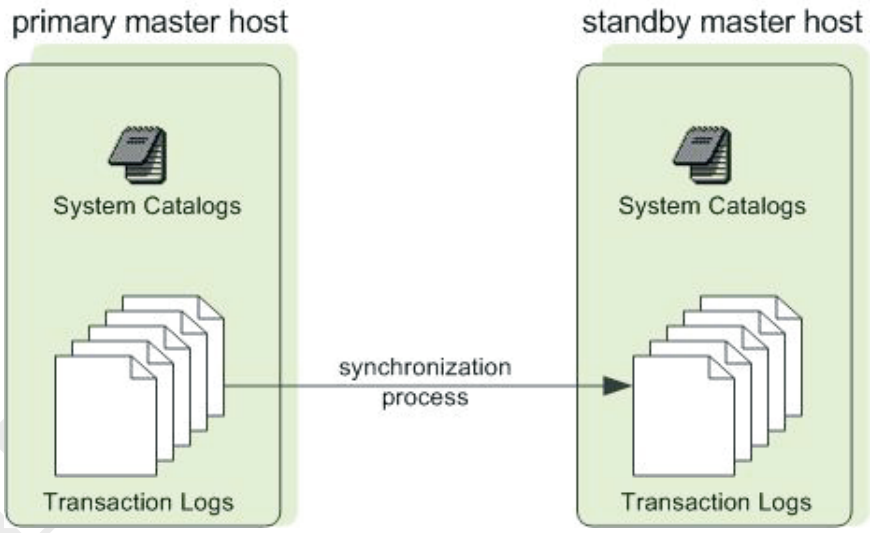
Master Mirror 概述

作为可选项，还可以在Master Instance主机之外部署一个Master镜像。后备Master(Standby)在Master不可用时作为”热备”主机服务。

Standby与Master之间通过同步程序(gpsyncagent)保持实时同步，同步程序运行在Standby上并在Master和Standby之间保持数据同步。除非Master出现故障，Standby总是处于非活动状态 – 仅有同步程序在运行。

在Master失效时，同步程序会停止，Standby可以被在本机被激活。激活Standby时，同步日志被用来恢复Master最后一次事务成功提交时的状态。激活的Standby的职能为GPDB Master，取代之前的Master，使用相同的端口接受连接。

由于Master不含有任何用户数据，仅有系统日志表需要在Primary和Standby之间被同步。这些表不会被频繁更新，不过，一旦更新，变化会自动复制到Standby，保证总是与Primary之间保持同步。



故障检测与恢复概述

故障检测由GPDB服务(postgres)的子进程ftsprobe来处理。该故障检测进程监测GP集群，按照可配置的间隔周期扫描所有Segment实例和数据库进程。

每当故障检测进程无法连接到某个Segment Instance，会将该Segment Instance在GPDB系统日志中标记为失败(down)状态。一旦某个Instance失败了，直到管理员执行恢复程序来恢复到在线为止，一直保持未运行状态。

在GPDB系统中的Mirror启用时，当Primary不可用时系统会自动切换到Mirror。只要GPDB系统中剩下的可用Instance能够确保全部的数据完整，系统将继续处于可用状态。

要恢复系统中失败的Instance，GPDB管理员需运行恢复程序(gprecoverseg)。该程序定位失败的Instance，检查它们是否可用，与当前活动的Instance比较事务状态以确定掉线期间发生的变化。然后与活动的Instance同步变化的数据库文件并唤醒回在线状态。恢复程序需要在GPDB系统启动运行时执行。

如果没有启用Mirror，在有Instance不可用时，系统会自动停止服务。在可以继续运行之前，必须手动恢复所有失败的Instance。

开启 GPDB 的 Mirror

GPDB系统的Mirror既可以在初始化(gpinitsystem)时配置，也可以在现有的系统上重新配置(gpaddmirrors和gpinitstandby)。本节讲述对于初始化时没有配置Mirror的系统如何添加Mirror。

有两种可以选择配置的Mirror类型：

- 启用Segment Mirror
- 启用Master Mirror

启用 Segment Mirror

Segment Mirror允许在Primary Segment不可用时故障切换到备份的Segment。要配置Mirror，在GPDB系统中必须有足够数量的节点以保证Mirror Segment总是位于不同于Primary Segment所在的主机。当然，还可以选择与Primary Host完全不同的主机来配置Mirror Segment。

在现有系统添加Segment Mirror(与Primary Segment相同的主机)

1. 在所有Segment主机上分配用以存储Mirror的数据存储区域。这个区域必须与Primary Segment位于不同的文件系统位置。
2. 必须确保所有Segment主机之间已经建立了互信，参见gpssh-exkeys命令。
3. 运行gpaddmirrors命令启用GPDB系统的Mirror。例如(-p参数加到Primary Segment Instance端口数字之上作为Mirror Instance计算Mirror Instance端口的基础数字)：

   $ gpaddmirrors -p 10000

在现有系统添加Segment Mirror(与Primary Segment不同的主机)

1. 确保GPDB软件已经在所有主机上安装。
2. 在所有Segment主机上分配用以存储Mirror的数据存储区域。
3. 必须确保所有Segment主机之间已经建立了互信，参见gpssh-exkeys命令。
4. 创建配置文件，该文件包含所有主机名称，端口，数据目录。可以使用下面的方式来创建参考配置文件：

   $ gpaddmirrors -o filename

   Mirror配置文件的格式为：

   filespaceOrder=[filespace1_fsname[:filespace2_fsname:...]

   mirror[content]=content:address:port:mir_replication_port:pri_replication_port:fselocation[:fselocation:...]

例如(2个Segment主机，每个Segment主机2个Segment Instance，且没有除了缺省文件系统pg_default之外的其他文件系统)：

```
filespaceOrder=
mirror0=0:sdw1:sdw1-1:52001:53001:54001:/gpdata/mir1/gp0
mirror1=1:sdw1:sdw1-2:52002:53002:54002:/gpdata/mir1/gp1
mirror2=2:sdw2:sdw2-1:52001:53001:54001:/gpdata/mir1/gp2
mirror3=3:sdw2:sdw2-2:52002:53002:54002:/gpdata/mir1/gp3
```

注意：Mirror配置文件中最重要的配置信息格式为第2行开始的内容：

- content对应Mirror的序号，该序号与Primary的序号匹配
- address对应Mirror所在的主机名，其决定了Mirror位于哪里
- port为Mirror运行时的监听端口，其功能与Primary活动时的端口类似
- mir_replication_port为Mirror的同步程序端口
- pri_replication_port为Primary的同步程序端口
- fselocation为Mirror所在文件系统的数据存储目录

如果手动修改程序生成的配置文件，务必保证同一主机端口不可存在冲突，否则在后面的操作会在一开始即提示失败，失败信息为端口冲突。

5. 运行gpaddmirrors命令启用GPDB系统的Mirror(-i参数指定刚刚创建的Mirror配置文件)：

```
$ gpaddmirrors -i mirror_config_file
```

启用 Master Mirror

GPDB系统的Standby既可以在初始化(gpinitsystem)时配置，也可以在现有的系统上配置(gpinitstandby)。假设现有系统初始化时未设置Standby，本节讲述如何为之增加Standby。

为已有系统增加Standby

1. 确保Standby主机已经正确的安装配置(gpadmin系统用户已创建，GPDB二进制文件已安装，环境变量已设置，互信已建，数据目录已建)。
2. 在当前活动的Master主机上运行gpinitstandby命令已添加GPDB系统的Standby。例如(-s参数指定Standby的主机名)：

```
$ gpinitstandby -s smdw
```

3. 关于切换到Standby，参照”恢复失败的Master”相关章节。

检查日志同步程序的状态

如果Standby上的同步程序(gpsyncagent)失败了，这对于系统的用户可能是不明显的。gp_master_mirroring日志表是GPDB管理员用以检查Standby目前是否处于同步状态的地方。例如：

```
$ psql dbname -c 'SELECT * FROM gp_master_mirroring;'
```

如果结果表明Standby的状态是”Not Synchronized”，检查detail_state 和error_message列以确定错误的原因。

恢复一个已经不同步的Standby：

```
$ gpinitstandby -s standby_master_hostname -n
```

获知 Segment 何时失败

如果开启了Mirror，在Primary Instance失败时GPDB将自动失败切换到Mirror Instance。只要数据每个部分都有一个Instance处于活动状态，通常Segment Instance失败对于用户来说不太明显。如果在失败发生时，一个事务正在进行，该事务会被回滚，然后会在重新配置了Segment之后自动重启该事务。

如果由于某个Segment Instance的失败导致了整个GPDB系统变得不可运行(数据完整性被破坏)，在尝试连接到数据库时会得到错误。返回到客户端程序的错误信息可能会给出一些失败的提示。例如：

```
ERROR: All segment databases are unavailable
```

启用警告和通知

GPDB管理员可以启用发生如Segment失效之类的系统事件时的email或者SNMP警告。查看”启用系统警告和通知”相关章节获取更多信息。

检查失败的 Segment

在启用Mirror情况下，可能出现Segment失败时，系统不会中断服务，而且没有明确提示。检查系统状态的一种方法就是使用gpstate命令。该命令会列出GPDB系统中每个独立组件(Primary Instance、Mirror Instance、Master、Standby)的状态。

检查失败的Segment

1. 在Master主机，使用-e参数执行gpstate命令。这将显示任何出错状态的Instance：
   ```
   $ gpstate -e
   ```
2. 处于修改跟踪(Change Tracking)状态表明对应的Mirror已经失败。
   如果某个Instance不在其初始的角色，意味着其当前的运行状态与系统初始化时的设置不同。这意味着系统处于非平衡状态，一个Segment主机可能会有较多的活动Instance，消耗更高的系统资源。参考”恢复所有Instance到初始角色”相关章节。
3. 要获得失败Instance的详细信息，可查看系统日志表gp_segment_configuration。例如：
   ```
   $ psql -c "SELECT * FROM gp_segment_configuration WHERE status='d';"
   ```
4. 对于失败的Instance，注意其主机、端口、初始角色和数据目录。这些将有助于确定Instance所在的主机和位置，有助于诊断故障。
5. 查看Primary Instance与Mirror Instance之间的映射关系，执行命令：
   ```
   $ gpstate –m
   ```

检查日志文件

日志文件对于确定出错的原因可以提供更多的信息。Master和每个Segment Instance都有自己的日志文件，位于数据目录下的pg_log中。Master的日志文件包含着最多的信息，应该总是首先检查Master日志文件。可以使用gplogfilter命令检查GPDB日志文件。要检查Segment的日志文件，可以使用gpssh在Segment主机上运行gplogfilter命令。

检查日志文件

1. 检查Master日志文件WARNING、ERROR、FATAL或PANIC级别的日志信息：

   $ gplogfilter -t

2. 使用gpssh检查Segment Instance日志文件WARNING、ERROR、FATAL或PANIC级别的日志信息：

   $ gpssh -f seg_hosts_file -e 'source /usr/local/greenplum-db/greenplum_path.sh ; gplogfilter -t /data1/primary/*/pg_log/gpdb*.log' > seglog.out

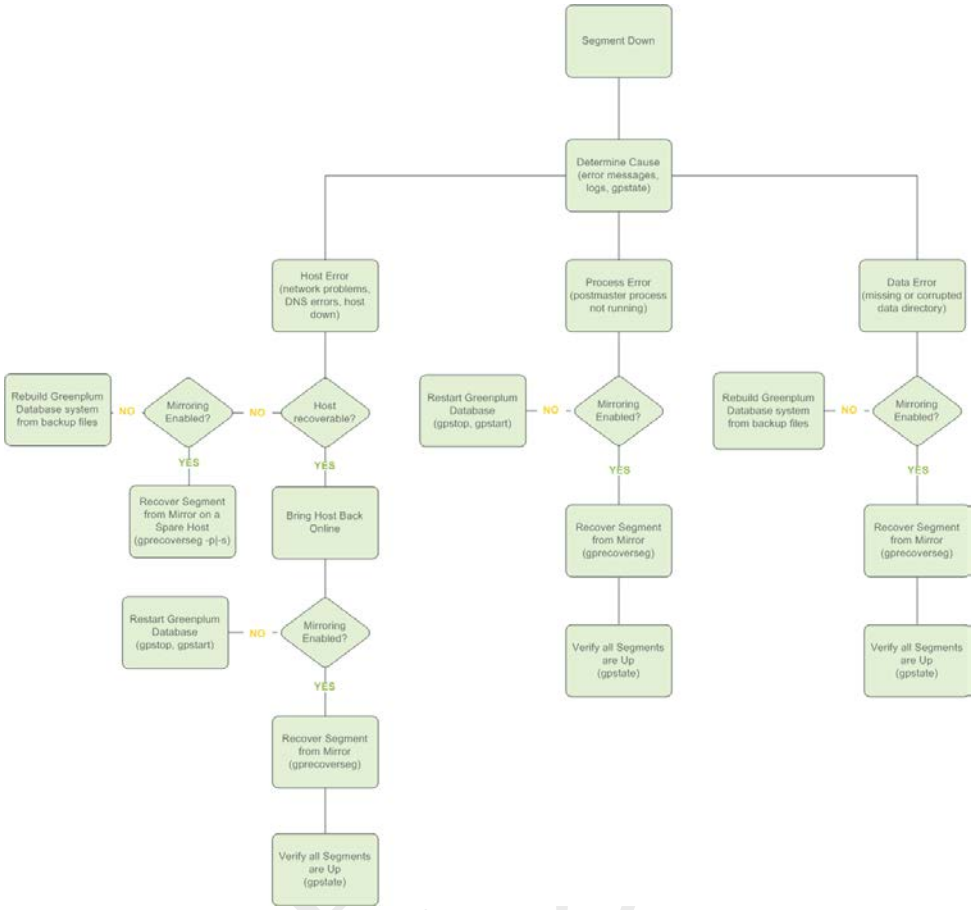
恢复失败的 Segment

当Master无法连接到Segment Instance，会在GPDB系统日志中将该Instance标记为失败状态。直到采取措施恢复失败Instance到在线状态前，该节点一直保持未运行状态。

恢复失败Instance或者主机的程序依赖于失败的原因和是否启用了Mirror。Instance变得不可用有多种因素，例如：

- 主机无法访问(网络错粗、硬件故障)。
- Instance未运行(无postgres数据路监听进程)。
- Instance的数据目录损坏或丢失(数据无法访问、文件系统损坏、磁盘故障)。

对于这些情况的失败，下图展示了解决步骤：



从 Segment 失败中恢复

若整个Segment主机失败，将会导致多个Instance失败，该主机上的所有Primary和Mirror都会被标记为失败并一直为未运行状态。如果GPDB系统没有配置Mirror，在任何Instance失败时，系统都将会自动停止并不可操作。

恢复有Mirror的系统

1. 第一步是确保从Master主机可以连通该Segment主机。例如：
   $ ping failed_seg_host_address
2. 找到Master无法连接到Segment主机的故障并排除。例如，Segment主机可能需要重启或更换。
3. 主机正常启动之后，先确认可以连接，然后从Master主机执行gprecoverseg命令恢复失败的Instance。例如(从Master主机执行)：
   $ gprecoverseg
4. 恢复进程会唤醒失败的Instance并确定需要被同步的变化文件。在此期间不能取消gprecoverseg进程，耐心的等待其结束。在此过程中，数据库暂时终止写操作。
5. 在gprecoverseg完成之后，系统变为重新同步(Resynchronizing)状态，开始拷贝覆盖变化的文件。此进程是后台运行的，期间系统处于可用状态且接受数据库请求。
6. 当重新同步进程完成，系统将重新变为已同步(Synchronized)状态。运行

gpstate命令确认同步进程的状态：

$ gpstate -m

恢复所有Instance到原有角色

当Primary Instance失效时，Mirror Instance被激活并变为Primary Instance。在执行gprecoverseg之后，当前作为Primary Instance活动的仍然是失败时激活的Mirror Instance。Primary Instance并没有恢复到系统初始化时的角色。这会导致系统处于非平衡状态，一个Segment主机可能会有较多的活动Instance，消耗更高的系统资源。要检查非平衡Instance，执行：

$ gpstate -e

在恢复失败的Instance之后再试图将其恢复到系统平衡状态。要恢复到平衡状态，所有的Instance必须启动且完全处于已同步状态。在恢复平衡期间的数据库会话，其正在处理的查询会被取消并回滚。

1. 运行gpstate -m命令确保所有的Mirror处于已同步状态。

   $ gpstate -m

2. 如果任何的Mirror处于重新同步(Resynchronizing)状态，请等待知道它们完成。
3. 使用-r参数运行gprecoverseg命令恢复Instance到原始角色。

   $ gprecoverseg -r

4. 在恢复平衡之后，运行gpstate –e确认所有Instance已经恢复到原始角色。

   $ gpstate -e

从双失效恢复

双失效指的是Primary与对应的Mirror一起失效的情况。这种情况极少发生，比如两个不同的Segment主机同时失效。在出现双失效时，GPDB会停止服务。想要从双失效恢复，执行如下步骤：

1. 重启GPDB：

   $ gpstop -r

2. 在系统重启之后，运行gprecoverseg:

   $ gprecoverseg

3. 在恢复失败的Instance之后，使用gpstate确认Mirror的状态：

   $ gpstate -m

4. 如果仍然有Instance处于修改跟踪(Change Tracking)状态，运行完全拷贝覆盖的恢复

   $ gprecoverseg -F

注意：使用-F参数会将失败的Instance数据完全清除并从配对的Instance完全复制所有数据。出现双失败的情况时上述步骤不确保一定可以恢复回来，如果一致性数据遭到破坏或者事务ID发生紊乱，在尝试恢复之前还需要其他修复工作。

恢复无Mirror的系统

1. 第一步是确保从Master主机可以连通该Segment主机。例如：

   $ ping failed_seg_host_address

2. 找到Master无法连接到Segment主机的故障并排除。例如，Segment主机可能需要重启或更换。

3. 在主机启动之后，确认可以连通该Segment主机，尝试重启GPDB

   $ gpstop -r

4. 运行gpstate确认是否所有Segment Instance已经启动：

   $ gpstate

注意：如果Segment Host无法恢复，可能需要重新创建GPDB系统并从备份文件恢复系统数据。当然在重新创建GPDB系统之前还有一些手段用来尝试恢复系统，比如诸多PostgreSQL方面的手段，理论上只要数据未丢失，都是有办法恢复的。

当某个Segment主机无法恢复

如果某个主机再也无法运行(比如硬件失败)，可能需要恢复到一个备用的硬件资源上。如果启用了Mirror，可以使用gprecoverseg从Mirror拷贝恢复到新的主机上。例如：

```
$ gprecoverseg -i recover_config_file
```

这里使用的recover_config_file文件格式为：

```
filespaceOrder=[filespace1_name[:filespace2_name:...]
failed_host_address:port:fselocation
[recovery_host_address:port:replication_port:fselocation[:fselocation:...]]
```

例如(如要恢复到一个不同的主机，且没有除了缺省文件系统pg_default之外的其他文件系统)：

```
filespaceOrder=
sdw5-2:50002:/gpdata/gpseg2 sdw9-2:50002:53002:/gpdata/gpseg2
```

系统日志表gp_segment_configuration and pg_filespace_entry可以帮助确定当前Segment Instance的配置，依此可以作为恢复配置文件的参考。例如，运行下面的查询：

```
=# SELECT dbid, content, hostname, address, port,
    replication_port, fselocation as datadir
    FROM gp_segment_configuration, pg_filespace_entry
    WHERE dbid=fsedbid
    ORDER BY dbid;.
```

新的用于恢复的Segment主机必须已经安装好GPDB软件并与已有的Segment主机进行了相同的配置，包括建立互信等。

恢复失败的 Master

在Master失效时，同步程序会停止，Standby可以被在本机被激活，激活Standby时，同步日志被用来恢复Master最后一次事务成功提交时的状态。在激活Standby时还可以指定一个新的Standby。

激活Standby

1. 首先，系统必须配置有Standby主机。参看”启用Master Mirror”相关章节。
2. 在Standby主机上运行gpactivatestandby命令。例如，-d参数指定要被激活的Standby的数据路径：

   $ gpactivatestandby -d /data/master/gpseg-1

注意，一旦激活了Standby，其将成为GPDB集群的Master。如果想在此时配置另外一个主机作为新的Standby，在运行gpactivatestandby时可以使用-c参数。例如：

```
$ gpactivatestandby -d /data/master/gpseg-1 -c new_standby_hostname
```

3. 在激活之后，运行gpstate命令检查状态：

```
$ gpstate -f
```

新激活的Master应处于活动(Active)状态，而且如果同时还配置了一个新的Standby主机，应该为被动(Passive)状态(如果没有配置，则为未配置(Not Configured)状态)。

4. 在切换之后，在新的Master主机上运行ANALYZE。例如：

```
$ psql dbname -c 'ANALYZE;'
```

5. 作为可选项，如果在激活Standby时没有指定新的Standby，可以在之后使用gpinitstandby命令配置一个新的Standby。在当前活动的Master主机上运行这个命令。例如：

```
$ gpinitstandby -s new_standby_master_hostname
```

恢复 Master 的原有角色

在激活Standby之后，假如Standby的主机与原有的Master主机功能和可靠性等价，可以继续将其作为Primary Master运行

除非在激活Standby的时候已经指定了新的Standby，最好初始化一个新的Standby以确保在有Master Mirror的情况下继续运行。在当前的Master上运行gpinitstandby命令配置一个新的Standby。

作为可选项，也许更愿意将Master与Standby恢复到最原始的主机上。这个过程实质上是将Master与Standby交换角色，这仅应该在必须使用最原始主机运行Master的情况再操作。

恢复Master和Standby到原始主机

1. 确保原始主机处于可靠的运行状态。那些导致失败的因素已经被彻底解决。
2. 在当前Master上执行如下操作，将Standby初始化到原始主机。例如：

```
$ gpinitstandby -s original_master_hostname
```

3. 在当前的Master主机(原始角色为Standby)上停止Master进程。例如：

```
$ gpstop -m
```

4. 在原始Master主机上(当前为Standby)运行gpactivatestandby命令。例如，-d参数指定要被激活的Standby的数据路径：

```
$ gpactivatestandby -d $MASTER_DATA_DIRECTORY
```

5. 在激活之后，运行gpstate命令检查状态：

```
$ gpstate -f
```

6. 原始Master状态应为活动(Active)，而Standby状态应为未配置(Not Configured)。一旦原始的Master再次作为GPDB的Master在运行，即可在原始Master主机上初始化一个Standby。例如：

```
$ gpinitstandby -s original_standby_master_hostname
```

重新同步Standby

有时可能会出现Master与Standby之间的日志同步程序停止，或者同步时间已经过期。可通过gp_master_mirroring系统日志表来查看Master与Standby之间的最后更新日期。例如：

```
$ psql dbname -c 'SELECT * FROM gp_master_mirroring;'
```

要同步Standby并更新到最新的同步，运行下面的gpinitstandby命令(使用-n参数)：

```
$ gpinitstandby -s standby_master_hostname -n
```

第十七章：备份与恢复

本章讲述在GPDB系统中备份与恢复数据库和用户数据。包含如下内容：

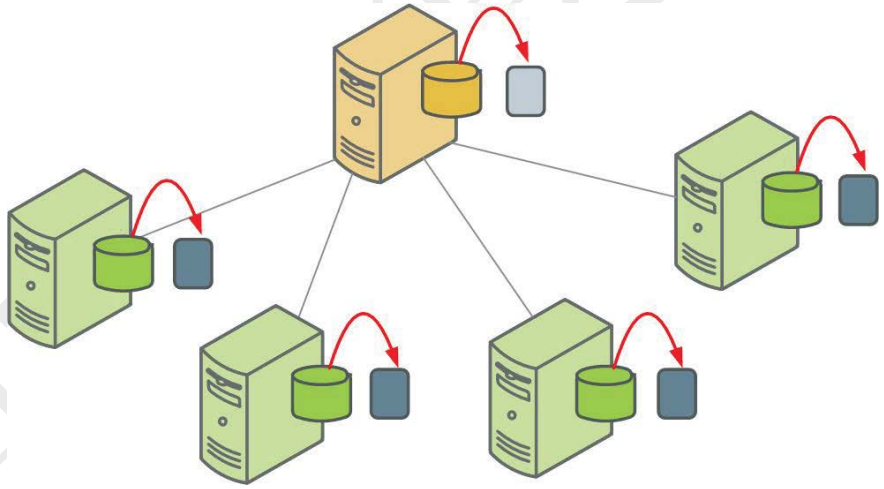
- 备份恢复操作概述
- 备份数据库
- 从并行备份文件恢复

备份恢复操作概述

GP建议定期备份数据库。在系统失败或者数据损坏时可以使用备份来恢复重建GPDB系统。另外还可以使用备份将GPDB数据从一个系统迁移到另外一个系统。

关于并行备份

GP提供了一个并行备份命令gp_dump。该命令同时备份Master和所有活动的Segment Instance。因为Instance是并行备份的，所以消耗的时间与系统中Instance的数量没有关系。在Master主机上的备份文件包含DDL语句和具有GP特征的系统日志表(比如gp_segment_configuration)。Instance上的备份文件包含独立Instance的数据。所有的备份文件组成了一个完整的备份集合，通过一个唯一的14位数字的时间戳来识别。



为了实现自动备份，GP还提供了gpcrondump命令，作为gp_dump命令的一个包装，备份命令可以直接被调度器CRON调用。gpcrondump命令还允许备份除了数据库和数据之外的对象，比如数据库角色和服务器配置等。查看”使用gpcrondump自动并行备份”相关章节获得更多信息。

关于非并行备份

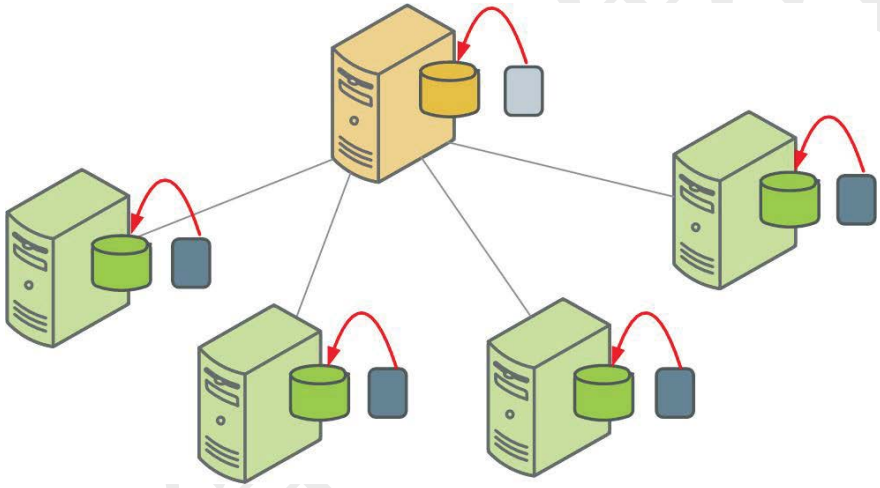
GP依然支持常规的PostgreSQL备份命令pg_dump和pg_dumpall。PostgreSQL的备份工具(在GPDB中使用)将在Master主机上创建一个大的备份文件，包含所有活动Instance的数据。大多数情况下，这种做法是不切实际的，Master主机上不太可能有足够的磁盘空间来存储一个分布式数据库的全部数据。这个命令多用于从常规的PostgreSQL数据库系统迁移到GPDB系统。

另外一个有用的从数据库导出数据的命令为SQL命令COPY TO。允许将一张表的全部或者一部分导出，以分割文本文件的方式存储在Master主机。

如果要迁移数据到一个Instance配置不同的GPDB系统(比如Instance数量不同)，GP建议使用gp_dump或者gpcrondump并行备份文件，然后使用如”恢复到配置不同的GPDB系统”章节描述的方式恢复。

关于并行恢复

要实现并行恢复，必须有一个完整的由gp_dump或gpcrondump创建备份集合。GP提供了一个并行恢复命令gp_restore。这个命令通过gp_dump产生的时间戳来辨识备份集合，恢复数据库对象和数据到分布式数据库中。与并行备份一样，每个Instance的数据被并行恢复。



GP还提供了gpdbrestore命令，该命令作为gp_restore的包装，提供了更灵活的选项，这对于有些场景是很有用的，比如，使用gpcrondump自动备份的文件来恢复，或者，备份文件已经移出GP集群到其他主机。参考”使用gpdbrestore恢复数据库”相关章节。

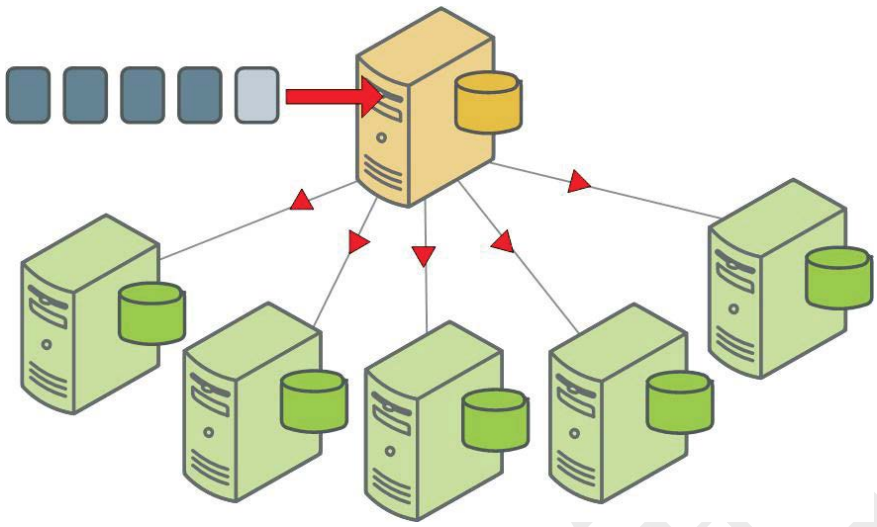
关于非并行恢复

GP依然支持常规的PostgreSQL恢复命令pg_restore，这个命令更多的用于支持从常规的PostgreSQL数据库迁移到GPDB，使用由pg_dump或pg_dumpall创建的备份文件来恢复。在恢复PostgreSQL备份文件到GPDB之前，需确认修改了CREATE TABLE语句，使得其包含了GPDB的DISTRIBUTED子句。

注意：如果不包含DISTRIBUTED子句，GPDB会选择一个缺省值。更多信息查看”创建表”相关章节。

有时，相对于并行恢复来说非并行恢复可能是很有必要的，例如，从4个Primary的系统迁移到5个Primary的系统。此时，无法做并行恢复，因为只有4份备份文件，无法在新的系统上平坦分布。非并行恢复涉及到，将每个Segment主机上

的备份文件收集，复制到新系统的Master主机，通过新Master进行装载。参见”恢复到配置不同的GPDB系统”相关章节。



备份数据库

备份数据库有3种方式：

- **为每个Instance创建一个备份文件。**如果想要备份一个数据库，或者在相同配置的系统之间迁移数据(比如相同数量的Instance但主机不同)，使用这种方式。要恢复，必须使用对应的命令gp_restore。如果有必要，对于恢复到配置不同的GPDB系统也可以使用gp_dump命令来备份文件。
- **使用gpcrondump进行定期备份。**这是对gp_dump的一个包装，使用cron(UNIX操作系统的调度程序)调度GPDB数据库的后台自动备份。调用gpcrondump的调度作业应该部署在GP的Master主机。gpcrondump命令还允许备份除了数据库和数据之外的对象，比如数据库角色和服务器配置等。
  使用gpcrondump调度可作为GP数据库的增强功能。
- **使用pg_dump或pg_dumpall创建单个备份文件。**如果要迁移数据到其他数据库系统可以选择这种方式。如果恢复到PostgreSQL或者GPDB，可以相应的使用pg_restore命令(归档格式的备份文件)，或者使用客户端程序如psql(平面格式的备份文件)。如果是计划恢复到其他的GPDB系统，GP推荐使用并行备份gp_dump或gpcrondump，而恢复时使用非并行模式恢复。

使用 DDBoost

DDBoost(Data Domain Boost)可用于备份和恢复操作。DDBoost提供了快速后续备份和源端去重功能以减少网络流量。要使用DDBoost，可简单的通过gpcrondump或gpdbrestore结合一个参数来实现。不要在gp_dump、pg_dump或pg_dumpall命令中使用DDBoost。当数据通过DDBoost从DD系统恢复，一些文件会百拷贝到Master本地磁盘并恢复，而其他的文件会被直接恢复。

使用DDBoost的条件

在配置DDBoost之前必须购买并在DD上安装DDBoost许可。还需要DDBoost在

尺寸规划上的建议。这需要联系EMC DD的客户代表以获得这方面的帮助。

译者注：由于DD属于EMC的备份设备，DDBoost是DD的备份软件，目前支持对GP的备份，如果有这方面的需求，购买DD和DDBoost以及GP时，相信这方面的技术问题EMC会有相关支持，这里不再说明，如有兴趣可以参考官方文档内容。

使用 gp_dump 备份

gp_dump命令会备份GPDB系统的内容为一系列的SQL命令文件，用作恢复GPDB系统的配置、数据库和数据。在备份操作期间，用户是可以访问数据库的。

gp_dump命令将执行如下的操作并产生相应的备份文件：

在Master主机上

- 在Master的数据目录备份GP配置的系统日志表到一个SQL文件，备份文件的名称为：
  gp_catalog_1_<dbid>_<timestamp>
- 在Master的数据目录备份CREATE DATABASE SQL语句到一个文件，备份文件的名称为：
  gp_cdatabase_1_<dbid>_<timestamp>
  该语句可以在Master上运行来重建数据库。
- 在Master的数据目录备份用户数据库的模式到一个SQL文件，备份文件的名称为：
  gp_dump_1_<dbid>_<timestamp>
  该文件被gp_restore用来重建数据库的模式。
- 在Master的数据目录备份一个包含重建Table相关对象的文件，备份文件的名称为：
  gp_dump_1_<dbid>_<timestamp>_post_data
  在使用gp_restore命令恢复数据库时，首先，模式和数据被恢复，然后这个备份文件被用来重建Table相关的其他对象。
- gp_dum在每个Instance上启动一个名为gp_dump_agent的程序进行备份。gp_dump_agent进程在Segment主机上运行并向Master主机上的gp_dump进程报告状态。

在Segment主机上

- 在Instance数据目录备份用户数据到一个SQL文件，缺省时，之后Primary(或活动的)Instance会备份。备份文件的名称为：
  gp_dump_0_<dbid>_<timestamp>
  该文件被gp_restore用来重建特定Instance的用户数据。
- 在Instance数据目录创建一个日志文件，日志文件的名称为：
  gp_dump_status_0_<dbid>_<timestamp>

注意14位数字的时间戳，它是唯一标识备份操作的标识符，且作为gp_dump操作产生的每个备份文件名称的组成部分。这个时间戳在使用gp_restore来恢复GPDB时必须指定。

使用gp_dump命令备份GP数据库

1. 从Master主机，运行gp_dump命令。例如(这里的mydatabase是需要备份的数据库名称)：

   $ gp_dump mydatabase

注意：gp_dump在试图解析Segment主机的Hostname时，是从命令发起的主机来解析的，而不是从Master所在的主机，而这些解析是在连接之前完成的。如果不是在Master主机上运行这个命令，可能会因此导致失败。为了避免这个问题，可使用gpcrondump命令。

使用 gpcrondump 备份

gpcrondump是对gp_dump的一个包装，可以直接调用或者从crontab中调用。这个命令还允许备份除了数据库和数据之外的对象，比如数据库角色和服务器配置等。

gpcrondump在Master和Segment的数据目录创建备份文件在如下目录：

<data_directory>/db_dumps/YYYYMMDD

Segment数据的备份文件使用gzip压缩格式。

使用CRON调度备份操作

1. 在Master主机上，以GP SUPERUSER登录(gpadmin)。
2. 定义一个调用gpcrondump的crontab条目。例如，安排在午夜过1分钟的时候备份sales数据库(SELL设置为/bin/bash，PATH包含了GPDB管理命令)：

   Linux示例:

   SHELL=/bin/bash

   GPHOME=/usr/local/greenplum-db-4.2.0.0

   MASTER_DATA_DIRECTORY=/data/gpdb_p1/gp-1

   01 0 * * * gpadmin source $GPHOME/greenplum_path.sh;gpcrondump -x sales -c -g -G -a -q >> gp_salesdump.log
3. 创建一个名为mail_contacts的文件放置在GP SUPERUSER根目录。例如：

   $ vi /home/gpadmin/mail_contacts
4. 在该文件中，每行输入一个电子邮件地址。例如：

   dba@mycompany.com

   jjones@mycompany.com
5. 保存关闭mail_contacts文件。gpcrondump将会通知这个文件中列出的电子邮件地址。

从并行备份文件恢复

从并行备份文件恢复数据库的程序依赖于几个因素。在决定使用恢复程序时，需先确定以下几个问题：

1. **备份文件在哪里？**如果备份文件位于gp_dump生成时的原始位置，可以简单的使用gp_restore命令来恢复。如果备份文件已经移出GP集群，例如，移到归档

主机，使用gpdbrestore来恢复。

2. **是否需要恢复整个系统，还是只恢复数据？**如果GPDB仍在运行并仅需要恢复数据，使用gp_restore或gpdbrestore命令来恢复。如果丢失了整个集群或者需要从备份来重建整个集群，使用gpinitsystem命令。
3. **是否恢复的系统与备份时的系统具有相同数量的Instance？**如果恢复的系统与备份的系统具有相同数量的Instance，使用gp_restore或gpdbrestore命令来恢复。如果是在不同集群迁移间迁移，必须使用非并行恢复。参见"恢复到配置不同的GPDB系统"。

使用 gp_restore 恢复

gp_restore命令使用由gp_dump操作生成的备份文件重建数据定义(模式)和用户数据等。要想进行恢复，必须具备以下几点：

1. 存在gp_dump操作生成的备份文件。
2. 备份文件位于gp_dump生成时的原始位置。
3. GPDB系统正在运行。
4. 当前恢复的GPDB系统与使用gp_dump备份时的GPDB系统具有相同数量的Instance。
5. 要被恢复的数据库(Database)在系统中已经创建。
6. 如果在备份时使用了参数：-s(仅模式)，-a(仅数据)，--gp-c(压缩)，--gp-d(修改备份文件目录)，那么在使用gp_restore恢复时也必须指定这些参数。

gp_restore命令将执行如下的操作：

在Master主机上

- 运行由gp_dump生成的gp_dump_1_<dbid>_<timestamp>文件中SQL DDL命令，重建数据库的模式和对象。
- 在Master数据目录生成日志文件，日志文件的名称为：
  gp_restore_status_1_<dbid>_<timestamp>
- gp_restore在每个需要恢复的Instance上启动一个名为gp_restore_agent的程序，gp_restore_agent进程在Segment主机上运行并向Master主机上的gp_restore进程报告状态。

在Segment主机上

- 每个Instance使用gp_dump生成的gp_dump_1_<dbid>_<timestamp>文件来恢复用户数据。恢复时Primary和Mirror都会被恢复。
- 每个Instance生成一个日志文件，日志文件的名称为：
  gp_restore_status_1_<dbid>_<timestamp>

注意14位数字的时间戳，它是唯一标识用作恢复的备份操作标识符，且作为gp_dump操作产生的每个备份文件名称的组成部分。这个时间戳在使用gp_restore来恢复GPDB时必须指定。

从gp_dump创建的备份恢复

1. 确保要恢复的GPDB系统，由gp_dump生成的备份文件位于Master和Segment主机上。
2. 确保在系统中，需要恢复的数据库(Database)已经被创建。例如：

$ createdb database_name

3. 从Master主机，运行gp_restore命令。例如(--gp-k指定备份操作时间戳标识符，-d指定恢复的数据库)：

$ gp_restore -gp-k=2007103112453 -d database_name

使用 gpdbrestore 恢复

gpdbrestore命令是对gp_restore命令的一个包装，提供了更灵活的选项，比如，使用gpcrondump自动备份的文件来恢复。使用gpdbrestore恢复必须具备：

1. 存在gpcrondump操作生成的备份文件。
2. GPDB系统正在运行。
3. 当前恢复的GPDB系统与使用gp_dump备份时的GPDB系统具有相同数量的Instance。
4. 要被恢复的数据库(Database)在系统中已经创建。

使用gpdbrestore从归档主机恢复

(这个过程假设已经将备份文件移出GP集群到同一网络内的其他主机)

1. 首先，确保可以供GP Master主机访问归档主机。

$ ping archive_host

2. 确保在系统中，需要恢复的数据库(Database)已经被创建。例如：

$ createdb database_name

3. 从Master主机，运行gpdbrestore命令。例如(-R指定备份文件所在的主机名和路径)：

$ gpdbrestore -R archive_host:/gpdb/backups/archive/20080714

恢复到配置不同的 GP 系统

要使用gp_restore或gpdbrestore并行恢复操作，恢复的系统必须与备份的系统具有相同的配置(相同数量的Instance)。如果想要恢复数据库对象和数据到配置不同的系统(比如系统扩展了更多的Segment)，仍然可以使用并行备份文件来恢复，通过GP Master做非并行装载。要进行非并行装载，必须具备：

1. 全部由gp_dump或gpcrondump操作生成的备份文件。Master的备份文件包含了重建数据库对象的DDL。Segment的备份文件包含了用户数据。
2. GPDB系统正在运行。
3. 确保在系统中，需要恢复的数据库(Database)已经被创建。

如果查看Segment备份文件的内容，会发现，其简单的包含了COPY命令，数据是分割平面格式。如果收集了所有Instance的备份文件并通过Master装载，即可恢复所有的数据并重分布到新的系统。

恢复到配置不同的GP系统

1. 首先确保具备了全部的备份文件。包括Master的备份文件(gp_dump_1_1_<timestamp>， gp_dump_1_1_<timestamp>_post_data)和每个Instance的备份文件(gp_dump_0_2_<timestamp>，gp_dump_0_3_<timestamp>，gp_dump_0_4_<timestamp>，等等)。所有的备

份文件必须含有相同时间戳标识符。缺省状态下，gp_dump在每个Instance的数据目录生成备份文件，因此，可能需要收集所有的备份文件并放置到需要恢复的系统Master主机。如果Master没有足够的磁盘空间，可以拷贝一个Instance，装载，删除，再拷贝其他Instance。

2. 确保在系统中，需要恢复的数据库(Database)已经被创建。例如：

   $ createdb database_name

3. 装载Master备份文件以恢复数据库对象。例如：

   $ psql database_name -f /gpdb/backups/gp_dump_1_1_20080714

4. 装载每个Segment的备份文件以恢复数据。例如：

   $ psql database_name -f /gpdb/backups/gp_dump_0_2_20080714

   $ psql database_name -f /gpdb/backups/gp_dump_0_3_20080714

   $ psql database_name -f /gpdb/backups/gp_dump_0_4_20080714

   $ psql database_name -f /gpdb/backups/gp_dump_0_5_20080714

   ...

5. 装载Table相关对象的文件，恢复数据库对象如索引、触发器、主键约束等：

   $ psql database_name -f /gpdb/backups/gp_dump_0_5_20080714_post_data

第十八章：扩展 GP 系统

本章介绍为了提升性能和增加存储容量而为已有系统增加新的资源。本章包含如下内容：

- GP系统扩展规划
- 节点的准备与添加
- 初始化新Instance
- 重分布表
- 清除扩展Schema

虽然本章会提供一些关于硬件准备的信息，但主要介绍软件方面的扩展准备。GP建议在配置扩展GPDB的硬件资源时最好获得GP的平台方面工程师的支持。

GP 系统的扩展规划

对于系统扩展操作来说，小心的规划是成功的关键。通过精心的准备硬件和规划所有扩展程序的步骤，可以最大可能的降低风险和GPDB的停机时间。

超大规模系统的管理员可能会特别注重”规划重分布表”相关章节提到的性能问题。

本节概述性的介绍系统扩展程序和清单。

系统扩展概述

系统扩展包含3个阶段：

- 添加和测试新硬件
- 初始化新Instance
- 重分布表

添加和测试新硬件 -- 在”规划新硬件”章节介绍了硬件部署方面的一般性内容。而更多的硬件方面的信息，GP建议联系Greenplum平台工程师，也可以联系译者。在准备好硬件及配置好网络之后，应该先使用GP命令对硬件性能进行测试。

初始化新Instance – 在GPDB安装到新的硬件后，需要使用gpexpand(不是gpinitsystem)命令来初始化新Instance。在此过程中，命令会根据所有已有DB在新节点上创建数据目录并拷贝所有用户Table，在初始化新Instance的过程中不断捕获数据库的Table元数据并存储到扩展Schema中。在该过程成功之后，扩展配置即被提交，且无法撤销。

这些操作需要系统处于停机状态。如果在执行gpexpand时没有关闭GPDB系统，该命令会主动关闭GPDB系统。注意，在扩展过程中，这一步是必须停掉GPDB系统的，而所谓的在线扩展与这步无关，虽然这步操作通常时间较短。

重分布表 – 作为初始化程序的一部分，gpexpand命令会将HASH分布策略作废，并将所有表的分布策略设置为RANDOMLY。这一动作会涉及到GPDB系统中所有Instance的所有Table。在初始化结束并重新启动系统后，GPDB继续可以被

访问，不过，此时那些严重依赖HASH分布的查询，其性能会受到影响。在此期间常规的操作如ETL任务、用户查询、报表等都可以继续使用，不过性能可能会下降并伴随较差的响应时间。

注意：当表被设置为RANDOMLY策略，GPDB无法执行唯一约束。这时ETL和装载会受到影响，这种影响会持续一直到Table重分布完成为止，这时的重复记录不会违反约束。

为了完成扩展，还必须执行gpexpand命令在新的Instance之间重分布Table中的数据。根据系统的尺寸和规模，可以选择单会话模式运行数个小时，或者选择多会话模式每次运行预定时间周期。每个正在被重分布的表或分区，操作期间对于读和写都是不可用的。随着表成功的在新Instance之间基于DK(如果有的话)被重分布，该表的性能应该高于或者至少等于扩展之前的水平。

通常，在完成扩展的过程中，需要结合不同的参数执行4次gpexpand命令：

- 交互式的生成一个扩展配置文件：
  gpexpand -f hosts_file
- 初始化Instance并生成扩展Schema
  gpexpand -i input_file -D database_name
- 重分布表
  gpexpand -d duration
- 清除扩展Schema
  gpexpand -c

大规模系统可能需要多次会话来完成表的重分布，gpexpand将需要被多次运行来完成扩展，而重分布表的排名设置可能也会很有用。更多信息可参照”规划重分布表”相关章节。

系统扩展清单

这个清单概述性的介绍系统扩展的必须步骤。

扩展前的在线状态准备
系统处于正常运行状态
规划并执行新硬件的采购、配置与网络调整
规划扩展计划。Segment 节点的 Instance 数量，安排性能测试的时间，以及初始化新 Instance 的停机时间、安排重分布表的时间
执行一个完整的模式备份
在新的主机上安装 GPDB 的二进制安装文件
交换 SSH 密钥(gpssh-exkeys)
检查新硬件的 OS 环境(gpcheck)
检查新硬件的磁盘 I/O 和内存带宽(gpcheckperf)
检查在 Master 数据目录 pg_log 和 gpperfmon/data 下没有超大尺寸文件
准备扩展配置文件(gpexpand)
停机扩展任务
系统在这一过程中会被活动用户锁住

检查已有系统和新硬件的环境(gpcheck)
检查已有系统和新硬件的磁盘 I/O 和内存带宽(gpcheckperf)
初始化新 Instance 并创建扩展 Schema(gpexpand -i input_file)
在线重分布表
系统处于正常运行状态
在开始重分布前，停止所有快照操作和消耗磁盘的操作
在系统上重分布表(gpexpand)
清除扩展 Schema(gpexpand -c)
运行 ANALYZE 更新分布统计信息 运行 gpexpand -a 或者后续使用 ANALYZE 命令来分析

规划新硬件

对于系统扩展来说，精心准备新硬件至关重要。周密部署完全兼容的硬件，对于后续的系统扩展程序来说，可大大降低实施的风险。

需要做扩展的GPDB集群，新节点应该与现有资源相匹配。GP建议在为GP集群购买新的扩展硬件之前，最好获得GP平台工程师的支持。

规划安装新硬件包含几个大因素。一些可能的情况包括：

- 为新硬件准备物理空间。需考虑冷却、电力以及其他物理因素。
- 确定已有硬件连接新硬件的物理网络和布线。
- 为扩展的系统规划IP地址和网络结构。
- 获取已有硬件的系统配置(用户、配置文件、网络等)以帮助配置新硬件。
- 制定一个构建计划以便于在特定的环境中按照理想配置部署硬件。

在将新硬件加入网络环境后，确保执行”检查系统设置”中描述的接入任务。

规划新 Instance 初始化

GPDB扩展需要在有限的系统停机时间段内完成。在此期间，必须执行gpexpand完成新Instance的初始化和生成扩展Schema。

所需的时间取决于GP系统中Schema对象的数量，还有就是硬件性能的因素。多数情况下，新Instance的初始化可以在小于30分钟的停机时间内完成。

注意：在完成初始化新的Instance之后，将无法再使用扩展前的gp_dump文件来恢复系统。在初始化成功之后，扩展配置即被提交，且无法回滚。

规划Mirror Instance

如果已有集群有Mirror，新的Instance也必须有Mirror配置。相反的，如果已有系统没有Mirror，使用gpexpand命令时新的Instance无法添加Mirror。

对于有Mirror的GPDB集群来说，必须确保新增主机的数量足够适应添加新的Mirror。需要新主机的数量取决于已有Mirror的策略。

Spread Mirror – 新主机的数量至少比每个主机Instance的数量大1。主机的数量必须大于每个主机Instance的数量，以确保Mirror分布的平坦性。

Grouped Mirror – 新主机数不小于2。在最少2台主机的情况下，确保第一个主机上Instance的Mirror可以在第二个主机上，反之亦然。

更多信息参考”Segment镜像”相关章节。

增加每个主机Instance数量

缺省时，新初始化的主机与已有主机配置相同数量的Instance。作为可选，在扩展时，可以调整增加每个主机的Instance数量，还可以在不增加主机数量的情况下只增加现有主机的Instance数量。

例如，现有系统每个主机有2个Instance，可以使用gpexpand初始化，在现有主机上增加2个Instance(总数变为4)，同时新主机的Instance数量也为4。

在生成扩展配置文件时，交互式处理命令会提示该选项，扩展配置文件的格式允许手动修改。参考”生成扩展配置文件”相关章节。

关于扩展Schema

在初始化新Instance时，gpexpand会生成一个扩展Schema。如果在初始化时(gpexpand -D)不指定数据库，扩展Schema会创建到环境变量PGDATABASE定义的数据库中。

系统中每张表的源数据被存到扩展Schema中，以帮助跟踪扩展过程中的状态。该Schema中包含两张Table和一个View用以跟踪扩展操作的状态：

- gpexpand.status
- gpexpand.status_detail
- gpexpand.expansion_progress

可以通过修改gpexpand.status_detail表来控制扩展细节。例如，从该表中删除记录以阻止相应Table扩展到新Instance上。通过修改记录的rank字段的值，可以控制Table在重分布过程中的先后顺序。更多信息参考”排名重分布表”相关章节。

规划重分布表

重分布表是系统运行时进行的。对于多数GP系统来说，重分布表的操作可以在一个短期的gpexpand会话中完成。对于大型系统来说，可能需要安排多个gpexpand会话来完成，还需要安排重分布表之间的顺序，目的是尽可能的减小性能的影响。如果数据库的尺寸和设计允许的话，GP建议尽可能使用一个会话来完成重分布。

注意：为了完成重分布表，Segment主机必须有足够的空间来存放大表的临时数据。每个正在被重分布的表，操作期间对于读和写都是不可用的。

重分布对性能的影响取决于表的尺寸、存储类型和分区结构。一张表重分布的时间与CREATE TABLE AS SELECT操作相当。在重分布一张TB级别的事实表时可能会占用大部分的可用系统资源，这对于其他的查询和工作负载会有很大的影响。

管理大规模GP系统的重分布

可以如在”排名重分布表”章节中描述的那样调整重分布的等级来管理重分布的顺序。对重分布顺序的管理可以应对磁盘空间的限制，以及快速恢复查询的性能。

在规划重分布时，应该考虑每个表在被重分布时独占锁的影响。用户活动对表的影响可能会延迟重分布计划的开始时间。同样的，当gpexpand正在重分布某张表时，其他操作也无法访问这张表。

系统有丰富的磁盘空间

在磁盘空间(存储大表临时数据需要)丰富的系统中，可以首先重分布最常用的表以尽快的恢复查询性能。因此，将这些表的排名调高，并在系统资源利用较低的时候安排重分布操作。只运行一次重分布程序，直到大表或关键表已经成功的完成了重分布。

系统空余磁盘空间有限

如果已有的节点上的磁盘空间有限，应该尽量先重分布较小的表，这样可以为后续被重分布的大表清理出足够的磁盘空间。已有节点的空余空间会随着每完成一张表的重分布而增加。一旦所有节点的磁盘空间都足够存储一张最大的表，此时可以开始大表和关键表的重分布了。重申一次，因为排它锁的要求，应该在系统空闲时间安排大表的重分布。

还应该考虑以下因素：

- 在空闲时间运行并发的重分布操作，最大化利用系统资源来完成重分布。
- 在使用并发重分布操作时，需确认GP系统的最大连接数限制。更多信息参考”限制并发连接”相关章节。

重分布AO表和压缩表

AO表与压缩AO表在使用gpexpand重分布时的效率与堆表是不同的。对于压缩表来说，需要CPU资源来压缩和解压数据，这会增加对系统性能的影响。对于类似数据类似尺寸的表来说，总体的性能差异大致如下：

- 非压缩AO表比堆表的重分布快10%。
- ZLIB压缩AO表比非压缩AO表的重分布慢的很明显，可能会慢80%。
- 使用如ZFS/LZJB之类数据压缩的文件系统，通常重分布也会慢的很明显。

主要：如果数据在已有节点上使用了压缩，那么在新的节点上应该使用同样的压缩以避免磁盘空间不足的情况出现。

重分布有主键约束的表

在初始化新Instance完成之后和重分布表完成之前，这段时间内的主键约束是不

生效的。在此期间插入到表中的重复数据会妨碍重分布表。一旦表被成功重分布，主键约束会再次生效。

如果在扩展中数据违反了约束，在所有表重分布完之后，会在屏幕打印出警告信息。可以使用如下的方法补救约束例外情况：

- 清除违反主键约束的重复数据，重新运行gpexpand命令。
- 删除主键约束，重新运行gpexpand命令。

重分布有自定义类型的表

对于那些有被删除自定义类型列的表，无法通过扩展命令来执行重分布。要重分布这些有被删除自定义类型列的表，先使用CREATE TABLE AS SELECT重建该表，这样，那些被删除的列在该过程中会被清除，然后再使用扩展命令重分布该表。

重分布分区表

因为扩展命令可以一个分区一个分区的重分布一张大表，因此，有效的分区设计可以显著的降低重分布对性能的影响。仅仅是分区表的子表会被置为RANDOMLY策略，而且也只有子表在重分布时会获取读写锁定的排它锁。

重分布有索引的表

由于在重分布表之后gpexpand命令必须重建索引，高度索引的表重分布会有很大的性能影响。重分布有密集索引表的系统将会慢的非常明显。

节点的准备与添加

为扩展准备新节点的系统，安装GPDB二进制文件，交换SSH互信，执行性能测试。GP推荐至少执行两次性能测试：第一次只测试新节点，第二次新节点和已有节点一起测试。为了避免用户活动扭曲测试结果，第二次必须在系统停机状态下进行。

除了这些通用的指导外，GP建议在网络环境发生变化或者任何系统环境发生变化时都应该执行性能测试。比如：要准备在两套网络环境下扩展系统，应该在每套环境下分别执行性能测试。

本节的剩余部分解释如何运行GP管理命令来检查准备整合到已有GP系统的新节点。

将新节点添加到互信环境

为了确保GP管理工具可以在不提示密码输入的情况下连接到所有节点，新节点必须与已有节点之间交换SSH密钥。

GP建议执行两次密钥交换：一次使用root用户(为了方便管理)，一次使用gpadmin用户(GP管理命令需要)。按照顺序执行如下任务：

- 交换root的SSH密钥
- 创建gpadmin用户

- 交换gpadmin的SSH密钥

交换root的SSH密钥

1. 创建两个独立的Host列表文件：一个包含现有GPDB集群的所有Host名称，另一个包含所有新扩展节点的Host名称。对于已有主机，可使用首次建立SSH密钥的Host文件。

   Host文件应该包含所有主机(Master、Standby和Instance主机)，每个Host名称一行。如果使用了多网口配置，需确保交换了每个主机所有HostName的SSH密钥。文件中不能有空行和多余的空字符。例如：

   ```
   mdw
   sdw1-1
   sdw1-2
   sdw1-3
   sdw1-4
   sdw2-1
   sdw2-2
   sdw2-3
   sdw2-4
   sdw3-1
   sdw3-2
   sdw3-3
   sdw3-4
   ```

   或者

   ```
   masterhost
   seghost1
   seghost2
   seghost3
   ```

2. 在Master主机上以root登录，从GP安装目录加载greenplum_path.sh文件。

   ```
   $ su -
   # source /usr/local/greenplum-db/greenplum_path.sh
   ```

3. 使用Host列表文件执行gpssh-exkeys命令。例如：

   ```
   # gpssh-exkeys -f /home/gpadmin/existing_hosts_file -x /home/gpadmin/new_hosts_file
   ```

4. gpssh-exkeys会检查远程主机并在所有主机之间执行密钥交换。在提示的时候输入root用户密码。例如：

   ```
   ***Enter password for root@hostname: <root_password>
   ```

创建gpadmin用户

1. 使用gpssh命令在所有新Segment主机上创建gpadmin用户(如果还没有的话)。使用之前创建用于交换密钥的Host文件。例如：

   ```
   # gpssh -f new_hosts_file '/usr/sbin/useradd gpadmin -d /home/gpadmin -s /bin/bash'
   ```

2. 设置新建gpadmin用户的密码。在Linux上，可以通过gpssh一次完成所有节点的操作。例如：

   ```
   # gpssh -f new_hosts_file 'echo gpadmin_password | passwd gpadmin --stdin'
   ```

在Solaris上，需要登录到每个Segment主机并设置gpadmin用户密码。例如：

```
# ssh segment_hostname
# passwd gpadmin
# New password: <gpadmin_password>
# Retype new password: <gpadmin_password>
```

3. 通过查看根目录确认gpadmin用户已经创建：

```
# gpssh -f new_hosts_file ls -l /home
```

交换gpadmin的SSH密钥

1. 以gpadmin用户登录Master书籍，使用相关的Host列表文件执行gpssh-exkeys命令。例如：

```
# gpssh-exkeys -e /home/gpadmin/existing_hosts_file -x /home/gpadmin/new_hosts_file
```

2. gpssh-exkeys会检查远程主机并在所有主机之间执行密钥交换。在提示的时候输入gpadmin用户密码。例如：

```
***Enter password for gpadmin@hostname: <gpadmin_password>
```

检查 OS 设置

使用gpcheck命令来检查所有新Segment主机，确保其OS设置符合GPDB软件运行的要求。

运行gpcheck

1. 使用运行GPDB系统的用户(比如gpadmin)登录Master主机。

```
$ su - gpadmin
```

2. 使用Host列表文件执行gpcheck测试新Segment主机。例如：

```
$ gpcheck -f new_hosts_file
```

检查磁盘 I/O 和内存带宽

使用gpcheckperf命令来测试磁盘I/O和内存带宽。

运行gpcheckperf

1. 使用Host列表文件执行gpcheckperf测试新Segment主机。使用-d参数指定在每个主机(必须具有这些目录的写权限)上需要测试的文件系统目录。例如：

```
$ gpcheckperf -f new_hosts_file -d /data1 -d /data2 -v
```

2. 由于完成该测试需要在Host上拷贝很大的文件，该命令需要花费较长的时间。在测试完成时，将会列出磁盘写、磁盘读、流测试结果的概要。

注意：这个例子中没有指定-r参数，真实的测试可参考”测试硬件性能”相关章节，而不是直接使用本例中的命令。

集成新硬件到系统中

在初始化新Instance之前，应在包含新Segment主机的所有节点上重复一遍性能测试。在停机之后使用包含所有主机名称的Host文件执行同样的测试：

- 检查OS设置
- 检查磁盘I/O和内存带宽

由于用户活动会扭曲测试结果，必须在GPDB停机(gpstop)后执行测试。

初始化新 Instance

使用gpexpand命令来初始化新Instance，生成扩展Schema，在系统范围内设置DB的分布策略为RANDOMLY。缺省情况下，在第一次使用有效的扩展配置文件从Master运行gpexpand命令时就完成了这些任务。随后再执行gpexpand命令，其将发现扩展Schema已经创建，并开始执行重分布表。

生成系统扩展配置文件

要开始扩展，gpexpand命令需要一个包含关于新Instance和Host信息的配置文件。如果不指定配置文件运行gpexpand命令，命令将显示一个交互式的方式，采集需要的信息以自动生成配置文件。

如果使用交互方式生成配置文件，可以选择性的指定文件包含哪些扩展主机。如果在交互式输入主机名时所使用的平台或命令对长度有限制，可以通过-f参数来指定主机名列表。

交互模式生成扩展配置文件

在执行gpexpand以交互模式生成配置文件之前，需要确认下列所需信息：

- 新节点数量(或者Host文件)
- 新节点的Host名称(或者Host文件)
- 已有节点的Mirror策略，假如有
- 每个主机添加的Instance数量，假如有

gpexpand程序会根据这些信息生成一个配置文件，包含dbid、content ID、数据目录(和gp_segment_configuration表中的一样)，该配置文件会被保存在命令运行的当前目录。

1. 以GPDB系统运行用户(比如gpadmin)登录Master主机。
2. 运行gpexpand命令。命令会提示关于准备扩展操作的信息，提示退出或继续。

   作为可选，使用-f指定一个Host文件。例如：

   ```
   $ gpexpand -f /home/gpadmin/new_hosts_file
   ```
3. 在提示时，选择[Y]继续。
4. 除非使用-f指定Host文件，命令会提示输入Host Name。将新扩展主机的主机名按照逗号分割的方式输入。不要包含多网口相关的主机名。例如：

   ```
   > sdw4-1, sdw4-2, sdw4-3, sdw4-4
   ```

   正确的使用是列出每个节点独立的主机名。例如：

   ```
   > sdw4, sdw5, sdw6, sdw7
   ```

   如果只是在已有节点上增加Instance，在提示处输入空行即可。不要输入localhost或者任何系统中已经存在的Host Name。
5. 假如有Mirror，输入Mirror策略。选项为spread|grouped|none，缺省为grouped。需确保有足够的节点数量来支撑选择的Mirror策略。更多信息参见”规划

Mirror Instance”相关章节。

6. 如果有必要，输入要在每个节点上增加的Instance数量。缺省情况下，新节点初始化与已有节点相同数量的Instance。但作为可选项，可以增加每个节点的Instance数量。

   如果想要增加每个节点Instance的数量，输入一个非0的数字。该数量的额外Instance会在所有节点上被初始化。例如，现有系统每个节点有2个Instance，输入一个数值2，在现有节点上会新初始化2个Instance,同时新节点的Instance数量也为4个。

7. 如果增加新的Instance，需要为新Instance输入数据目录的跟路径。不要指定真实的数据目录，其会由gpexpand命令在已有的数据目录下自动创建。
   例如，已有的数据目录像这样：

   /gpdata/primary/gp0

   /gpdata/primary/gp1

   应该像下面这样输入(每个提示输入一个)来指定两个新的Instance的数据目录：

   /gpdata/primary

   /gpdata/primary

   在执行初始化时，gpexpand命令会在/gpdata/primary目录下自动创建gp2和gp3两个新的目录。

8. 如果增加新的Mirror，需要为新Mirror输入数据目录的跟路径。不要指定真实的数据目录，其会由gpexpand命令在已有的数据目录下自动创建。
   例如，已有的数据目录像这样：

   /gpdata/mirror/gp0

   /gpdata/mirror/gp1

   应该像下面这样输入(每个提示输入一个)来指定两个新的Mirror的数据目录：

   /gpdata/mirror

   /gpdata/mirror

   在执行初始化时，gpexpand命令会在/gpdata/mirror目录下自动创建gp2和gp3两个新的目录。

   重要提示：那些为Instance和Mirror指定的目录必须在新的节点存在，而且运行gpexpand命令的用户需具备这些目录的写权限。

   在输入全部需要的信息之后，命令会在当前路径生成一个扩展配置文件。
   例如：

   gpexpand_inputfile_yyyymmdd_224553

扩展配置文件的格式

完全可以按照要求的格式自行创建扩展配置文件。除非对于扩展来说有特殊的需求，GP都建议使用交互的模式来生成扩展配置文件。

扩展配置文件的格式为：

hostname:address:port:fselocation:dbid:content:preferred_role:replication_port

例如：

sdw5:sdw5-1:50011:/gpdata/primary/gp9:11:9:p:53011

sdw5:sdw5-2:50012:/gpdata/primary/gp10:12:10:p:53011

sdw5:sdw5-2:60011:/gpdata/mirror/gp9:13:9:m:63011

sdw5:sdw5-1:60012:/gpdata/mirror/gp10:14:10:m:63011

一个扩展配置文件在格式上需要每个新Instance的如下信息：

参数	有效值	描述
hostname	主机名	Segment 节点的主机名
port	可用的端口号	数据库的监听端口。应该从已有的端口为基数增加端口号的数值。
fselocation	目录名称	Instance 的数据目录(文件空间)位置。如同 pg_filespace_entry 系统表中一样。
dbid	整数。不可与已有的冲突	Instance 的数据库 ID。该值应该从已有的值(参见系统日志表 gp_segment_configuration)增加而来。例如，已有的 10 个 Instance 的集群 dbid 为 1-10，那么新的 Instance 的 dbid 应该为 11-14
content	整数。不可与已有的冲突	Instance 的目录 ID。Primary 的应该与对应的 Mirror 相同。该值应该从已有的值增加而来。更多信息参考系统日志表 gp_segment_configuration
preferred_role	p|m	指定该 Instance 为 Primary 或者 Mirror。p 代表 Primary，而 m 代表 Mirror
replication_port	可用的端口号	Instance 的文件同步端口。该值应该从已有的 replication_port 值增加而来

运行 gpexpand 初始化新 Instance

在生成扩展配置文件之后，运行gpexpand来初始化新Instance。由于初始化Instance的需要，该命令会自动停止GPDB，在完成初始化之后会自动重启系统。

1. 以GPDB系统运行用户(比如gpadmin)登录Master主机。
2. 使用-i参数指定扩展配置文件，运行gpexpand命令。作为可选，使用-D指定扩展Schema所在的数据库。例如：

   $ gpexpand -i input_file -D database1

   该命令会查出GP系统中是否已经存在一个扩展Schema。如果已经存在一个扩展Schema，在开始新的扩展操作之前应该使用gpexpand -c命令清除该Schema。参照”清除扩展Schema”相关章节。

   在新Instance被成功初始化且扩展Schema被成功创建后，gpexpand命令会打印成功信息并退出。

在初始化过程完成之后，即可连接到GPDB查看扩展Schema。该Schema位于-D参数指定的数据库，或者通过PGDATABASE环境变量指定的数据库。更多信息查看”关于扩展Schema”相关章节。

回滚失败的扩展

可以使用gpexpand -r|--rollback命令来回滚失败的扩展安装。不过，这个命令仅适用于失败的情况。一旦操作成功完成，扩展即被提交，且不能回滚。

要回滚失败的扩展安装，使用下面的命令，指定包含扩展Schema的数据库名称：

```
gpexpand --rollback -D database_name
```

重分布表

在成功生成一个扩展Schema之后，就可以将GPDB恢复到在线状态并在全部集群中重新分布表。可以按照特定的时间间隔使用gpexpand来重分布表，在空闲时间段使用gpexpand命令重分布表可以降低CPU使用和表锁对数据库操作的影响。另外，可以调整Table的排名来确保大表和关键表按照优先顺序被重分布。

在重分布表的过程中：

- 任何新表或分区的创建都是在所有Instance节点上的，这也刚好是正常的情况。
- 查询会使用所有Instance，即便相关的数据可能还未分布到新的Instance上。
- 正在重分布的表会被锁住，其他任何读写操作都无法访问该表。在重分布完成之后，即恢复正常操作。

排名重分布表

对于大型系统，建议通过在扩展Schema中调整重分布表的排名来空值重分布的顺序。这样就允许首先重分布重要的表从而最小化对系统性能的影响。另外，可用磁盘空间的两也会影响到重分布表的排名，更多信息参考”管理大规模GP系统的重分布”相关章节。

通过更新gpexpand.status_detail表的rank值来调整重分布表的排名，使用psql或者其他支持的客户端连接到GPDB来修改。更新gpexpand.status_detail表的命令例如这样：

```
=> UPDATE gpexpand.status_detail SET rank= 10;
UPDATE gpexpand.status_detail SET rank=1 WHERE fq_name = ‘public.lineitem’;
UPDATE gpexpand.status_detail SET rank=2 WHERE fq_name = ‘public.orders’;
```

第一条命令是将所有表的排名降到10，然后将lineitem的排名设置为1，而orders的排名设置为2。在开始重分布时，lineitem会首先被重分布，接下来的是orders，然后才是gpexpand.status_detail中的其他表。

注意：任何不希望被重分布的表，必须从gpexpand.status_detail表中删除对应的记录。

使用 gpexpand 重分布表

使用gpexpand重分布表

1. 以GPDB系统运行用户(比如gpadmin)登录Master主机。
2. 执行gpexpand命令。作为可选，可以选择使用-d或者-e来定义重分布会话的周期。例如，运行该命令并限制最长60个小时的周期：

   ```
   $ gpexpand -d 60:00:00
   ```

   该命令会一直运行到分布Schema中的最后一张Table被标记为成功的完成分布为止，或者直到指定的持续时间或者结束时间到达为止。每个会话的开始和结束时间，gpexpand命令都会更新到gpexpand.status表中。

监测重分布表

在重分布表过程中的任何时间，都可以查询该扩展Schema。视图gpexpand.expansion_progress提供了当前进度的摘要，包括预估的已经重分布的比率和估计的完成时间。gpexpand.status_detail表可用于查看每张表的重分布状态信息。

查看扩展状态

由于gpexpand.expansion_progress中估算的比率是基于每张表的比率获取的，因此该视图无法准确的计算出比率，直到第一张表的扩展完成为止。每次开始一个新的重分布表的会话时都会重新开始计算。

使用psql或者其他支持的客户端连接到GPDB，通过查询gpexpand.expansion_progress来监测扩展的进度。像下面这样使用SQL命令查询gpexpand.expansion_progress：

```
=# select * from gpexpand.expansion_progress;
             name             |       value
------------------------------+---------------------
Bytes Left                    | 5534842880
Bytes Done                    | 142475264
Estimated Expansion Rate      | 680.75667095996092 MB/s
Estimated Time to Completion  | 00:01:01.008047
Tables Expanded               | 4
Tables Left                   | 4
(6 rows)
```

查看表的状态

表gpexpand.status_detail中存储了每张表的状态、最后更新时间以及其他的有用信息。使用psql或者其他支持的客户端连接到GPDB，通过查询gpexpand.status_detail来监测特定Table的状态。像下面这样使用SQL命令查询gpexpand.status_detail：

```
=> SELECT status, expansion_started, source_bytes FROM
gpexpand.status_detail WHERE fq_name = 'public.sales';
```

```
      status      |      expansion_started      |      source_bytes
------------------+-----------------------------+----------------------
COMPLETED  | 2009-02-20 10:54:10.043869      | 4929748992
(1 row)
```

清除扩展 Schema

在确保扩展操作已经完成之后，扩展Schema可以被安全的删除。要想在GP系统上运行其他的扩展操作，首先必须清除已有的扩展Schema。

1. 以GPDB系统运行用户(比如gpadmin)登录Master主机。
2. 使用-c参数执行gpexpand命令。例如：

```
$ gpexpand -c
```

第十九章：使用 GP MapReduce

- 关于GP MapReduce
- GP MapReduce编程
- 提交MapReduce作业执行
- MapReduce作业故障诊断

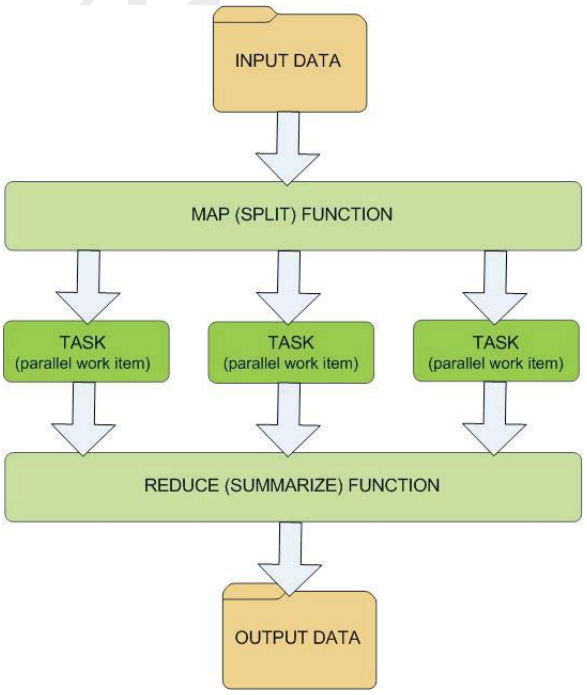
关于 GP MapReduce

MapReduce是一种编程模型，最初由谷歌开发用以在普通计算机集群上处理大量数据。GP MapReduce允许编程者按照MapReduce规范编写Map函数和Reduce函数，提交到GPDB并行处理数据流。GPDB系统处理输入数据的分发细节，在集群中执行程序，处理机器故障，管理机器之间所需的内部通信。

MapReduce 基础

通常，MapReduce是一个简单的数据流编程模型，使用自定义的函数处理数据从一个阶段到下一个阶段。MapReduce函数通常从一个分割为小块文件的大型数据文件开始，连续的数据块称为split(块)，这类似数据库的记录(rows)。通常每个块被处理为(key、value)对并交给Map模块。Map模块会调用自定义的Map函数将每个块处理为新的(key、output_list)对。每个新的(key、output_list)对被交给Reduce模块。Reduce模块收集所有的对，按照key分组，调用自定义的Reduce函数为不同的输入key生成一个输出列表。Map和Reduce模块都会并行利用许多Map和Reduce任务在多个机器上同时工作。

MapReduce引擎以抽象的方式工作，让编程者倾力于所需的数据计算上，而隐藏掉并行架构、任务分发、负载平衡以及故障处理的细节。



GP MapReduce 如何工作

为了使得GP能够处理自定义的Map和Reduce函数，这些函数需要按照GP MapReduce文档要求的格式编写，然后提交到GP MapReduce程序gpmapreduce，由GPDB的并行引擎执行。

GP MapReduce文档定义了组成一个完整的MapReduce作业的构成部分：

- **输入数据** – 输入数据可以来自多种数据库内外的数据源。GPDB支持多种外部数据的文件格式，如同使用SQL访问数据库内部的数据一样。
- **Map函数** – 用户提供使用Python、PERL或者C编写的自定义Map函数。
- **Reduce函数** --用户提供使用Python、PERL或者C编写的自定义Reduce函数或者内置的Reduce函数。
- **输出数据** – 输出可以被存储到数据库中，发送到可写外部表，或者直接输出到标准输出流或外部文件。

一旦按照文档规范定义了MapReduce作业，即可使用gpmapreduce客户端程序将作业提交到GPDB并行数据流引擎开始执行。

GP MapReduce 编程

为了能够提交一个MapReduce作业到GPDB执行，必须按照特定的GP MapReduce规范文档定义一个MapReduce作业。GP MapReduce使用TAML文档框架并实现了自己的YAML模式。更多内容参见”GP MapReduce规范”相关章节。

该节分阶段介绍GP MapReduce规范并为每阶段提供示例：

- 定义输入
- 定义Map函数
- 定义Reduce函数
- 定义输出
- 完整的GP MapReduce规范

这些阶段在GP MapReduce规范文档中的DEFINE章节中有说明。

定义输入

每个MapReduce作业至少需要一个输入数据源。数据源可以是单文件，由GP并行文件分发程序(gpfdist)提供的文件服务，数据库中的表，SELECT语句，或者一个操作系统命令的输出数据。

外部文件(FILE)输入

一个文件输入描述了一个位于GP Segment主机上的单个文件。该文件必须是文本分割格式或者逗号分割(CSV)格式。参见”装载数据到GPDB”相关章节了解更多关于输入数据文件格式要求的信息。如果没有指定COLUMNS，缺省情况下该文件将会整体被作为一个大的文本列处理，列名为value。使用文件输入，必须是GPDB的SUPERUSER才可以执行MapReduce作业。

```
- INPUT:
    NAME: my_file_input
    FILE: seghostname:/var/data/gpfiles/employees.txt
    COLUMNS
        - first_name text
        - last_name text
        - dept text
        - hire_date text
    FORMAT: TEXT
    DELIMITER: '|'
```

GPFDIST输入与文件输入类似，只是文件是由gpfdist提供的服务，而不是单个Segment主机的文件系统。参见”使用GP并行文件服务(gpfdist)”相关涨或了解建立gpfdist文件服务。使用gpfdist文件服务的优势(相对于文件输入来说)在于，可以利用GPDB系统中全部的Instance来读取外部数据文件。除非将参数gp_external_grant_privileges设置为on，否则必须是GPDB的SUPERUSER才可以执行使用GPFDIST作为输入的MapReduce作业。

```
- INPUT:
    NAME: my_distributed_input
    # specifies the host, port and the desired files served
    # by gpfdist. /* denotes all files on the gpfdist server
    GPFDIST:
        - gpfdisthost:8080/*
    COLUMNS
        - first_name text
        - last_name text
        - dept text
        - hire_date text
    FORMAT: TEXT
    DELIMITER: '|'
```

数据库输入

可以使用已经存在GPDB中的数据作为MapReduce的数据源。一个TABLE输入通过简单的指定一个表名来获取其所有的数据。列和数据的类型在指定的Table中已经定义。

```
- INPUT:
    NAME: my_table_input
    TABLE: sales
```

类似的，一个QUERY输入指定一个SELECT语句返回的数据作为数据源。列和数据的类型在源Table中已经定义。

```
- INPUT:
    NAME: my_query_input
    QUERY: SELECT vendor, amt FROM sales WHERE region='usa';
```

操作系统可执行输入

一个EXEC输入允许指定一个SHELL命令或者脚本在所有的GP Segment上执行来作为数据源。所有Instance处理结果输出的全部结果组成了数据源。这些命令会在所有节点的活动Instance上被执行。例如，如果节点上有4个Primary在运行，该命令在每个节点上将运行4次。数据由MapReduce作业在每个Instance上运行时命令的输出组成。所有Instance并行执行该命令。如果命令调用了一个脚本或者程序，该脚本或程序必须位于所有GP Segment主机上。

```
- INPUT:
    NAME: my_query_input
    EXEC: /var/load_scripts/get_log_data.sh
    COLUMNS
        - url text
        - date timestamp
    FORMAT: TEXT
    DELIMITER: '|'
```

如果在命令中用到了环境变量(比如$PATH)，需要注意的是，命令是从数据库执行而不是从登录的SHELL。因此，当前用户的.bashrc或者.profile文件不会被装载。不过，在EXEC定义语句中，可以根据需要设置环境变量，比如：

```
EXEC 'export PATH=/var/myscripts:$PATH; get_log_data.sh;'
```

要使用这个输入类型，必须是GPDB的SUPERUSER并且在Master配置文件postgresql.conf中将服务器配置参数gp_external_enable_exec设置为on。

定义 Map 函数

借用数据库的术语来说，一个Map函数根据一个输入行(一组分配到参数的值)，生成0个或多个输出行。缺省情况下，输入输出都会有两个文本类型的参数key和value。然而，GP MapReduce允许使用任何SQL表定义中的类型作为输入输出参数。输入的格式在MAP说明的PARAMETERS定义中指定，而输出的格式在RETURNS定义中指定。RETURNS的定义要求使用SQL的数据类型，以用于后续的步骤作为相关SQL查询的输入。如不确定分，可选择SQL的text类型，因为不管是PERL还是Python都能够很好的解读text字符串。

MAP的定义还包括FUNCTION定义，其提供了函数的代码，通过LANGUAGE指定特定的脚本语言。目前支持的语言为PERL和PYTHON。

一个典型的Map函数定义使用PARAMETERS的值通过某种方式的计算产生匹配RETURNS定义的输出值。因此，定义Map函数的主题就是要知道如何使用脚本语言访问PARAMETERS的数据，以及为RETURNS准备需要的输出数据。

PERL语言的Map函数

在PERL风格下，Map函数的PARAMETERS参数可以通过@_获得取值列表。PERL函数的一个典型步骤是首先将参数通过赋值语句抽取到本地变量。Map函数的输出必须是PERL的hash表，RETURNS定义中的每个参数都有一个hash

键。输出通常使用称为return_next的特定PERL函数返回到MapReduce运行引擎。该函数和通常的return的行为类似，除了再次调用时会生成另一个输出行，在最后一个return_next被执行之后，该函数才会处理接下来的行。这种编程风格使得单行输入多行输出(每行通过PERL的循环调用return_next返回)成为可能。当没有更多的结果返回，标准的PERL调用return undef将告诉MapReduce运行引擎处理下一个输入记录，重新从Map函数的顶端开始执行。

如果已知Map函数对于每个输入值仅返回单个输出值，可以指定Map定义的描述为MODE: SINGLE，并且调用标准的PERL return来返回hash散列，而不使用return_next。

下面的Map函数例子，将逗号分割的行转换为多行，每个值一行。注意FUNCTION后的竖线(YAML的标记)：表明随后缩进的行作为单一的字符串来对待。

```
- MAP:
    NAME: perl_splitter
    LANGUAGE: PERL
    PARAMETERS: [key, value]
    RETURNS: [key text, value text]
    FUNCTION: |
        my ($key, $value) = @_;
        my @list = split(/,/, $value);
        for my $item(@list) {
            return_next({"key" => $key, "value" => $item});
        }
        return undef;
```

Map Functions in Python

在Python中，Map函数的参数PARAMETERS可以直接作为本地参数来使用。而没有PERL风格中赋值的必要。Map函数的输出必须是Python字典，RETURNS定义中的每个参数都有一个key。输出通过Python的yield返回到MapReduce运行引擎，再次调用Map函数生成其他的输出时，运行引擎会在最后一次yield被执行后会自动处理下一行输入。这种编程风格使得单行输入多行输出(每行通过Python的循环调用yield返回)成为可能。当没有更多的结果被返回，该Python代码将会简单的跳到脚本的结尾。当没有更多的结果返回，MapReduce运行引擎将会处理下一个输入记录，重新从Map函数的顶端开始执行。

如果已知Map函数对于每个输入值仅返回单个输出值，可以指定Map定义的描述为MODE: SINGLE，并且调用标准的Python return来返回hash散列，而不使用yield。

下面的Map函数例子，将逗号分割的行转换为多行，每个值一行。注意FUNCTION后的竖线(YAML的标记)：表明随后缩进的行作为单一的字符串来对待。

```
- MAP:
    NAME: py_splitter
    LANGUAGE: PYTHON
    PARAMETERS: [key, value]
    RETURNS: [key text, value text]
    FUNCTION: |
        list = value.split(',')
        for item in list:
            yield {'key': key, 'value': item}
```

定义 Reduce 函数

Reduce函数将匹配的多个输入值生成一个独立的缩减的值。GPDB提供了多种预定义的REDUCE函数可供执行，这些都是针对名为value的列进行操作：

- IDENTITY – 返回(key, value)对并不做任何改变
- SUM – 计算数字的总和
- AVG – 计算数字的平均值
- COUNT – 计算数据的计数
- MIN – 计算数字的最小值
- MAX – 计算数字的最大值

要使用这些预定义的REDUCE函数，可以简单的通过其名称在MapReduce定义中的EXECUTE部分直接使用。例如：

```
EXECUTE
    - RUN
        SOURCE: input_or_task_name
        MAP: map_function_name
        REDUCE: IDENTITY
```

编辑自定义的Reduce函数比Map函数涉及的内容要多一些，因为Reduce函数必须定义以符合多行输入数据，而不仅仅是单行数据。要实现这么目的，必须定义与REDUCE相关的TRANSITION函数，其针对每行输入都会调用。为了在调用TRANSITION函数期间能够记住相关信息，必须使用名为state的变量作为第一个输入参数。在数据被缩减之前，state变量将被以INITIALIZE定义的值初始化。state的值必须是SQL数据类型。例如，SQL文本字符串。在一组处理期间，state变量记录着TRANSITION函数最近的返回值。在TRANSITION函数处理完一组记录之后，state变量将被传给FINALIZE函数，FINALIZE函数可以返回多记录(通过PERL的return_next或者Python的yield)。输出的每一行都应该是缩减后的散列。

缺省情况下，Reduce的参数为(key, value)对。然而对于自定义Reduce函数来说，任意数量的列都可以传入。KEYS定义了用于将输入划分到不同子集以达到缩减目的的列，缺省的KEYS定义是称为key的列。如果缺少KEYS的定义，key将被定义为TRANSITION函数PARAMETERS未提及部分的参数(如此拗口译者也不太明白什么意思)。

为了优化性能，还可以选择定义一个CONSOLIDATE函数，其将多个state组合到一个state变量。这将允许在GPDB的机器之间发送state变量，而不是交换输入数据，从而大大降低网络的负载。CONSOLIDATE类似于TRANSITION的结构，每次将2个state变量生成一个state变量。

下面是一个PERL Reduce函数，定义了计算所有正数值的平均值：

```
- REDUCE:
    NAME: perl_pos_avg
    TRANSITION: perl_pos_avg_trans
    CONSOLIDATE: perl_pos_avg_cons
    FINALIZE: perl_pos_avg_final
    INITIALIZE: '0,0'
    KEYS: [key]

TRANSITION:
    NAME: perl_pos_avg_trans
    PARAMETERS: [state, value]
    RETURNS: [state text]
    LANGUAGE: perl
    FUNCTION: |
        my ($state, $value) = @_;
        my ($count, $sum) = split(/,/, $state);
        if ($value > 0) {
            $sum += $value;
            $count++;
            $state = $count . "," . $sum;
        }
        return $state;

- CONSOLIDATE:
    NAME: perl_pos_avg_cons
    PARAMETERS: [state, value]
    RETURNS: [state text]
    LANGUAGE: perl
    FUNCTION: |
        my ($state, $value) = @_;
        my ($scount, $ssum) = split(/,/, $state);
        my ($vcount, $vsum) = split(/,/, $value);
        my $count = $scount + $vcount;
        my $sum = $ssum + $vsum;
        return ($count . "," . $sum);
```

```
- FINALIZE:
    NAME: perl_pos_avg_final
    PARAMETERS: [state]
    RETURNS: [value float]
    LANGUAGE: perl
    FUNCTION: |
        my ($state) = @_;
        my ($count, $sum) = split(/,/, $state);
        return_next ($count*1.0/$sum);
        return undef;
```

定义输出

定义OUTPUT是可选的。如果没有定义输出，缺省的是将最终结果输出到GP MapReduce客户端的标准输出。还可以直接输出到GP MapReduce客户端的文件中，或者通过定义OUTPUT输出到数据库中。

表输出

TABLE输出定义了有MapReduce生成的最终结果最终输出到数据库的表中。缺省情况下，TABLE指定的表名如果在数据库中不存在将会被自动创建。如果表在数据库中不存在，必须定义一个MODE以指定追加(APPEND)输出到该表还是重新创建(REPLACE)该表。缺省情况下，该表会以REDUCE的keys作为分部键，作为可选，还可以通过KEYS来指定分部键。

```
- OUTPUT:
    NAME: gpmr_output
    TABLE: wordcount_out
    KEYS:
        - value
    MODE: REPLACE
```

输出表还可以是可写外部表，其将数据传输到文件或者其他程序以完成进一步的处理。如果使用可写外部表，该表必须在运行MapReduce之前就创建好。对于可写外部表来说，MODE必须设置为APPEND。

文件输出

FILE定义了GP MapReduce客户端的文件位置以接受输出。在MapReduce运行时指定名称的文件将会被创建。如果文件已经存在等，将会返回一个错误。

```
- OUTPUT:
    NAME: gpmr_output
    FILE: /var/data/mapreduce/wordcount.out
```

定义任务

TASK定义是可选项，但对于多级MapReduce是很有用的。一个任务定义了完整的端到端INPUT/MAP/REDUCE阶段一个完整的GP MapReduce作业管道。一

旦定义了，一个TASK对象可以被作为其他进一步处理的输入。

例如，如果定义了一个表输入为documents而另外一个为keywords。每个表的输入通过自己的MAP函数document_map和keyword_map来处理。如果想要使用这些处理阶段的结果作为其他进一步处理的MapReduce输入，可以像下面这样定义2个任务：

```
- TASK:
    NAME: document_prep
    SOURCE: documents
    MAP: document_map
- TASK:
    NAME: keyword_prep
    SOURCE: keywords
    MAP: keyword_map
```

这些命名的任务在后续的处理阶段可以作为输入来使用。该例中，定义一个SQL查询作为输入，其用到了之前定义的两个任务(document_prep和keyword_prep)并做join关联。

```
- INPUT:
    NAME: term_join
    QUERY: |
        SELECT doc.doc_id, kw.keyword_id, kw.term, kw.nterms,
            doc.positions as doc_positions, kw.positions as kw_positions
        FROM document_prep doc INNER JOIN keyword_prep kw
            ON (doc.term = kw.term)
```

组装完整的 MapReduce 定义

一旦按照GP MapReduce规范定义了MapReduce所有阶段，还必须定义一个执行部分以指定最终的INPUT/MAP/REDUCE阶段。在EXECUTE部分定义的所有对象名称都是之前按照GP MapReduce规范已经定义好的。

```
EXECUTE:
  - RUN:
    SOURCE: input_or_task_name
    TARGET: output_name
    MAP: map_function_name
    REDUCE: reduce_function_name
```

提交 MapReduce 作业执行

一旦按照GP MapReduce规范定义了MapReduce程序，还必须通过gpmapreduce客户端程序提交该程序到GPDB以得到执行。该客户端程序类似psql，要求提供连接信息，如数据库名称、连接的用户名、主机名和端口等。可以使用命令行或者环境变量来提供，如$PGDATABASE、$PGUSER、$PGHOST和$PGPORT。例如：

```
gpmapreduce -h gpmasterhost -p 54321 -f my_gpmr_spec.yml mydatabase
```

在数据库中创建语言

在执行Map和Reduce函数时，GP MapReduce会使用过程语言将这些函数构建到数据库中。GP需要提前将这些语言在数据库中创建，以运行MapReduce作业。使用CREATE LANGUAGE命令在数据库中为要执行的MapReduce作业创建需要的语言。创建语言必须使用数据库的超级用户。例如，创建PERL过程语言：

```
$ psql -c 'CREATE LANGUAGE plperl;'
```

以及创建Python过程语言：

```
$ psql -c 'CREATE LANGUAGE plpythonu;'
```

安装库文件(自定义C函数)

如果使用自定义C语言的Map或Reduce函数，函数定义的库文件必须安装到GP所有主机相同路径(Master和Segment)。例如，在MapReduce的YAML定义中存在一个C函数的定义：

```
DEFINE:
    - TRANSITION:
        NAME: int4_accum
        PARAMETERS: [state int8, value int4]
        RETURNS: [state int8]
        LANGUAGE: C
        LIBRARY: $libdir/gpmr.so
        FUNCTION: int4_accum
```

由LIBRARY指定的$libdir/gpmr.so文件必须在所有主机的$GPHOME/lib目录(缺省的库目录)下存在。FUNCTION属性指定了该库文件中要被执行的函数名。

MapReduce 作业故障诊断

本节讲述GP MapReduce的一些常见执行错误以及如何解决这些错误。

语言不存在

症状：

```
ERROR: language "pl*" does not exist
HINT: Use CREATE LANGUAGE to load the language into the database.
```

解释：

在执行Map和Reduce函数时，GP MapReduce会使用过程语言将这些函数构建到数据库中。GP需要提前将这些语言在数据库中创建，以运行MapReduce作业。

解决方案：

如果在连接的数据库中试图使用未创建的语言，应该以管理员身份在执行之前先创建相应的语言。例如：

```
psql database_name -c 'CREATE LANGUAGE plperl;'
```

一些语言可能需要SUPERUSER权限才可以使用。

通用 Python 迭代器错误

症状：

ERROR: plpython: function “function_name” error fetching next item from iterator

DETAIL:

<type 'exceptions.IOError'>: [Errno 2] No such file or directory: '/tmp/file/doesnt/exist' (, line 39)

解释：

该异常来自Python，在迭代函数(比如使用yield的函数)没有错误时发生。这通常表明在Python代码中存在bug。

解决方案：

最简单的调试办法是在函数体外包装一个错误捕获结构。例如：

```
FUNCTION: |
    try:
        ...
        user code
        ...
    except Exception, e:
        plpy.warning('my function name:' + str(e))
```

这将生成一条便于调试的警告消息。

函数定义使用了错误的模式

症状：

ERROR: set-returning PERL function must return reference to array or use return_next

---

ERROR: composite-returning PERL function must return reference to hash

---

ERROR: returned sequence's length must be same as tuple's length

---

ERROR: no attribute named "key"

HINT: to return null in specific column, let returned object to have attribute named after column with value None

---

解释：

在GP MapReduce函数中有两种主要模式：

- MODE: SINGLE 每行接收返回单行
- MODE: MULTI 每行接收可能返回0~N的任意行

TRANSITION函数和CONSOLIDATE函数仅支持SINGLE模式。因为他们属于有限状态函数，且必须返回下一个状态。

MAP函数和FINALIZE函数支持两种模式且缺省为MULTI模式。因为这样更具有通用性。

解决方案：

在SINGLE模式的函数中必须使用return方法来返回单行数据。

在MULTI模式中，最好作为通用函数来编写。在Python中使用yield。在PERL中使用return_next。例如：

```
- MAP:
    NAME: perl_single
    MODE: SINGLE
    PARAMETERS: [key text, value text]
    RETURNS: [key text, value text]
    LANGUAGE: perl
    FUNCTION: |
        my ($key, $value) = @_;
        return {'key' => $key, 'value' => $value}

- MAP:
    NAME: perl_multi
    MODE: MULTI
    PARAMETERS: [key text, value text]
    RETURNS: [key text, value text]
    LANGUAGE: perl
    FUNCTION: |
        my ($key, $value) = @_;
        for my $i (0..10) {
            return_next {'key' => $key, 'value' => $value}
        }
        return undef

- MAP:
    NAME: python_single
    MODE: SINGLE
    PARAMETERS: [key text, value text]
    RETURNS: [key text, value text]
    LANGUAGE: python
    FUNCTION: |
        return {'key': key, 'value': value}

- MAP:
    NAME: python_multi
    MODE: MULTI
    PARAMETERS: [key text, value text]
    RETURNS: [key text, value text]
    LANGUAGE: python
    FUNCTION: |
        try:
            for i in range(0,10):
                yield {'key': key, 'value': value}
```

```
        except Exception, e:
            plpy.warning('python_multi: ' +str(e))
```

当在MULTI模式函数中视图使用return时通常会出现”returned sequence’s length ...”错误。在这种情况下，单个的行不再以独立的行列返回，而是以一组行的列返回。这样每个列的宽度将会变得过小而取法容难全部行的列，因此会得到这种错误信息。

一个不太常见的发生同样错误的场景是，返回的列数量与声明的不一致。例如：

```
- MAP:
    NAME: python_error
    MODE: SINGLE
    PARAMETERS: [key text, value text]
    RETURNS: [key text, value text]
    LANGUAGE: python
    FUNCTION: |
        return {'key': key}
```

因为该函数声明了返回一个key和一个value，但实际上值返回了一个key，这同样或得到一个’returned sequence’s length ...’异常。

第二十章：日常系统维护任务

和其他的数据库软件一样，GPDB需要一些定期的维护任务以达到最佳性能。这里讨论的任务是必要的，他们是可重复的，所以可以使用标准的UNIX工具如cron脚本来调度。不过，设置合适的脚本以及确保它们成功的执行是数据库管理员的责任。

- 日常Vacuum和Analyze
- 日常索引重建
- 管理GPDB日志文件

日常 Vacuum 和 Analyze

由于GPDB使用的是MVCC事务并发模型，被删除或更新的数据行依然占据着物理磁盘空间，即便它们对于新的事务已经不可见。如果数据库有大量的更新和删除，会产生大量过期记录。VACUMM命令还会收集表级别的统计信息，如行数和页面数，因此对于AO表VACUUM也是有必要的。VACUUM对于AO表的操作很快，因为其没有空间需要被回收。

事务 ID 管理

GP的MVCC事务依赖于事务ID(XID)的数字以确定其他事务的可见性。但是事务ID的值是有限的，长时间(超过4百万个事务)运行的GP系统将会出现事务ID循环的问题：XID计数器回到0，所有过去的事务突然变成将来的事务—这意味着不可见的记录变得可见。为了避免这个问题，在每个数据库每2百万个事务的时候，对每张表执行VACUUM是很有必要的。

系统目录维护

大量的CREATE和DROP命令会导致系统表的迅速膨胀，以至于影响系统性能。例如，在大量的DROP TABLE语句之后，扫描系统元数据的操作性能会大大降低。根据不同的系统，在数千个DROP TABLE语句之后就可能会发生性能下降的情况。

GP建议定期的执行系统目录维护操作，以回收因删除对象而导致的空间占用。如果很长时间没有执行，可能需要执行一个更密集的操作来清理系统目录。本节讲述这两种操作。

定期系统目录维护

GP建议定期的在系统目录上执行VACUUM清理删除对象占用的空间。如果DROP语句是常规的数据库操作，每天在空闲时间段运行系统目录维护操作将是很有必要且安全的。也可以在系统处于常规运行状态时做这些操作。

下面的示例脚本是对GPDB系统目录进行VACUUM操作：

```
#!/bin/bash
DBNAME="<database_name>"
```

```
VCOMMAND="VACUUM ANALYZE"
psql -tc "select '$VCOMMAND' || ' pg_catalog.' || relname || ';' from pg_class a,pg_namespace b
where a.relnamespace=b.oid and b.nspname= 'pg_catalog' and a.relkind='r'" $DBNAME | psql -a
$DBNAME
```

密集系统目录维护

如果很长时间都没有执行系统目录维护，系统表可能会被过期空间严重膨胀，导致即便简单的元数据操作也需要很长的等待时间。比如列出用户表的操作需要等待1到2秒，例如使用psql的\d元命令，这可以表明系统目录的膨胀。

如果发现了系统目录膨胀的迹象，一个密集的系统目录维护操作必须在预定的停机期间使用VACUUM FULL来完成执行。在这段时间必须停止系统上的所有操作，因为在系统目录维护过程中会对这些系统目录采用独占式的排它锁。

定期的运行系统目录维护可以避免较高成本的密级操作。

为优化查询进行回收和分析

GPDB根据数据库的统计信息使用基于成本的查询规划器。精确的统计信息使得查询规划器更好的评估选择性和检索的行数从而选择最有效的查询计划。ANALYZE命令收集查询规划器需要用到的列统计信息。VACUUM和ANALYZE操作可以在同一个命令中一起运行。例如：

```
=# VACUUM ANALYZE mytable;
```

日常重建索引

对于B-tree索引，新重建的索引通常比存在较多更新的表访问更快，因为逻辑上相邻的页面在新重建的索引中也是逻辑上相邻的。定期的重建索引对于改善访问速度是值得的。同样，如果全部索引键的一部分被删除了，索引页面也会存在过期空间。重建索引可以收回这些过期的空间。在GPDB中，删掉索引(DROP INDEX)然后重新创建(CREATE INDEX)通常比REINDEX命令更快。

当更新索引列时，Bitmap索引不会被更新。如果更新了一张有Bitmap索引的表，为了保持最新的信息，应该删除并重新创建该索引。

管理 GPDB 日志文件

- 数据库服务日志文件
- 管理程序的日志文件

数据库服务日志文件

GPDB的日志输出量很大(尤其在较高的调试级别)而且不需要无限期的保存这些日志。管理员需要定期的滚动日志文件，从而可以保证新的日志文件被使用而旧的日志文件可以被定期的删除。

GPDB在Master和所有的Segment实例上开启了日志文件滚动。每天的日志文件放在每个Instance数据目录的pg_log目录下并使用约定命名方式：

gpdb-YYYY-MM-DD.log

尽管日志是按天滚动的，但它们不会被自动清空或删除。管理员需要通过一些脚本或程序来定期的清理各实例pg_log目录下的旧日志文件。

管理程序的日志文件

GP管理程序的日志文件缺省位于~/gpAdminLogs目录下。管理程序日志文件的约定命名方式为：

<script_name>_<date>.log

日志记录的格式为：

<timestamp>:<utility>:<host>:<user>:[INFO|WARN|FATAL]:<message>

在程序运行时，其日志文件会追加到特定的日期文件。

第廿一章：系统表参考

这里介绍系统表和GPDB视图。所有与GPDB并行特性相关的系统表以gp_作为表的前缀。以pg_作为前缀的是标准的PostgreSQL系统表(这些表在GPDB中同样需要使用)，或者GP添加的用以加强PostgreSQL在数据仓库工作负载方面的功能。需要注意的是，这些全局系统目录对于GPDB来说是存储在Master实例上的。

系统表

- gp_configuration (不再使用，由gp_segment_configuration取代)
- gp_configuration_history
- gp_db_interfaces
- gp_distribution_policy
- gp_fastsequence
- gp_fault_strategy
- gp_global_sequence
- gp_id
- gp_interfaces
- gp_master_mirroring
- gp_persistent_database_node
- gp_persistent_filespace_node
- gp_persistent_relation_node
- gp_persistent_tablespace_node
- gp_relation_node
- gp_san_configuration
- gp_segment_configuration
- gp_version_at_initdb
- gpexpand.status
- gpexpand.status_detail
- pg_aggregate
- pg_am
- pg_amop
- pg_amproc
- pg_appendonly
- pg_appendonly_alter_column (4.2版本未实现)
- pg_attrdef
- pg_attribute
- pg_attribute_encoding
- pg_auth_members
- pg_authid
- pg_autovacuum
- pg_cast
- pg_class
- pg_compression
- pg_constraint

- pg_conversion
- pg_database
- pg_depend
- pg_description
- pg_exttable
- pg_filespace
- pg_filespace_entry
- pg_foreign_data_wrapper (4.2版本未实现)
- pg_foreign_server (4.2版本未实现)
- pg_foreign_table (4.2版本未实现)
- pg_index
- pg_inherits
- pg_language
- pg_largeobject
- pg_listener
- pg_locks
- pg_namespace
- pg_opclass
- pg_operator
- pg_partition
- pg_partition_encoding
- pg_partition_rule
- pg_pltemplate
- pg_proc
- pg_resourcetype
- pg_resqueue
- pg_resqueuecapability
- pg_rewrite
- pg_shdepend
- pg_shdescription
- pg_stat_activity
- pg_stat_last_operation
- pg_stat_last_shoperation
- pg_statistic
- pg_tablespace
- pg_trigger
- pg_type
- pg_type_encoding
- pg_user_mapping (4.2版本未实现)
- pg_window

System Views

GPDB还包含如下目前的PostgreSQL中没有的系统视图。

- gp_distributed_log
- gp_distributed_xacts

- gp_pgdatabase
- gp_resqueue_status
- gp_transaction_log
- gpexpand.expansion_progress
- pg_max_external_files (使用file协议的外部表每个Segment Host可用文件数量)
- pg_partition_columns
- pg_partition_templates
- pg_partitions
- pg_resqueue_attributes
- pg_resqueue_status 已弃用 (由gp_toolkit.gp_resqueue_status取代)
- pg_roles
- pg_stat_operations
- pg_stat_partition_operations
- pg_stat_resqueues
- pg_user_mappings (4.2版本未实现)

关于PostgreSQL的标准系统视图(在GPDB中同样需要使用)，查看下面的PostgreSQL文档章节：

- System Views
- Statistics Collector Views
- The Information Schema

---

gp_configuration_history

该表包含关于系统故障诊断和恢复操作的系统变化信息。fts_probe进程记录日志到这个表中，还有一些相关的管理命令，如gpcheck、gprecoverseg和gpinitsystem。例如，在添加一个新的Instance和Mirror到系统时，这些事件的日志会被记录到该表中。

该表中存储的时间描述信息有助于Greenplum技术人员诊断严重的系统问题。

该表仅位于Master主机。其定义在pg_global表空间，在系统中的所有数据库之间都是共享可访问的。

字段	类型	参考	描述
time	timestamp with time zone		事件记录的时间
dbid	smallint	gp_segment_configuration.dbid	实例的唯一标识符
desc	text		事件的描述信息

---

gp_distributed_log

此视图包含分布式事务和关联的本地事务的状态信息。一个分布式事务涉及到Segment实例上的数据更新。Greenplum的分布式事务管理确保节点之间保持同步。该视图允许查看分布式事务的状态。

字段	类型	参考	描述
segment_id	smallint	gp_segment_configuration.content	Master 始终为-1
dbid	smallint	gp_segment_configuration.dbid	实例的唯一标识符
distributed_xid	xid		全局事务标识符
distributed_id	text		系统为分布式事务分配的标识符

status	text		分布式事务的状态(Committed 或 Aborted)
local_transaction	xid		本地事务标识符

gp_distributed_xacts

此视图包含Greenplum数据库分布式事务的信息。一个分布式事务涉及到Segment实例上的数据更新。Greenplum的分布式事务管理确保节点之间保持同步。该视图允许查看当前活动的连接和相关的分布式事务。

字段	类型	参考	描述
distributed_xid	xid		跨越 GPDB 节点的分布式事务标识符
distributed_id	text		分布式事务分配的标识符-，分为 2 部分—唯一的时间戳和分布式事务序号
state	text		关于分布式事务会话的当前状态
gp_session_id	int		与事务相关的 GPDB 会话标识符
xmin_distributed_snapshot			该事务开始时打开的分布式事务的最小数量。用做 MVCC 分布式快照。

gp_db_interfaces

此表包含Segment与网络接口的关系信息。结合gp_interfaces表数据，系统用以优化各种目的网络使用，包括故障检测。

字段	类型	参考	描述
dbid	smallint	gp_segment_configuration.dbid	实例的唯一标识符
interfaceid	smallint	gp_interfaces.interfaceid	系统分配的网络接口 ID
priority	smallint		此 Segment 的网络接口优先级

gp_distribution_policy

此表包含GPDB表数据分布策略的信息。该表仅位于Master实例上。此表不是全局共享的，这意味着每个DB都有自己的这样一张表。

字段	类型	参考	描述
localoid	oid	pg_class.oid	表对象的标识符(OID)
attrnums	smallint[]	pg_attrbute.attnum	分部键的列序号数组

gp_fastsequence

此表包含AO列存表的索引信息。AO列存表的文件片段用于跟踪最大记录数。

字段	类型	参考	描述
objid	oid	pg_class.oid	pg_aoseg.pg_aocsseg_*表用以跟踪 AO 文件片段的对象标识符
objmod	bigint		Object modifier(不知如何翻译)
last_sequence	bigint		该对象最后使用的序列值

gp_fault_strategy

此表指定故障操作。

字段	类型	参考	描述
fault_strategy	char		当节点失效出现时的故障操作：

			n = 未定义 f = 基于文件的故障切换 s = 基于网络的故障切换

gp_global_sequence

此表包含事务日志的序列位置，被文件复制进程用于决定在主备节点之间复制文件块。

字段	类型	参考	描述
sequence_num	bigint		事务日志的序列位置

gpexpand.expansion_progress

此视图包含系统扩展操作状态的信息。该视图提供了对表重分布完成时间的评估计算。特定表的扩展状态存储在gpexpand.status_detail表中。

字段	类型	参考	描述
name	text		包含的数据域有： 剩余的字节数 已处理字节数 评估的扩展率 评估的完成时间 已扩展的表 剩余的表
value	text		进度的数据值。 例如： 评估扩展率 - 9.75667095996092 MB/s

gpexpand.status

此表包含系统扩展操作的信息。特定表的扩展状态存储在gpexpand.status_detail表中。

一般的扩展操作不需要修改该表中存储的数据。

字段	类型	参考	描述
status	text		跟踪扩展操作的状态。有效的值为： SETUP SETUP DONE EXPANSION STARTED EXPANSION STOPPED COMPLETED
updated	timestamp with time zone		最后一次更新状态的时间戳

gpexpand.status_detail

此表包含系统扩展操作涉及的表信息。可以通过该表查看正在扩展的表，或者查看已经完成扩展的的开始结束时间。

一般的扩展操作不需要修改该表中存储的数据。

字段	类型	参考	描述
dbname	text		表所属的数据库

fq_name	text		表的完整限定名称
schema_oid	oid		表所在数据库中模式的 OID
table_oid	oid		表的 OID
distribution_policy	smallint()		分部键的列序号数组
distribution_policy_names	text		HASH 分布键的名称
distribution_policy_coloids	text		分部键的列 ID 串
storage_options	text		目前不支持修改该属性
rank	int		决定扩展顺序的等级，该值越小的表将首先被扩展
status	text		表的扩展状态，有效值为： NOT STARTED IN PROGRESS FINISHED
last updated	timestamp with time zone		表扩展最后一次修改的时间戳
expansion started	timestamp with time zone		表扩展的开始时间戳。该字段只有在成功扩展之后才倍填充
expansion finished	timestamp with time zone		表扩展完成的时间戳
source bytes			原表所占磁盘空间的尺寸。由于表的膨胀和不同数量的扩展字段，该值不一定与最终扩展的尺寸等效，

gp_id

此系统表标识数据库系统名称和系统节点数。该表在特定的节点上(Segment和Master)都有本地值。其定义在pg_global表空间，在系统中的所有数据库之间都是共享可访问的。

字段	类型	参考	描述
gpname	name		GP 数据库系统的名称
numsegments	integer		GP 数据库系统的节点数
dbid	integer		实例的唯一标识符
content	integer		数据在节点上的标识符。主节点与镜像节点具有相同的 content 值。对于一个节点来说，值为 0~N，N 为 GP 数据库的节点数。而对于 Master 来说该值为-1

gp_interfaces

此表包含节点上的网络接口信息。结合gp_db_interfaces表数据，系统用以优化各种目的网络使用，包括故障检测。

字段	类型	参考	描述
interfaceid	smallint		系统分配的网络接口 ID
address	name		包含网络接口的节点主机名。可以是 IP 地址或者主机名。
status	smallint		网络接口的状态，0 表明该接口不可用

gp_master_mirroring

此表包含Standby Master主机和相关WAL复制进程的信息。如果在Standby Master上的同步进程(gpsyncagent)失败了，这对于系统用户来说可能并不总是显而易见的。GP数据库管理员可以监测该目录表以查看Master当前是否处于完全同步状态。

字段	类型	参考	描述
summary_state	text		Master 与 Standby 之间同步进程的状态。状态为‘Synchronized’或‘Not Synchronized’，未配置的情况下为‘ Not Configured’
detail_state	text		如果状态为‘Not Synchronized’，该字段包含一些异常信息

log_time	timestampz		Master 最后发送日志到 Standby 的时间戳
error_message	text		如果状态为'Not Synchronized'，该字段包含尝试同步的失败信息

gp_persistent_database_node

此表包含与数据库对象相关的文件系统对象信息。这些信息用以确保系统日志和文件系统的同步关系。这些信息也用于主实例与镜像实例之间的文件同步。

字段	类型	参考	描述
tablespace_oid	oid	pg_tablespace.oid	表空间对象标识符
database_oid	oid	pg_database.oid	数据库对象标识符
persistent_state	smallint		0 – 空闲 1 – 待创建 2 – 已创建 3 – 待删除 4 – 取消创建 5 – 即时待创建 6 – 容量负载待创建
mirror_existence_state	smallint		0 – 无 1 – 未镜像 2 – 镜像待创建 3 – 镜像已创建 4 – 镜像在创建之前失效 5 – 镜像在创建时失效 6 – 镜像待删除 7 – 镜像残余
parent_xid	integer		全局事务标识符
persistent_serial_num	bigint		位于事务日志文件块中的日志序列数
previous_free_tid	tid		GP 数据库内部用以管理文件系统

gp_persistent_filespace_node

此表跟踪文件空间对象事务状态相关的文件系统对象状态。这些信息用于确保系统日志的状态和文件系统之间的同步。这些信息用于主实例与镜像实例之间的文件同步。

字段	类型	参考	描述
filespace_oid	oid	pg_filespace.oid	文件空间的对象标识符
db_id_1	smallint		主节点标识符
location_1	text		主节点文件系统位置
db_id_2	smallint		镜像节点标识符
location_2	text		镜像节点文件系统位置
persistent_state	smallint		0 – 空闲 1 – 待创建 2 – 已创建 3 – 待删除 4 – 取消创建 5 – 即时待创建 6 – 容量负载待创建
mirror_existence_state	smallint		0 – 无 1 – 未镜像 2 – 镜像待创建 3 – 镜像已创建 4 – 镜像在创建之前失效 5 – 镜像在创建时失效 6 – 镜像待删除

			7 – 镜像残余
parent_xid	integer		全局事务标识符
persistent_serial_num	bigint		位于事务日志文件块中的日志序列数
previous_free_tid	tid		GP 数据库内部用以管理文件系统

gp_persistent_relation_node

此表跟踪与数据库对象(表、视图、索引等)的事务状态相关的文件系统对象状态。这些信息用以确保系统日志与文件系统的磁盘文件之间的同步。这些信息用于主实例与镜像实例之间的文件同步。

字段	类型	参考	描述
tablespace_oid	oid	pg_tablespace.oid	表空间对象标识符
database_oid	oid	pg_database.oid	数据库对象标识符
relfilenode_oid	oid	pg_class.relfilenode	相关文件的对象标识符
segment_file_num	integer		对于 AO 表来说，AO 片段文件的数量
relation_storage_manager	smallint		HEAP 存储或者 AO 存储
persistent_state	smallint		0 – 空闲 1 – 待创建 2 – 已创建 3 – 待删除 4 – 取消创建 5 – 即时待创建 6 – 容量负载待创建
mirror_existence_state	smallint		0 – 无 1 – 未镜像 2 – 镜像待创建 3 – 镜像已创建 4 – 镜像在创建之前失效 5 – 镜像在创建时失效 6 – 镜像待删除 7 – 镜像残余
parent_xid	integer		全局事务标识符
persistent_serial_num	bigint		位于事务日志文件块中的日志序列数
previous_free_tid	tid		GP 数据库内部用以管理文件系统

gp_persistent_tablespace_node

此表跟踪表空间对象事务状态相关的文件系统对象状态。这些信息用以确保系统日志与文件系统的磁盘文件之间的同步。这些信息用于主实例与镜像实例之间的文件同步。

字段	类型	参考	描述
filespace_oid	oid	pg_filespace.oid	文件空间对象标识符
tablespace_oid	oid	pg_tablespace.oid	文件空间对象标识符
persistent_state	smallint		0 – 空闲 1 – 待创建 2 – 已创建 3 – 待删除 4 – 取消创建 5 – 即时待创建 6 – 容量负载待创建
mirror_existence_state	smallint		0 – 无 1 – 未镜像 2 – 镜像待创建

			3 – 镜像已创建 4 – 镜像在创建之前失效 5 – 镜像在创建时失效 6 – 镜像待删除 7 – 镜像残余
parent_xid	integer		全局事务标识符
persistent_serial_num	bigint		位于事务日志文件块中的日志序列数
previous_free_tid	tid		GP 数据库内部用以管理文件系统

gp_relation_node

此表包含数据库对象(表、视图、索引等)相关的文件系统对象的信息。

字段	类型	参考	描述
relfilenode_oid	oid	pg_class.relfilenode	相关文件的对象标识符
segment_file_num	integer		对于 AO 表来说，AO 片段文件的数量
persistent_tid	tid		GP 数据库内部用作管理持久化文件系统对象
persistent_serial_num	bigint		位于事务日志文件块中的日志序列数

gp_san_configuration

此表包含SAN故障切换的挂载点信息。

字段	类型	参考	描述
mounted	smallint		
active_host	char		
san_type	char		
primary_host	text		
primary_mountpoint	text		
primary_device	text		
mirror_host	text		
mirror_mountpoint	text		
mirror_device	text		

gp_segment_configuration

此表包含主实例与镜像实例的配置信息。

字段	类型	参考	描述
dbid	smallint		实例的唯一标识符
content	smallint		数据在节点上的标识符。主节点与镜像节点具有相同的 content 值。对于一个节点来说，值为 0~N，N 为 GP 数据库的节点数。而对于 Master 来说该值为-1
role	char		实例当前运行的角色，可以为 p(primary)或 m(mirror)
preferred_role	char		系统初始化时分配给实例的角色，可以为 p(primary)或 m(mirror)
mode	char		主节点与镜像节点之间的同步状态，可以为 s(同步)、c(变化跟踪)、r(重新同步中)
status	char		实例的故障状态，可以为 u(启动)或 d(停止)
port	integer		数据库服务监听程序使用的 TCP 端口
hostname	text		实例主机的主机名
address	text		用以访问主机上特定实例的主机名。可以是系统的主机名，也可以每个实例指定对应一个独立网络接口的主机名
replication_port	integer		复制程序保持主实例与镜像实例之间同步的 TCP 端口
san_mounts	int2vector	gp_san_configuration	gp_san_configuration 表的引用。仅用于使用共享存储初

		.oid	始化系统的情况

gp_pgdatabase

此视图显示GP节点实例的状态信息，以及是否以主节点或镜像节点运行。此视图被GP内部用作故障诊断，以及恢复命令用以确定故障节点。

字段	类型	参考	描述
dbid	smallint	gp_segment_configuration.dbid	实例的唯一标识符
isprimary	boolean	gp_segment_configuration.role	实例是否激活。是否以主实例角色运行(相反的为镜像角色)
content	smallint	gp_segment_configuration.content	数据在节点上的标识符。主节点与镜像节点具有相同的 content 值。对于一个节点来说，值为 0~N，N 为 GP 数据库的节点数。而对于 Master 来说该值为-1
definedprimary	boolean	gp_segment_configuration.preferred_role	实例是否在系统初始化时被定义为主实例(相反的为镜像实例)

gp_transaction_log

此视图包含特定实例的本地事务状态信息。该视图用于查看本地事务状态。

字段	类型	参考	描述
segment_id	smallint	gp_segment_configuration.content	数据在节点上的标识符。主节点与镜像节点具有相同的 content 值。对于一个节点来说，值为 0~N，N 为 GP 数据库的节点数。而对于 Master 来说该值为-1
dbid	smallint	gp_segment_configuration.dbid	实例的唯一标识符
transaction	xid		本地事务标识符
status	text		本地事务的状态(Committed 或 Aborted)

gp_version_at_initdb

此表位于GP数据库系统的Master和每个Segment实例。在系统首次初始化时，其表明GP数据库的版本。其定义在pg_global表空间，在系统中的所有数据库之间都是共享可访问的。

字段	类型	参考	描述
schemaversion	integer		模式版本号
productversion	text		产品版本号

pg_aggregate

此表存储着聚合函数的信息。聚合函数将一组值处理为一个值。典型的聚合函数有sum、count、max等。该表中的每条记录都是pg_proc表中的扩展。pg_proc表中包含着聚合函数的名称，输入输出参数的类型以及其他类似普通函数的信息。

字段	类型	参考	描述
aggfnoid	regproc	pg_proc.oid	聚合函数对象标识符
aggtransfn	regproc	pg_proc.oid	转换函数对象标识符
aggprelimfn	regproc		预处理函数对象标识符
aggfinalfn	regproc	pg_proc.oid	结束函数对象标识符

agginitval	text		转换状态的初始值，为一个包含外部字符串初始值的文本字段。如果该字段为 NULL，该过度状态以 NULL 开始
agginvtransfn	regproc	pg_proc.oid	aggtransfn 的逆函数对象标识符
agginvprelimfn	regproc	pg_proc.oid	aggprelimfn 的逆函数对象标识符
aggordered	boolean		如果为真，该聚合被定义为 ORDERED
aggsortop	oid	pg_operator.oid	相关的操作对象标识符
aggtranstype	oid	pg_type.oid	聚合函数内部过度的数据类型

pg_am

此表存储着索引访问方式的信息。系统支持的每种索引访问方式为一行。

字段	类型	参考	描述
amname	name		访问方式的名称
amstrategies	int2		访问方式操作策略的序号
amsupport	int2		访问方式支持过程的序号
amorderstrategy	int2		如果索引不提供排序则为 0，否则为排序操作策略的序号
amcanunique	boolean		访问方式是否支持唯一索引
amcanmulticol	boolean		访问方式是否多列索引
amoptionalkey	boolean		访问方式是否支持没有任何限制的访问第一个索引列
amindexnulls	boolean		访问方式是否支持 NULL 索引条目
amstorage	boolean		索引存储的类型是否可以与数据列不同
amclusterable	boolean		索引是否可以被聚集
aminsert	regproc	pg_proc.oid	插入元组的函数
ambeginscan	regproc	pg_proc.oid	开始新扫描的函数
amgettuple	regproc	pg_proc.oid	下一有效元组的函数
amgetmulti	regproc	pg_proc.oid	捕获多元组的函数
amrescan	regproc	pg_proc.oid	重新扫描的函数
amendscan	regproc	pg_proc.oid	结束扫描的函数
ammarkpos	regproc	pg_proc.oid	标记当前扫描位置的函数
amrestrpos	regproc	pg_proc.oid	恢复已标记扫描位置的函数
ambuild	regproc	pg_proc.oid	建立新索引的函数
ambulkdelete	regproc	pg_proc.oid	批量删除的函数
amvacuumcleanup	regproc	pg_proc.oid	回收清理的函数
amcostestimate	regproc	pg_proc.oid	评估索引扫描开销的函数
amoptions	regproc	pg_proc.oid	为索引解析和验证 reloptions 的操作

pg_amop

此表存储着和索引访问方法操作符类关联的信息。如果一个操作符是一个操作符类中的成员，那么在这个表中会占据一行。

字段	类型	参考	描述
amopclaid	oid	pg_opclass.oid	该条目对应的索引操作符类
amopsubtype	oid	pg_type.oid	区分一个策略的多个条目的子类型
amopstrategy	int2		操作符策略号
amopreqcheck	boolean		索引是否必须重新检查(recheck)
amopopr	oid	pg_operator.oid	操作符标识符

pg_amproc

此表存储着与索引访问方法操作符类相关联的支持过程的信息。每个属于某个操作符类的支持过程都占有一行。

字段	类型	参考	描述
amopclaid	oid	pg_opclass.oid	使用这条记录的索引操作符类
amprocsubtype	oid	pg_type.oid	如果是跨类型的过程，就是子类型，否则就是零
amprocnum	int2		支持过程编号
amproc	regproc	pg_proc.oid	过程的 OID

pg_appendonly

此表存储着AO表的存储选项和一些其他特性信息。该表仅位于Master主机。

字段	类型	参考	描述
relid	oid		AO 表的对象标识符
blocksize	integer		AO 表的块大小，有效值为 8K-2M。缺省为 32K
safefswritesize	integer		能够安全写入不熟悉文件系统的最小尺寸。通常设置为文件系统扩展尺寸(Linux ext3 为 4096，因此常用 32768)的倍数。
majorversion	smallint		AO 表的主要版本号
minorversion	smallint		AO 表的次要版本号
checksum	boolean		一个检查求和值，用以确保数据块在压缩和扫描时的完整性。除非 gp_appendonly_verify_block_checksums 参数(缺省状态下为了性能而禁用)开启该数据才会被存储
compresstype	text		压缩 AO 表的压缩方式。有效的值为 zlib(gzip 压缩)和 quicklz
compresslevel	smallint		压缩级别，随着压缩率从 1 到 9.指定 quicklz 压缩方式时有效值为，指定 zlib 时，有效值为 1 - 9
columnstore	boolean		t 对应按列存储，f 对应按行存储
segrelid	oid		表在磁盘上的文件标识符
segidxid	oid		索引在磁盘上的文件标识符
blkdirrelid	oid		磁盘上用作列存表使用文件的块
blkdiridxid	oid		磁盘上用作列存索引文件的块

pg_attrdef

此表存储字段缺省值。字段的主要信息存放在 pg_attribute。只有明确声明一个缺省值(在创建表或增加字段时指定)的字段才会出现在这里。

字段	类型	参考	描述
adrelid	oid	pg_class.oid	列所属的表标识符
adnum	int2	pg_attibute.attnum	列的顺序
adbin	text		内部用以表示缺省值
adsrc	text		易读的缺省值。为遗留字段，最好不要使用

pg_attribute

此表存储关于表的字段信息。数据库里每个表的每个字段都在该表里有一行。(对于索引和pg_class表中的对象该表中也有对应记录)Attribute等效为Column。这种用法是历史原因。

字段	类型	参考	描述
attrelid	oid	pg_class.oid	字段所属的表标识符
attname	name		字段名字
atttypid	oid	pg_type.oid	字段的数据类型
attstattarget	int4		控制 ANALYZE 为这个字段积累的统计细节的级别。零值表示不收集统计信息。负数表示使用系统缺省的统计对象。正数值意味着和数据类型相关。对于标量数据类型，attstattarget 既是要收集的"最常用数值"的目

			标数量，也是要创建的柱状图的目标数量。
attlen	int2		字段类型的 pg_type.typlen 的拷贝
attnum	int2		字段序号。普通字段是从 1 开始计数的。
attndims	int4		表示数据的维度。否则是 0 。目前，一个数组的维数并未强制，因此任何非零值都表示"这是一个数组"
attcacheoff	int4		在磁盘上的时候总是 -1 ，但是如果加载入内存中的行描述器中，它可能会被更新为缓冲在行中字段的偏移量
atttypmod	int4		记录创建新表时支持特定类型的数据(比如一个 varchar 字段的最大长度)。它传递给类型相关的输入和长度转换函数当做第三个参数。其值对那些不需要 atttypmod 的类型通常为 -1
attbyval	bool		字段类型的 pg_type.typbyval 的拷贝
attstorage	char		字段的类型的 pg_type.typstorage 的拷贝。对于可压缩的数据类型(TOAST)，这个字段可以在字段创建之后改变，以便于控制存储策略
attalign	char		字段类型的 pg_type.typalign 的拷贝
attnotnull	bool		代表一个非空约束。可以改变这个字段以打开或者关闭这个约束
atthasdef	bool		若字段有一个缺省值，此时它对应 pg_attrdef 表里实际定义此值的记录
attisdropped	bool		字段已经被删除了，不再有效。一个已经删除的字段物理上仍然存在表中，但会被分析器忽略，因此不能再通过 SQL 访问
attislocal	bool		字段是局部定义在关系中的。请注意一个字段可以同时是局部定义和继承的
attinhcount	int4		字段所拥有的直接祖先的个数。如果一个字段的祖先个数非零，那么它就不能被删除或重命名

pg_attribute_encoding

此表存储着列存储信息。

字段	类型	矫正值	存储	描述
attrelid	oid	not null	普通	外键 pa_attribute.attrelid
attnum	smallint	not null	普通	外键 pg_attribute.attnum
attoptions	text[]		扩展	参数

pg_auth_members

此表展示角色之间的成员关系。任何非循环的关系集合都是允许的。因为角色是系统范围的，pg_auth_members在GPDB系统中是全局共享的。

字段	类型	参考	描述
roleid	oid	pg_authid.oid	拥有成员的角色标识符
member	oid	pg_authid.oid	属于 roleid 角色一个成员的角色标识符
grantor	oid	pg_authid.oid	赋予此成员关系的角色标识符
admin_option	bool		如果 member 可以把 roleid 角色的成员关系赋予其它角色，则为真。

pg_authid

此表包含有关数据库认证标识符(角色)的信息。一个角色包含了用户和组的概念。一个用户实际上是一个设置了rolcanlogin标志的角色。任何角色(无论是否设置了rolcanlogin标识符)都可以有其他角色作为成员。参照pg_auth_members。

由于这个表包含密码信息，它必不是公共可读的。pg_roles是建立在pg_authid上的视图，其密码字段被置为空白。

由于用户是系统范围的，pg_authid在GPDB系统中是全局共享的：每个库中只是一个pg_authid的拷贝，而并非每个库中有一个独立的表。

字段	类型	描述
rolname	name	角色名称
rolsuper	boolean	角色是否拥有超级用户权限
rolinherit	boolean	角色是否自动继承其所属角色的权限
rolcreaterole	boolean	角色是否可以创建角色
rolcreatedb	boolean	角色是否可以创建数据库
rolcatupdate	boolean	角色是否可以直接更新系统表(如果没有设置为真，即便超级用户也不能这样做)
rolcanlogin	boolean	角色是否可以登录。就是说，该用户是否可以通过会话的方式连接
rolconnlimit	int4	对可登录的角色，限制其最大并发连接数。-1 表示无限制
rolpassword	text	角色密码(可能是加密的)。如果没有密码则为 NULL
rolvaliduntil	timestamptz	角色的密码失效时间，没有失效期则为 NULL
rolconfig	text[]	运行会话时缺省配置参数，一般通过 ALTER ROLE rolename SET 方式设置
rolresqueue	oid	角色锁分配到的资源队列的对象标识符
rolcreaterextgpfd	boolean	是否可以创建基于 gpfdist 协议的可读外部表
rolcreaterexthttp	boolean	是否可以创建基于 http 协议的可读外部表
rolcreatewextgpfd	boolean	是否可以创建基于 gpfdist 协议的可写外部表
rolcreaterexthdfs	boolean	是否可以创建基于 hdfs 协议的可读外部表
rolcreatewexthdfs	boolean	是否可以创建基于 hdfs 协议的可写外部表

pg_autovacuum

此表存储着autovacuum守护进程针对每个表相关的配置参数，如果一个表有在这里有记录，这些特定的参数将用于自动清理该表。对于没有记录的表，将使用系统缺省的配置参数。

当某个表的更新或删除记录数超过vac_base_thresh加上vac_scale_factor乘以表中当前有效的评估数，autovacuum守护进程将执行一次VACUUM操作。类似的，当摸个表的更新或者删除记录数超过anl_base_thresh加上anl_scale_factor乘以表中当前有效的评估数，将执行一次ANALYZE操作。

不过，如果pg_class.relfrozenxid字段超过freeze_max_age次事务，无论该表是否被修改，autovacuum都会执行VACUUM操作以避免事务ID的重叠，即便在autovacuum被禁用的情况下，系统也会调用autovacuum来执行VACUUM。

任何数字类型的字段都可以包含-1的值，这标识着将使用系统缺省值。需注意，vac_cost_delay参数从autovacuum_vacuum_cost_delay参数继承缺省值，如果前面的参数设置为负值，则从vacuum_cost_delay继承缺省值。对于vac_cost_limit也一样。然而，autovacuum将忽略试图针对单个表设置大于全局值的freeze_max_age值(仅允许比全局值小)，同时，freeze_min_age值将被限制为系统全局参数autovacuum_freeze_max_age值的一半之内。

字段	类型	参考	描述
vacrelid	oid	pg_class.oid	此记录对应的表
enable	boolean		如果为假，该表不会被自动清理
vac_base_thresh	integer		清理前修改的最小记录数
vac_scale_factor	float4		追加到 vac_bash_thresh 上的表记录数的倍数
anl_bash_thresh	integer		分析前修改的最小记录数

anl_scale_factor	float4		追加到 anl_base_thresh 上的表记录数的倍数
vac_cost_delay	integer		自定义的 vacuum_cost_delay 参数
vac_cost_limit	integer		自定义的 vacuum_cost_limit 参数
freeze_min_age	integer		自定义的 vacuum_freeze_min_age 参数
freeze_max_age	integer		自定义的 autovacuum_freeze_max_age 参数

pg_cast

此表存储着数据类型转换的路径，包括内置路径和那些通过CREATE CAST定义的路径。在pg_cast表中列出的类型转换函数必须总是以类型转换的源类型作为它的第一个参数类型，同时以转换的目的类型作为结果类型。一个类型转换函数最多可以有3个参数。如果出现第二个参数，必须是integer类型，它接受与目标类型关联的修饰词，如果没有则为-1。如果出现第三个参数，其必须是boolean类型，如果该类型转换是一种明确的转换，那么它接受true，否则接受false。

如果函数接受多于1个参数，在pg_cast里创建一条源类型和目标类型相同的记录是合理的。这样的记录代表长度转换函数，他们把该类型的数值转换为对特定的类型修饰词数值合法的值。请注意，现在还不支持将非缺省类型修饰词和用户创建数据类型关联起来，因此这个设置只用于少量的内置类型，这些类型都有内置于语法分析器里的类型修饰词语法。

如果一条pg_cast记录有着不同的原类型和目标类型，并且有一个接收多于一个参数的函数，那么它就意味着用一个步骤从一种类型转换到另外一种类型，同时还附加一个长度转换。如果没有这样的记录，那么转换成一个使用了类型修饰词的类型涉及两个步骤，一个是在数据类型之间转换，另外一个是附加修饰词。

字段	类型	参考	描述
castsource	oid	pg_type.oid	源数据类型的 OID
casttarget	oid	pg_type.oid	目标数据类型的 OID
castfunc	oid	pg_proc.oid	用于执行这个转换的函数 OID。如果该数据类型是二进制兼容的，那么值为 0(就是说，不需要运行时的操作来执行转换)
castcontext	char		标识这个转换可以在什么环境里调用。e 表示只能进行明确的转换(使用 CAST 或::语法)。a 表示在赋值给目标字段的时候隐含调用。i 表示表达式中隐含，以及其他情况

pg_class

此表存储着几乎所有有字段或者类似表的东西。包括索引(还需参照pg_index)、序列、视图、复合类型和一些特殊关系类型。在下面，所有这些对象都称为关系(relations)。不是所有字段对所有类型都有意义的。

字段	类型	参考	描述
relname	name		表、索引、视图等的名字
relnamespace	oid	pg_namespace.oid	包含这个关系的名字空间(模式)OID
reltype	oid	pg_type.oid	对应该表的行类型的数据类型(索引为 0，其无 pg_type 记录)
relowner	oid	pg_authid.oid	关系的所有者
relam	oid	pg_am.oid	如果行是索引,那么就是所用的访问模式(B-tree,hash 等)
relfilenode	oid		关系在磁盘上的文件的名字，如果没有则为 0

reltablespace	oid	pg_tablespace.oid	关系所在的表空间。如果为零，意味着使用该数据库的缺省表空间。如果关系在磁盘上没有文件，该字段没有意义
relpages	int4		关系在磁盘上块(大小为 BLCKSZ)的数量。只是规划器用的一个近似值，是由 VACUUM，ANALYZE 和几个 DDL 命令，比如 CREATE INDEX 更新
reltuples	float4		表中行的数目。规划器使用的一个估计值，是由 VACUUM，ANALYZE 和几个 DDL 命令，比如 CREATE INDEX 更新
reltoastrelid	oid	pg_class.oid	关联的 TOAST 表的 OID，如果没有为 0。TOAST 表在一个从属表里离线存储大字段
reltoastidxid	oid	pg_class.oid	TOAST 表的索引的 OID，如果不是 TOAST 表则为 0
relaosegidxid	oid		在 GPDB 的 3.4 版本已经弃用
relaosegrelid	oid		在 GPDB 的 3.4 版本已经弃用
relhasindex	boolean		如果它是一个表而且至少有(或者最近有过)一个索引，则为真。它是由 CREATE INDEX 设置的，但 DROP INDEX 不会立即将它清除。如果 VACUUM 的表没有索引，那么它将清理 relhasindex
relisshared	boolean		如果该表在整个系统全局共享则为真。只有某些系统表(比如 pg_database)是共享的
relkind	char		对象的类型： r:普通表，i:索引，S:序列，v:视图，c:复合类型，t:TOAST 表，o:内部 AO 节点文件，u:未知类型的临时堆表
relstorage	char		表的存储类型: a:AO 表，h:堆表，v:虚拟表，x:外部表
relnatts	int2		关系中用户字段数目(除了系统字段以外)。在 pg_attribute 里有相同数目对应行。参见 pg_attribute.attnum
relchecks	int2		表中约束的数目
reltriggers	int2		表中触发器数目
relukeys	int2		未使用
relfkeys	int2		未使用
relrefs	int2		未使用
relhasoids	boolean		如果为关系中每条记录生成一个 OID 则为真
relhaspkey	boolean		如果表有一个(或者曾经有一个)主键，则为真
relhasrules	boolean		如果表有规则则为真
relhassubclass	boolean		如果有(或者曾经有)任何继承的子表，则为真
relfrozenxid	xid		该表中所有在这个之前的事务 ID 已经被一个固定的(frozen)事务 ID 替换。这用于跟踪该表是否需要为了防止事务 ID 重叠或者允许收缩 pg_clog 而进行清理。如果该关系不是表则为零(InvalidTransactionId)
relacl	aclitem[]		由 GRANT 和 REVOKE 分配的访问权限
reloptions	text[]		访问方法特定的选项，使用"keyword=value"格式的字符串

pg_compression

此表存储的是可用的压缩方法。

字段	类型	矫正值	存储	描述
compname	name	not null	普通	压缩的名称
compconstructor	regproc	not null	普通	压缩的构造函数
compdestructor	regproc	not null	普通	压缩的析构函数
compcompressor	regproc	not null	普通	压缩包的名称
compdecompressor	regproc	not null	普通	解压缩包的名称
compvalidator	regproc	not null	普通	压缩验证器的名称
compowner	oid	not null	普通	来自 pg_authid 的 oid

pg_constraint

此表存储着表上的检查约束、主键、唯一约束和外键约束。字段约束不会得到特殊对待。每个字段约束都等同于某些表约束。非空约束记录在pg_attribute表中。在域上的检查约束也存储在这里。

字段	类型	参考	描述
conname	name		约束名称(不必唯一)
connamespace	oid	pg_namespace.oid	包含该约束的名字空间的 OID
contype	char		c:检查约束，f:外键约束，p:主键约束，u:唯一约束
condeferrable	boolean		约束是否可以推迟
condeferred	boolean		缺省时这个约束是否推迟
cornelid	oid	pg_class.oid	约束所在的表，如果不是表则为 0
contypid	oid	pg_type.oid	约束所在的域，如果不是域则为 0
confrelid	oid	pg_class.oid	如果是外键，则为参考的表，否则为 0
confupdtype	char		外键更新动作代码
confdeltype	char		外键删除动作代码
confmatchtype	char		外键匹配类型
conkey	int2[]	pg_attribute.attnum	如果是表约束，则为约束控制的字段列表
confkey	int2[]	pg_attribute.attnum	如果是外键约束，则为约束控制的字段列表
conbin	text		如果是检查约束，则为其表达式的内容
consrc	test		如果是检查约束，则为可阅读的表达式的内容

pg_conversion

此表包含可用的编码转换信息。这些信息通过CREATE CONVERSION定义。

字段	类型	参考	描述
conname	name		转换的名称(在一个名字空间里是唯一的)
connamespace	oid	pg_namespace.oid	转换所在的名字空间的 OID
conowner	oid	pg_authid.oid	转换的所有者
conforencoding	int4		源编码 ID
contoencoding	int4		目的编码 ID
conproc	regproc	pg_proc.oid	转换的过程
condefault	boolean		为真表示这是缺省转换

pg_database

此表存储着可用数据库的信息。数据库是通过CREATE DATABASE命令创建的。与多数的系统表不同，该表在GPDB系统中是全局共享的：每个库中只是一个pg_database的拷贝，而并非每个库中有一个独立的表。

字段	类型	参考	描述
datname	name		数据库名称
datdba	oid	pg_authid.oid	数据库的所有者，通常为其创建者
encoding	int4		数据库的字符编码方式。pg_encoding_to_char()能够将这个数字转换为相应的编码名称
datistemplate	boolean		如果为真则此数据库可以用于 CREATE DATABASE 的 TEMPLATE 子句，把新数据库创建为此数据库的克隆。事实上在 GP 中为假的数据库也可以用作 TEMPLATE 子句
datallowconnn	boolean		如果为假则没有人可以连接到这个数据库。这个字段用于保护 template0 数据库不被更改
datconnlimit	int4		设置该数据库上允许的最大并发连接数，-1 表示无限制

datlastsysoid	oid		数据库里最后一个系统 OID，对 pg_dump/gp_dump 有用
datfrozenxid	xid		该数据库中所有在此之前的事务 ID 已经被一个固定的(frozen)事务 ID 替换。这用于跟踪该数据库是否需要为了防止事务 ID 重叠或者允许收缩 pg_clog 而进行清理。它是每个表的 pg_class.relfrozenxid 中的最小值
dattablespace	oid	pg_tablespace.oid	该数据库的缺省表空间。在这个数据库里，所有 pg_class.reltablespace 为零的表都将保存在这个表空间里。所有非共享的系统表也都存放在这里
datconfig	text[]		运行会话时缺省配置参数，一般通过 ALTER DATABASE databasename SET 方式设置
datacl	aclitem[]		由 GRANT 和 REVOKE 分配的访问权限

pg_depend

该表记录着数据库对象之间的依赖关系。这些信息允许DROP命令找出哪些对象必须有DROP CASCADE删除，或者避免在DROP RESTRICT的情况下被删除。类似功能参照pg_shdepend表，其记录着在GPDB系统中全局共享的对象之间的依赖关系。

在所有情况下，一个pg_depend记录表在依赖对象被删除之前不能删除该对象。不过这里还有几种deptype定义的情况：

- DEPENDENCY_NORMAL (n)
  独立创建的对象之间的一般关系。有依赖的对象可以在不影响被引用对象的情况下被删除。被引用对象只有在声明了CASCADE的情况下删除，这时有依赖的对象也被删除。例如：一个表字段对其数据类型有一般依赖关系。
- DEPENDENCY_AUTO (a)
  有依赖对象可以和被引用对象分别删除，并且如果删除了被引用对象则被自动删除(不管是RESTRICT还是CASCADE模式)。例如：一个表上的命名约束是在该表上的自动依赖关系，因此如果删除了表，它也会被删除。
- DEPENDENCY_INTERNAL (i)
  有依赖的对象是最为被引用对象的一部分被创建的，实际上只是其内部实现的一部分。直接DROP有依赖对象是不被允许的(会通知用户需使用一条删除引用对象的DROP)。一个被引用对象的DROP将传播到有依赖的对象，不管是否声明了CASCADE。例如：一个创建用做外键约束的触发器在该约束的pg_constraint记录上标记为内部依赖。
- DEPENDENCY_PIN (p)
  无依赖对象，这种类型的记录标志着系统本身依赖于被引用对象，因此这个对象绝不能被删除。这种类型的记录只有在initdb的时候被创建，有依赖对象的字段里包含0。

字段	类型	参考	描述
classid	oid	pg_class.oid	有依赖对象所在系统表的 OID
objid	oid	任何 OID	指定的有依赖对象的 OID
objsubid	int4		对于表字段，这个是该属性的字段数(objid 和 classid 引用表本身)。对于所有其它对象类型，目前这个字段是零
refclassid	oid	pg_class.oid	被引用对象所在的系统表的 OID
refobjid	oid	任何 OID	指定的被引用对象的 OID
refobjsubid	int4		对于表字段，这个是该字段的字段号(refobjid 和 refclassid 引用表本身)。对于所有其它对象类型，目前这个字段是零
deptype	char		一个定义这个依赖关系特定语义的代码，如上面所述

pg_description

此表可以给每个数据库对象存储一个可选的描述(注释)。可以通过COMMENT命令操作这些描述，并可以通过psql的\d命令查看。许多内置的系统对象的描述信息由该表初始提供。pg_shdescription提供了类似的功能，其记录整个GPDB系统内共享对象的注释信息。

字段	类型	参考	描述
objoid	oid	任意 OID	此描述所对应的对象 OID
classoid	oid	pg_class.oid	对象出现的系统表的 OID
objsubid	int4		对于一个表字段的注释，它是字段号(objoid 和 classoid 指向表自身)。对于其它对象类型，它是零
description	text		对于该对象描述的任意文本

pg_exttable

此表用于跟踪和记录CREATE EXTERNAL TABLE命令创建的外部表和WEB表。

字段	类型	参考	描述
reloid	oid	pg_class.oid	外部表的对象标识符(OID)
location	text[]		外部表文件的 URI 位置
fmttype	char		尾部表文件的格式，t:text 或者 c:csv
fmtopts	text		外部表的格式化选项，比如字段分隔符，空字符串，逃逸字符等
command	text		访问外部表时执行的系统命令
rejectlimit	integer		每个节点的拒绝限制阀值，之后装载将失败
rejectlimittype	char		拒绝限制阀值的类型:r 为记录行数
fmterrtbl	oid	pg_class.oid	出错格式被记录的错误表的 OID
encoding	text		客户端编码
writable	boolean		可读外部表为 false，可写外部表为 true

pg_filespace

此表存储着GPDB系统中创建的文件空间的信息。任何系统都包含一个缺省的文件空间pg_system，这是系统初始化时创建的所有数据目录位置的集合。

一个表空间必须有一个文件系统位置以存储数据库文件。在GPDB中，Master和所有的Segment(Primary和Mirror)都需要自己独立的存储位置。GP系统中所有组件的文件系统位置的集合称为一个文件空间。

字段	类型	参考	描述
fsname	name		文件空间的名字
fsowner	oid	pg_roles.oid	创建文件空间的角色的对象标识符

pg_filespace_entry

一个表空间必须有一个文件系统位置以存储数据库文件。在GPDB中，Master和所有的Segment(Primary和Mirror)都需要自己独立的存储位置。GP系统中所有组件的文件系统位置的集合称为一个文件空间。pg_filespace_entry存储着整个GPDB系统的文件系统位置信息的集合，这些构成了GPDB的文件空间。

字段	类型	参考	描述
fsefsoid	oid	pg_filespace.oid	文件空间的对象标识符
fsedbid	integer	gp_segment_configuration.dbid	节点标识符
fselocation	text		该借点标识符对应的文件系统位置

pg_index

此表存储着关于索引的一部分信息。其它的信息大多数在pg_class表中。

字段	类型	参考	描述
indexrelid	oid	pg_class.oid	该索引在 pg_class 里的 OID
indrelid	oid	pg_class.oid	使用这个索引的表在 pg_class 中的 OID
indnatts	int2		索引中的字段数目(复制的 pg_class.relnatts)
indisunique	boolean		如果为真，此为唯一索引
indisprimary	boolean		如果为真，此为主键。如果该字段为真，indisunique 应总是为真
indisclustered	boolean		如果为真，该表最后在这个索引上建立了聚集
indisvalid	boolean		如果为真，那么该索引可以用于查询。如果为假，那么该索引可能不完整，仍然必须在 INSERT/UPDATE 操作时进行更新，不过不能安全的用于查询
indkey	int2vector	pg_attribute.attnum	这是一个包含 indnatts 值的数组，数组值表示这个索引所建立的表字段。比如一个值为 1 3 的意思是第一个字段和第三个字段组成这个索引键字。这个数组里的零表明对应的索引属性是在这个表字段上的一个表达式，而不是一个简单的字段引用
indclass	oidvector	pg_class.oid	对于索引键里面的每个字段，每个值为指向所使用的操作符类的 OID
indexprs	text		表达式树(以 nodeToString()形式表现)用于那些非简单字段引用的索引属性。它是一个列表，在 indkey 里面的每个零条目一个元素。如果所有索引属性都是简单的引用，则为空
indpred	text		部分索引断言的表达式树(以 nodeToString()的形式表现)。如果不是部分索引，则是空字符串

pg_inherits

此表存储着表继承层次的信息。数据库中的每个直接子表都有一条记录。间接的继承关系可以通过记录链关系来确定。在GPDB中，继承关系可以通过CREATE TABLE命令的INHERITS子句(独立继承)和PARTITION BY子句(分区表继承)创建。

字段	类型	参考	描述
inhrelid	oid	pg_class.oid	子表的 OID
inhparent	oid	pg_class.oid	父表的 OID
inhseqno	int4		如果一个子表存在多个直系父表(多重继承)，这个数字表明此继承的层数。计数从 1 开始

pg_language

此表登记着编程语言，可使用这些语言写函数或者存储过程。其通过CREATE LANGUAGE操作创建。

字段	类型	参考	描述
lanname	name		语言的名字
lanispl	boolean		对于内部语言而言是假(比如 SQL)，对于用户定义的语言则是真。目前，pg_dump 仍然使用这个东西判断哪种语言需要转储，但是这些可能在将来

			被其它机制取代
lanpltrusted	boolean		如果这是可信语言则为真，意味着系统相信它不会被授予任何正常 SQL 执行环境之外的权限。只有超级用户可以创建不可信的语言
lanplcallfoid	oid	pg_proc.oid	对于非内部语言，这是指向该语言处理器的引用，语言处理器是一个特殊函数，负责执行以某种语言写的所有函数
lanvalidator	oid	pg_proc.oid	语言校验器函数，它负责检查新创建函数的语法和有效性。如果没有提供校验器，则为零
lanacl	aclitem[]		由 GRANT 和 REVOKE 分配的访问权限

pg_largeobject

此表存储着那些被标记为大对象的数据。大对象是使用其创建时分配的OID标识的。每个大对象都分解成足够小的小段或者页面以便以行的形势存储在pg_largeobject中。每页的数据量定义为LOBLKSIZE(目前是BLCKSZ/4或者通常是8K)。

pg_largeobject的每一行保存一个大对象的一个页面，从该对象内部的字节偏移(pageno*LOBLKSIZE)开始。这种实现允许松散的存储：页面可以丢失，而且可以比LOBLKSIZE字节少，即使不是对象的最后一页。大对象内丢失的部分读作0.

字段	类型	描述
loid	oid	标识本页大对象的标识符
pageno	int4	本页在大对象中的页码(从 0 开始计算)
data	bytea	存储在大对象中的实际数据。这些数据绝不会超过 LOBLKSIZE 字节，而且可能更少

pg_listener

此表支持LISTEN和NOTIFY命令。一个监听器为它监听的每个通知名称在pg_listener中创建一条记录。一个通知发起人扫描pg_listener并且更新每条匹配的记录以显示一个通知已经发生。通知发起人还发送一个信号给监听器(使用记录在表中的PID)以唤醒它做处理。

该表目前在GPDB中没有被使用。

字段	类型	描述
relname	name	通知条件名(该名字不需要匹配任何数据库中的实际关系)
listenerpid	int4	创建此条目的服务器进程的 PID
notification	int4	如果这个监听器上没有等待的事件，那么是零。如果有等待的事件，那么是发送通知的服务器的 PID

pg_locks

pg_locks 提供有关在数据库服务器中由打开的事务持有的锁的信息。pg_locks对每个活跃的可锁定对象、请求的锁模式、以及相关的事务保存一行。因此，如果多个事务持有或者等待对同一个对象的锁，那么同一个可锁定的对象可能出现多次。不过，一个目前没有锁在其上的对象将肯定不会出现。

有好几种不同的可锁定对象：所有的关系(比如一个表)、关系中独立页面、关系中独立的行、一个事务ID、以及一般的数据库对象。还有，扩展一个关系的权限也是

用一种独立的可锁定对象表示。

字段	类型	参考	描述
locktype	text		可锁定对象的类型：relation，extend，page，tuple，transactionid，object，userlock，resource queue 或者 advisory
database	oid	pg_datanase.oid	对象所在的数据库的 OID，如果对象是共享对象，那么就是零，如果对象是一个事务 ID，就是 NULL
relation	oid	pg_class.oid	关系的 OID，如果对象不是关系，也不是关系的一部分，则为 NULL
page	integer		关系内部的页面编号，如果对象不是行也不是关系页，则为 NULL
tuple	smallint		页面里面的行编号，如果对象不是行，则为 NULL
transactionid	xid		事务的 ID，如果对象不是事务 ID，则为 NULL
classid	oid	pg_class.oid	包含该对象的系统表的 OID，如果对象不是普通数据库对象，则为 NULL
objid	oid	任意 OID	对象在其系统表内的 OID，如果对象不是普通数据库对象，则为 NULL
objsubid	smallint		对于表的一个字段，这是字段编号(classid 和 objid 指向表自身)。对于其它对象类型，这个字段是零。如果这个对象不是普通数据库对象，则为 NULL
transaction	xid		持有此锁或者在等待此锁的事务的 ID
pid	integer		持有或者等待这个锁的服务器进程的进程 ID。如果锁是被一个预备事务持有的，那么为 NULL
mode	text		这个进程持有的或者期望的锁模式
granted	boolean		如果持有锁，为真，如果等待锁，为假
mppsessionid	integer		与此锁相关的客户端会话 ID
mppiswriter	boolean		是否被一个写操作持有该锁
gp_segmeng_id	integer		被持有的锁所在的 GP 节点的标识符(dbid)

pg_namespace

此表存储着名字空间。名字空间是SQL模式下的层次结构：每个名字空间有独立的且互相不冲突的关系、类型等集合。

字段	类型	参考	描述
nspname	name		名字空间的名字
nspowner	oid	pg_authid.oid	名字空间的所有者
nspacl	aclitem[]		由 GRANT 和 REVOKE 分配的访问权限

pg_opclass

此表存储着索引访问方法操作符类的定义。每个操作符类定义了语义索引列的一个特定的数据类型和一个特定的索引访问方法。值得注意的是，对于一种指定的数据类型/访问方法，可能会有多种操作符类，这样就可以支持多种行为。大多数的操作符类的定义信息不在pg_opclass中，而是在相关的pg_amop和pg_amproc中。这些记录被认为是操作符类定义的一部分——这不像那样关系被定义为一个pg_class行，再加上相关的pg_atrribute和其他表的记录。

字段	类型	参考	描述
opcamid	oid	pg_am.oid	操作符类所服务的索引访问方法
opcname	name		操作符类的名字
opcnamespace	oid	pg_namespace.oid	操作符类的名字空间
opcowner	oid	pg_authid.oid	操作符类所有者
opcintype	oid	pg_type.oid	操作符类的输入数据类型
opcdefault	boolean		如果操作符类是 opcintype 的缺省，则为真

opckeytype	oid	pg_type.oid	索引数据的类型，如果和 opcintype 相同则为零

pg_operator

此表存储着操作符相关的信息，包括内置的操作符和通过CREATE OPERATOR定义的操作符。未使用的列包含0值。例如：对于前操作符来说左操作的值为0。

字段	类型	参考	描述
oprname	name		操作符的名字
oprnamesspace	oid	pg_namespace.oid	包括此操作符的名字空间的 OID
oprowner	oid	pg_authid.oid	操作符的所有者
oprkind	char		操作符的种类：b=中缀，l=前缀，r=后缀
oprcanhash	boolean		操作符是否支持 hash join
oprleft	oid	pg_type.oid	左操作数的类型
oprtight	oid	pg_type.oid	右操作数的类型
oprresult	oid	pg_type.oid	结果的类型
oprcom	oid	pg_operator.oid	操作符的交换符，如果有的话
oprnegate	oid	pg_operator.oid	操作符的反转符，如果有的话
oprlsortop	oid	pg_operator.oid	如果此操作符支持 Merge 连接，这是左操作数排序的操作符
oprrsortop	oid	pg_operator.oid	如果此操作符支持 Merge 连接，这是右操作数排序的操作符
oprltcmpop	oid	pg_operator.oid	如果此操作符支持 Merge 连接，这是比较左右操作数类型的小于操作符
oprgtcmpop	oid	pg_operator.oid	如果此操作符支持 Merge 连接，这是比较左右操作数类型的大于操作符
oprcode	regproc	pg_proc.oid	该操作符的实现函数
oprrest	regproc	pg_proc.oid	操作符的约束选择性计算函数
oprjoin	regproc	pg_proc.oid	操作符的连接选择性计算函数

pg_partition

此表存储着分区表以及他们的继承层次关系。每条记录都表示了一个分区表在分区层次中的层次结构，或者一个子分区模版的描述。paristemplate属性决定了是否是分区模版。

字段	类型	参考	描述
parrelid	oid	pg_class.oid	表的对象标识符
parkind	char		分区类型：r 表示范围(range)分区，l 表示列表(list)分区
parlevel	smallint		分区级别：0 表示顶级父表，1 表示第一层分区表，2 表示第二层分区表，以此类推
paristemplate	boolean		是否表示一个子分区模版(true)或者一个实际的分区(false)
parnatts	smallint	pg_attribute.attnum	定义这个分区级别的属性序号
paratts	int2vector		定义这个分区级别的属性序号(pg_attribute.attnum)数组
parclass	oidvector	pg_opclass.oid	分区列的操作符标识符

pg_partition_columns

此视图记录着一个分区表的分区键。

字段	类型	参考	描述
schemaname	name		分区表所在的模式的名称
tablename	name		顶级父表的表名
columnname	name		分区键的名称
partitionlevel	smallint		子分区的层级
position_in_partition_key	integer		对于分区表，可以使用复合(多列)分区键。此为该列在符合键

			中的位置序号

pg_partition_encoding

此表存储着分区模版的可用列压缩选项。

字段	类型	矫正值	存储	描述
parencoid	oid	not null	普通	
parencattnum	smallint	not null	普通	
parencattoptions	text[]		扩展	

pg_partition_rule

此表存储着对分区表的追踪信息，检查约束，以及数据包含规则。每条记录都表示一个叶子分区(包含数据的最底层分区)，或者分支分区(顶级或者中间级别的分区，只是用以定义分区层次，但并不包含任何数据)。

字段	类型	参考	描述
paroid	oid	pg_partition.oid	分区所属的分区级别(来自 pg_partition)标识符。对于一个分支分区，是一个空的容器表。对于叶子分区，是一个包含数据记录符合容器规则的表
parchildrelid	oid	pg_class.oid	分区(子表)表的标识符
parparentrule	oid	pg_partition_rule.paroid	分区的父表相关规则的行标识符
parname	name		分区的名字
parisdefault	boolean		是否为缺省分区
parruleord	smallint		对于范围分区表，在这个层次划分的等级
parrangestartincl	boolean		对于范围分区表，是否包含开始值
parrangeendincl	boolean		对于范围分区表，是否包含结束值
parrangestart	text		对于范围分区表，范围的开始值
parrangeend	text		对于范围分区表，范围的结束值
parrangeevery	text		对于范围分区表，EVERY 子句指定的间隔值
parlistvalues	text		对于列表分区表，此分区表分配的值列表
parreloptions	text[]		特定分区的存储特征描述数组

pg_partition_templates

此视图用以显示通过子分区模版创建的子分区。

字段	类型	参考	描述
schemaname	name		分区表所在的模式的名称
tablename	name		顶级父表的名字
partitionname	name		子分区的名字(是 ALTER TABLE 命令修改分区时的名字)。如果在创建时没有指定名字或者由 EVERY 子句生成的，该值为 NULL
partitiontype	text		子分区的类型(范围或者列表)
partitionlevel	smallint		分区表在层次中的层级
partitionrank	bigint		对于范围分区，相对于同等级其他分区的排名(rank)
partitionposition	smallint		该子分区的规则顺序的位置
partitionlistvalues	text		对于列表分区表，此分区表分配的值列表
partitionrangestart	text		对于范围分区表，范围的开始值
partitionstartinclusive	boolean		对于范围分区表，是否包含开始值
partitionrangeend	text		对于范围分区表，范围的结束值
partitionendinclusive	boolean		对于范围分区表，是否包含结束值
partitioneveryclause	text		该子分区的 EVERY 子句(间隔)
partitionisdefault	boolean		t:缺省子分区，f:其他

partitionboundary	text		该子分区的整个分区描述

pg_partitions

此视图用于展示分区表的结构。

字段	类型	参考	描述
schemaname	name		分区表所在的模式的名称
tablename	name		顶级父表的名字
partitiontablename	name		分区表的名字(用于直接访问分区表)
partitionname	name		子分区的名字(是 ALTER TABLE 命令修改分区时的名字)。如果在创建时没有指定名字或者由 EVERY 子句生成的，该值为 NULL
parentpartitiontablename	name		该分区所属的上一级分区表的关系名字
parentpartitionname	name		该分区所属的上一级分区表的指定的名字
partitiontype	text		子分区的类型(范围或者列表)
partitionlevel	smallint		分区表在层次中的层级
partitionrank	bigint		对于范围分区，相对于同等级其他分区的排名(rank)
partitionposition	smallint		该子分区的规则顺序的位置
partitionlistvalues	text		对于列表分区表，此分区表分配的值列表
partitionrangestart	text		对于范围分区表，范围的开始值
partitionstartinclusive	boolean		对于范围分区表，是否包含开始值
partitionrangeend	text		对于范围分区表，范围的结束值
partitionendinclusive	boolean		对于范围分区表，是否包含结束值
partitioneveryclause	text		该子分区的 EVERY 子句(间隔)
partitionisdefault	boolean		t:缺省子分区，f:其他
partitionboundary	text		该子分区的整个分区描述

pg_pltemplate

此表存储着过程语言的模版信息。作为一个语言模版允许该语言在特定的数据库中通过简单的CREATE LANGUAGE命令创建，而不需要指定实现细节。与多数的系统表不同，该表在GPDB系统中是全局共享的：每个库中只是一个pg_pltemplate的拷贝，而并非每个库中有一个独立的表。这样就允许这些信息在需要时每个数据库都可以访问。

目前没有命令可用于操作过程语言模版，要修改这些内建的信息，超级用户必须使用普通的INSERT、DELETE、UPDATE命令来修改。

字段	类型	参考	描述
tmplname	name		模版使用的语言的名字
tmpltrusted	boolean		如果语言被认为是可信的，则为真
tmplhandler	text		调用的处理函数的名字
tmplvalidator	text		校验函数的名字，如果没有则为 NULL
tmpllibrary	text		语言实现的共享库的目录
tmplacl	aclitem[]		模版访问权限(尚未实现)

pg_proc

此表存储着关于函数(或过程)的信息，包括内置函数和那些通过CREATE FUNCTION定义的函数。该表包含聚集函数和普通函数的数据。如果proisagg为真，那么pg_aggregate表中应该有一条记录相匹配。

对于编译函数，不管是内置的和动态加载的函数，prosrc包含函数的C语言名字(连接符)。对于所有其他目前已知的语言类型，prosrc包含函数的原文本。probin除了用于动态加载的C函数之外没有其他用途，此时其给出了包含次函数的共享库文件名。

字段	类型	参考	描述
proname	name		函数的名字
pronamespace	oid	pg_namespace.oid	函数所在名字空间的 OID
proowner	oid	pg_authid.oid	函数的所有者
prolang	oid	pg_language.oid	函数的实验语言或调用接口
proisagg	boolean		是否为聚集函数
prosecdef	boolean		是否为安全定义器(一个 setuid 函数)
proisstrict	boolean		如果任何调用参数都为空，函数直接返回空，此时函数实际上不会调用。不是 strict 的函数必须准备处理空输入
proretset	boolean		是否返回一个集合(指定数据类型的多个数值)
provolatile	char		函数结果是否只依赖于输入参数，或者会被外界因素影响。i:immutable(同样的输入总是返回同样的结果)。s:stable(在同一次扫描中，相同的输入结果不变)。v:volatile(结果可能任何时候发生变化，或者有副作用的函数)
pronargs	int2		参数个数
prorettype	oid		返回值的数据类型
proiswin	boolean		真:既不是一个聚合函数也不是一个标量函数，而是一个纯粹的窗口函数
proargtypes	oidvector	pg_type.oid	存放函数参数数据类型的数组。数组里只包括输入参数(包括 INOUT 参数)，代表该函数的调用签名(接口)
proallargtypes	oid[]	pg_type.oid	存放函数参数数据类型的数组。包括所有参数类型(包括 OUT 和 INOUT)。如果所有参数都是 IN 参数，该字段就为空。数组下标以 1 为起点，因历史原因，proargtypes 下标起点为 0
proargmodes	char[]		存放函数参数模式的数组。i:IN，o:OUT，b: INOUT。如果所有参数都是 IN，该字段为空。下标对应的是 proallargtypes 的位置，而不是 proargtypes
proargnames	text[]		存放函数参数名字的数组。没有名字的参数在数组里为空字符串。如果没有一个参数有名字，该字段为空。此数组的下标对应 proallargtypes 而不是 proargtypes
prosrc	text		该字段告诉函数处理器如何调用该函数。实际上对于解释语言来说就是函数源程序，或一个链接符号，一个文件名，或者任何其它东西，具体取决于语言/调用习惯的实现
probin	bytes		如何调用该函数的附加信息。同样，其含义也和语言相关
proacl	aclitem[]		由 GRANT 和 REVOKE 分配的访问权限

pg_resqueue

此表存储着GPDB中资源队列的信息，用于工作负载管理功能。该表仅位于Master主机。该表被定义在pg_global表空间中，这意味着，该表在GPDB系统中是全局共享的。

字段	类型	参考	描述
rsqname	name		资源队列的名字
rsqcountlimit	real		资源队列活动语句限制的阀值。-1 意味着无限制
rsqcostlimit	real		资源队列成本(cost)限制的阀值。-1 意味着无限制
rsqovercommit	boolean		在系统空闲时，是否允许超过成本限制阀值的语句执行
rsqignorecostlimit			低于该成本限制的语句被当作小查询。低于该成本限制的语句将不需要排队，且会立即得到执行

pg_resourcetype

此表存储着可以分配给GPDB资源队列的扩展属性信息。每条记录描述一个属性和内在特性，如缺省设置、是否必须、是否禁用(如果允许的话)等。

该表仅位于Master主机。该表被定义在pg_global表空间中，这意味着，该表在GPDB系统中是全局共享的。

字段	类型	参考	描述
resname	name		资源类型的名字
restypid	smallint		资源类型的 ID
resrequired	boolean		对于一个有效的资源队列来说，该资源类型是否必须
reshasdefault	boolean		该资源类型是否有缺省值，如果为真，缺省值由 resdefaultsetting 指定
rescandisable	boolean		该资源类型是否可以被删除或者禁用。如果为真，缺省值由 resdisabledsetting 指定
resdefault	text		在合适的时候作为该资源类型的缺省设置
resdisabledsetting	text		禁用(如果允许的话)该资源类型时的缺省值

pg_resqueue_attributes

此视图允许管理员查看一个资源队列的属性设置，比如活动会话限制、查询成本限制、优先级等。

字段	类型	参考	描述
rsqname	name	pg_resqueue.rsqname	资源队列的名字
resname	text	pg_resourcetype.resname	资源队列属性的名字
resetting	text		资源队列属性的当前值
restypid	integer	pg_resourcetype.restypid	资源类型的 ID

pg_resqueue_status

此视图允许管理员查看工作负载管理资源队列的状态。其显示特定资源队列中多少在等待的查询、多少正在执行的查询等。

字段	类型	参考	描述
rsqname	name	pg_resqueue.rsqname	资源队列的名字
rsqcountlimit	real	pg_resqueue.rsqcountlimit	资源队列活动语句限制的阀值。-1 意味着无限制
rsqcountvalue	real		该资源队列目前正被使用的活动查询槽位的数量
rsqcostlimit	real	pg_resqueue.rsqcostlimit	资源队列成本(cost)限制的阀值。-1 意味着无限制
rsqcostvalue	real		该资源队列目前正被使用的成本(COST)总和
rsqwaiters	real		该资源队列目前正处于等待状态的会话数量
rsqholders	integer		来自该资源队列正在系统中运行的会话数量

pg_resqueuecapability

此表存储着GPDB系统中已有资源队列的扩展属性信息。只有被分配了扩展属性(如优先级设置)的资源队列才会记录到该表中。该表通过资源队列标识(resqueueid)符与pg_resqueue表关联，同时通过资源类型标识符与pg_resourcetype(restypid)表关联。

该表仅位于Master主机。该表被定义在pg_global表空间中，这意味着，该表在GPDB系统中是全局共享的。

字段	类型	参考	描述
resqueueid	oid	pg_resqueue.oid	关联的资源队列的对象标识符
restypid	smallint	pg_resourcetype.restypid	资源类型。源于 pg_resourcetype 表
resetting	opaque type		属性设定的值。根据资源类型的不同，该值可能会有不同的数据类型

pg_rewrite

此表存储着表和视图的规则信息。如果一个表有任何的规则在该表中，相应的pg_class.relhasruls必须为真。

字段	类型	参考	描述
rulename	name		规则的名字
ev_class	oid	pg_class.oid	规则所属的表
ev_attr	int2		规则适用的字段(目前总为 0，意指整个表)
ev_type	char		规则适用的时间类型。1:SELECT，2:UPDATE，3:INSERT，4:DELETE
is_instead	boolean		如果该规则是 INSTEAD 规则，该字段则为真
ev_qual	text		规则资格条件表达式树(以 nodeToString()形势存在)
ev_action	text		规则动作的查询树(以 nodeToString()形势存在)

pg_roles

该视图提供了访问数据库的角色有关信息的接口。此视图是pg_authid表的简单公开可读的视图，而把口令字段用空白填充了。该视图明确显示了底层表的OID字段，可用于同其他表连接。

字段	类型	参考	描述
rolname	name		角色名称
rolsuper	boolean		角色是否拥有超级用户权限
rolinherit	boolean		角色是否自动继承其所属角色的权限
rolcreaterole	boolean		角色是否可以创建角色
rolcreatedb	boolean		角色是否可以创建数据库
rolcatupdate	boolean		角色是否可以直接更新系统表(如果没有设置为真，即便超级用户也不能这样做)
rolcanlogin	boolean		角色是否可以登录。就是说，该用户是否可以通过会话的方式连接
rolconnlimit	int4		对可登录的角色，限制其最大并发连接数。-1 表示无限制
rolpassword	text		角色密码(可能是加密的)。如果没有密码则为 NULL
rolvaliduntil	timestamptz		角色的密码失效时间，没有失效期则为 NULL
rolconfig	text[]		运行会话时缺省配置参数，一般通过 ALTER ROLE rolename SET 方式设置
rolresqueue	oid	pg_resqueue.oid	角色锁分配到的资源队列的对象标识符
oid	oid	pg_authid.oid	角色的对象标识符
rolcreaterextgpfd	boolean		是否可以创建基于 gpfdist 协议的可读外部表
rolcreaterexthttp	boolean		是否可以创建基于 http 协议的可读外部表
rolcreatewextgpfd	boolean		是否可以创建基于 gpfdist 协议的可写外部表
rolcreaterexthdfs	boolean		是否可以创建基于 hdfs 协议的可读外部表
rolcreatewexthdfs	boolean		是否可以创建基于 hdfs 协议的可写外部表

pg_shdepend

此表存储着数据库对象和共享对象(比如角色)之间的依赖关系。这些信息允许GPDB确保在试图删除对象之前，其没有被引用。pg_depend表的作用是类似的，只不过其只是在一个数据库内部用于存储对象之间的依赖关系。与多数的系统表不同，该表在GPDB系统中是全局共享的：每个库中只是一个pg_shdepend的拷贝，而并非每个库中有一个独立的表。

在所有情况下，一个pg_shdepend记录表在依赖对象被删除之前不能删除该对象。不过这里还有几种deptype定义的情况：

- DEPENDENCY_NORMAL (n)
  独立创建的对象之间的一般关系。有依赖的对象可以在不影响被引用对象的情况下被删除。被引用对象只有在声明了CASCADE的情况下删除，这时有依赖的对象也被删除。例如：一个表字段对其数据类型有一般依赖关系。
- DEPENDENCY_AUTO (a)
  有依赖对象可以和被引用对象分别删除，并且如果删除了被引用对象则被自动删除(不管是RESTRICT还是CASCADE模式)。例如：一个表上的命名约束是在该表上的自动依赖关系，因此如果删除了表，它也会被删除。
- DEPENDENCY_INTERNAL (i)
  有依赖的对象是最为被引用对象的一部分被创建的，实际上只是其内部实现的一部分。直接DROP有依赖对象是不被允许的(会通知用户需使用一条删除引用对象的DROP)。一个被引用对象的DROP将传播到有依赖的对象，不管是否声明了CASCADE。例如：一个创建用做外键约束的触发器在该约束的pg_constraint记录上标记为内部依赖。
- DEPENDENCY_PIN (p)
  无依赖对象，这种类型的记录标志着系统本身依赖于被引用对象，因此这个对象绝不能被删除。这种类型的记录只有在initdb的时候被创建，有依赖对象的字段里包含0。

字段	类型	参考	描述
dbid	oid	pg_database.oid	有依赖对象所在数据库的 OID，共享对象为 0
classid	oid	pg_class.oid	有依赖对象所在系统表的 OID
objid	oid	任何 OID	指定的有依赖对象的 OID
objsubid	int4		对于表字段，这个是该属性的字段数(objid 和 classid 引用表本身)。对于所有其它对象类型，目前这个字段是零
refclassid	oid	pg_class.oid	被引用对象所在的系统表的 OID
refobjid	oid	任何 OID	指定的被引用对象的 OID
refobjsubid	int4		对于表字段，这个是该字段的字段号(refobjid 和 refclassid 引用表本身)。对于所有其它对象类型，目前这个字段是零
deptype	char		一个定义这个依赖关系特定语义的代码，如上面所述

pg_shdescription

此表存储着共享数据库对象可选的描述(注释)。可以通过COMMENT命令操作这些注释的内容，并可以通过psql的\d命令查看。pg_description提供了类似的功能，其存储着单个数据库中对象的注释。与多数的系统表不同，该表在GPDB系统中是全局共

享的：每个库中只是一个pg_shdescription的拷贝，而并非每个库中有一个独立的表。

字段	类型	参考	描述
objoid	oid	任意 OID	此描述所对应的对象 OID
classoid	oid	pg_class.oid	对象出现的系统表的 OID
description	text		对于该对象描述的任意文本

pg_stat_activity

此视图为每个服务器进程显示一行，以及用户会话相关的细节和查询。报告当前查询相关信息的各个字段只有在打开stats_command_string参数的时候才可用。另外，除非查询这些字段的用户是超级用户或者会话的所有者，否则它们显示为空。

字段	类型	参考	描述
datid	oid	pg_database.oid	数据库对象标识符
datname	name		数据库名称
procpid	integer		服务器进程的进程 ID
sess_id	integer		会话 ID
usesysid	oid	pg_authid.oid	角色 OID
usename	name		角色名字
current_query	text		正在执行的查询语句
waiting	boolean		如果在等待锁则为真，不在等待则为假
query_start	timestampz		后端处理开始的时间
client_addr	inet		客户端地址
client_port	integer		客户端端口
application_name	text		客户端应用程序名字
xact_start	timestampz		事物开始的时间

pg_stat_operations

此视图显示数据库对象(比如表、索引、视图或数据库)或全局共享对象(比如角色)上最后执行的操作细节信息。

字段	类型	参考	描述
classname	text		对象存储在系统模式 pg_catalog 中的名字。pg_class:relations，pg_database:databases，pa_namespace:schemas，pg_authid:roles
objname	name		对象的名字
objid	oid		对象的 OID
schemaname	name		对象所属的模式的名字
usestatus	text		在该对象上执行了最后操作的角色的状态。CURRENT:仍在系统中活动的角色，DROPPED:在系统中已不存在，CHANGED:执行该操作之后发生了变化
usename	name		执行该操作的角色名字
actionname	name		此对象上的操作的名字
subtype	text		此对象上的操作的类型或子操作
statime	timestampz		操作的时间戳。这与记录到 GPDB 服务器日志文件的时间戳相同，以便在日志中查看操作的更多细节信息

pg_stat_partition_operations

此视图显示分区表上最后执行的操作的详情。

字段	类型	参考	描述

classname	text		对象存储在系统模式 pg_catalog 中的名字。 pg_class:relations，pg_database:databases， pa_namespace:schemas，pg_authid:roles
objname	name		对象的名字
objid	oid		对象的 OID
schemaname	name		对象所属的模式的名字
usestatus	text		在该对象上执行了最后操作的角色的状态。 CURRENT:仍在系统中活动的角色，DROPPED:在系统中已不存在，CHANGED:执行该操作之后发生了变化
usename	name		执行该操作的角色名字
actionname	name		此对象上的操作的名字
subtype	text		此对象上的操作的类型或子操作
statime	timestampz		操作的时间戳。这与记录到 GPDB 服务器日志文件的时间戳相同，以便在日志中查看操作的更多细节信息
partitionlevel	smallint		分区表在层次中的层级
parenttablename	name		当前分区的上一级父表的关系名字(用于直接访问)
parentschemaname	name		父级所在模式的名字
parent_relid	oid		当前分区的上一级父表的 OID

pg_stat_resqueues

此视图允许管理员查看过去的资源队列的工作负载的指标。为了是的该视图可以收集相关的统计信息，必须启用位于GPDB的Master实例上的服务器配置参数stats_queue_level。启用该统计指标的收集会导致稍微的性能损失，因为每个通过资源队列提交的语句都必须记录到系统日志表中。

字段	类型	参考	描述
queueoid	oid	pg_resqueue.oid	资源队列的 OID
queuename	name	pg_resqueue. rsqname	资源队列的名字
n_queries_exec	bigint		从该资源队列提交的查询数量
n_queries_wait	bigint		从该资源队列提交，且在执行之前必须等待的查询数量
elapsed_exec	bigint		通过该资源队列执行的查询运行的总时间
elapsed_wait	biging		通过该资源队列执行的查询在运行之前必须等待的总时间

pg_stat_last_operation

此表存储着数据库对象(表、视图等)的跟踪元数据。

字段	类型	参考	描述
classid	oid		包含该对象的系统日志表的 OID
objid	oid		对象的 OID
staactionname	name		此对象上的操作的名字
stasysid	oid	pg_authid.oid	执行该操作的角色 OID
stausename	name		执行该操作的角色名字
stasubtype	text		此对象上的操作的类型或子操作
statime	timestampz		操作的时间戳。这与记录到 GPDB 服务器日志文件的时间戳相同，以便在日志中查看操作的更多细节信息

pg_stat_last_shoperation

此表存储着数据库全局共享对象(角色、表空间等)的跟踪元数据。

字段	类型	参考	描述

classid	oid		包含该对象的系统日志表的 OID
objid	oid		对象的 OID
staactionname	name		此对象上的操作的名字
stasysid	oid	pg_authid.oid	执行该操作的角色 OID
stausename	name		执行该操作的角色名字
stasubtype	text		此对象上的操作的类型或子操作
statime	timestampz		操作的时间戳。这与记录到 GPDB 服务器日志文件的时间戳相同，以便在日志中查看操作的更多细节信息

pg_statistic

此表存储着与数据库内容有关的统计数据。记录由ANALYZE创建，并被查询规划器使用。被分析过的每个表的列都有一条记录。需注意的是，所有的统计信息天生就是近似值，哪怕是最新更新的。

该表还存储与索引表达式值有关的统计数据。这些会把他们当作实际的数据字段来描述，特别是starelid引用索引。不过，普通非表达式索引字段不会有这样的记录，因为会和下层的表字段记录重复。

因为不同类型的统计信息适用于不同类型的数据，该表被设计成不太在意存储的是什么类型的统计。只有极为常用的统计信息(比如NULL率)才会在该表中有专用的字段。其他所有的东西都存储在通用槽位中，槽位指的是一组相关的字段，其内容用槽位中的一个字段的代号表示。

该表不应该是公共可读的，因其即便是表内容的统计信息也应该作为敏感信息(比如薪水字段的最大最小值)。pg_stats是建立在pg_statistic上的全局可读视图，他只显示对向前用户可读的信息。

字段	类型	参考	描述
starelid	oid	pg_class.oid	所描述的字段所属的表或者索引
staattnum	smallint	pg_attribute.attnum	字段序号。普通字段是从 1 开始计数的。
stanullfrac	real		字段中 NULL 记录的比率
stawidth	integer		非 NULL 记录的平均存储宽度，以字节计算
stadistinct	real		字段中唯一非 NULL 数据值的数量。整数表示独立数值的实际数目，负数为数值出现概率倒数的负数(比如，一个字段的数值平均出现概率为 2，那么可以表示为 sradistinct=-0.5)。0 表示独立数值的数目未知
stakindN	smallint		数字代码,表示数据存储在第 n 个插槽的类型
staopN	oid	pg_operator.oid	用于生成存储在第 N 个槽位的统计信息的操作符。比如，一个统计槽位若显示<操作符，该操作符定义了该数据的排序顺序
stanumbersN	real[]		第 N 槽位相关类型的数值类型统计，如果该槽位和数值类型无关，则为 NULL
stavaluesN	anyarray		第 N 槽位相关类型的字段数据值，如果该槽位不存储任何类型数据值则为 NULL。每个数据的元素值实际是指定字段的数据类型，因此只能定义为 anyarray 类型

pg_tablespace

此表存储着有关可用的表空间的信息。将表放在特定的表空间中可以帮助管理磁盘布局。与多数的系统表不同，该表在GPDB系统中是全局共享的：每个库中只是一

个pg_tablespace的拷贝，而并非每个库中有一个独立的表。

字段	类型	参考	描述
spcname	name		表空间的名字
spcowner	oid	pg_authid.oid	表空间的所有者，通常为其创建者
spclocation	text[]		已弃用
spcacl	aclitem[]		表空间的访问权限
spcprilocations	text[]		已弃用
spcmirlocations	text[]		已弃用
spcfsoid	oid	pg_filespace.oid	该表空间使用的文件空间的对象标识符。一个文件空间定义了 Primary、Mirror 和 Master 的目录位置

pg_trigger

此表存储着表上的触发器。

字段	类型	参考	描述
tgrelid	oid	pg_class.oid	触发器所在的表
tgname	name		触发器的名字(同一个表上必须唯一)
tgfoid	oid	pg_proc.oid	要调用的函数
tgtype	smallint		标识触发器条件的位掩码
tgenabled	boolean		如果触发器启用则为真
tgisconstraint	boolean		如果触发器实现一个参照完整性约束则为真
tgconstrname	name		参照完整性约束的名字
tgconstrrelid	oid	pg_class.oid	参照完整性约束引用的表
tgdeferrable	boolean		如果可推迟则为真
tginitdeferred	boolean		如果初始被推迟则为真
tgnargs	smallint		传给触发器函数的参数个数
tgattr	int2vector		目前未使用
tgargs	bytea		传递给触发器的参数字符串，每个已 NULL 结尾

pg_type

此表存储着有关数据类型的信息。基本类型(标量类型)是用CREATE TYPE创建的，域是用CREATE DOMAIN创建的。数据库中的没个表会自动创建一个符合类型，以表示表的行结构。还可以用CREATE TYPE AS创建符合类型。

字段	类型	参考	描述
typname	name		数据类型的名字
typnamespace	oid	pg_namespace.oid	类型所在名字空间的 OID
typowner	oid	pg_authid.oid	类型的所有者
typlen	smallint		对于定长类型是该类型内部表现形式的字节数。变长类型为负数。-1 表示有长度属性的变长类型，-2 表示一个 NULL 结尾的 C 字符串
typbyval	boolean		内部过程传递这个类型的数值时传值还是传引用。如不是 1、2、4、8 字节长度或变长只能传引用，最好为假。可以传值的也可以为假，比如 float4 类型
typtype	char		b:基础类型，c:符合类型(比如一个表的类型)，d:域类型，p:伪类型
typisdefined	boolean		如果定义了类型则为真，如果是尚未定义的类型则为假。如果为假，除了类型名字、名字空间和 OID 之外没有可靠的信息
typdelim	char		分析数组输入时，分割两个此类型数值的字符。注意该分隔符是与数组元素数据类型相关，而不是与数组数据类型相关
typrelid	oid	pg_class.oid	如果是复合类型需要该字段指向 pg_class 中定义该表的记录。对于自由的复合类型，pg_class 记录不表示一个表，但需要他来查找该类型的 pg_attribute 记录。对于非复合类型为 0

typelem	oid	pg_type.oid	如不为 0 则标识 pg_type 的另一条记录。当前类型可以像数组使用下标的方式产生 typelem 类型。一个真正的数据类型是变长的(typlen=-1)，但一些定长(typlen>0)类型也有非零的 typelem(比如 name 和 point)。如果一个定长类型有一个 typelem，其内部必须是某个数目的 typelem 类型的数据，不能有其他数据。变长数据类型有一个数组自过程的都信息
typinput	regproc	pg_proc.oid	输入转换函数(文本格式)
typoutput	regproc	pg_proc.oid	输出转换函数(文本格式)
typreceive	regproc	pg_proc.oid	输入转换函数(二进制格式)，如果没有则为 0
typsend	regproc	pg_proc.oid	输出转换函数(二进制格式)，如果没有则为 0
typanalyze	regproc	pg_proc.oid	自定义的 ANALYZE 函数，如使用标准函数则为 0
typalign	char		存储此类型数据时要求的对齐格式。其应用在磁盘存储以及在 GPDB 内部的大多数形式。如果多个值连续存储，如一个完整的行插入磁盘，那么现在此类型数据前填充空白。对齐引用是该序列中第一个数据的开头。可能的值有： c:char 对齐，就是不需要对齐。s:short 对齐(多数机器上为 2 字节)。i:int 对齐(多数机器上为 4 字节)。d:double(多数机器上为 8 字节)
typstorage	char		对于变长类型(typlen=-1)确定如何分配存储方案以及缺省策略。可能的值为： p:平面存储。e:可以离线存储(如果有的话，参见 pg_class.reltoastrelid)。m:可以在线压缩。x:可以在线压缩或者离线存储。
typnotnull	boolean		表示类型上的一个非空约束。仅用于域(domain)
typbasetype	oid	pg_type.oid	域基于的类型。如果不是域则为 0
typtypmod	integer		域用 typtypmod 记录作用到基础类型上的 typmod(如基础类型不使用 typmod 则为-1)。如果此类型不是域，那么为-1
typndims	integer		如一个域是数组，那么 typndims 是数组的维数(就是说，typbasetype 是一个数组类型；域的 typelem 将匹配基本类型的 typelem)。非域非数组域为 0
typdefaultbin	text		如果为非 NULL ，它是该类型缺省表达式的 nodeToString()表现形式。目前这个字段只用于域
typdefault	text		如没有缺省值，typdefault 是 NULL。如 typdefaultbin 不是 NULL，typdefault 须包含一个 typdefaultbin 代表的缺省表达式可读版本。如果 typdefaultbin 为 NULL 但 typdefault 不是，那么 typdefault 是该类型缺省值的外部表现形式，可以把它交给该类型的输入转换器生成一个常量

pg_type_encoding

此表包含列存储类型的信息(不过笔者没找到这张表的用处)。

字段	类型	矫正值	存储	描述
typid	oid	not null	普通	不知是 pg_attribute 表中的那个键
tpyoptions	text[]		扩展	实际的选项信息

pg_window

此表存储着窗口函数的信息。窗口函数常用以构成复杂的OLAP(在线分析处理)查询。窗口函数用于通过一个查询实现分区的结果集。一个窗口分区是一个查询结果的子集，其通过OVER()子句定义。典型的窗口函数有rank，dense_rank，row_number。该表中的记录都是pg_proc中的扩展。pg_proc中保存着窗口函数的名字，输入输出数据类型，以及其他与普通函数类似的信息。

字段	类型	参考	描述
winfnoid	regproc	pg_proc.oid	窗口函数在 pg_proc 中的 OID
winrequireorder	boolean		窗口函数是否要求分区规范有一个 ORDER BY 子句

winallowframe	boolean		窗口函数是否允许分区规范有一个 ROW 或者 RANGE 框架子句
winpeercount	boolean		是否需要在分区中统计同等记录的计数。窗口函数必须向前访问以确保内部状态的可用性
wincount	boolean		窗口函数是否需要统计分区中的记录数
winfunc	regproc	pg_proc.oid	pg_proc 中对应的函数 OID
winprefunc	regproc	pg_proc.oid	计算偏值的递延型窗函数的预处理窗口函数的 OID
winpretype	oid	pg_type.oid	预处理窗口函数返回结果的类型
winfinfunc	regproc	pg_proc.oid	根据 winprefunc 返回的结果计算出最终结果的函数
winkind	char		窗口函数功能类型的标识符号： w:普通窗口函数。n:NTILE 函数。f:初值函数。l:末值函数。g:LAG 函数。LEAD 函数

第廿二章：管理模式 gp_toolkit

GP提供了一个名为gp_toolkit的管理模式，据此可以查询系统表，日志文件，以及针对系统状态的操作环境信息。此模式包含多个可以使用SQL命令访问的视图。此模式对于所有数据库用户都是可访问的，即便一些对象是需要超级用户权限。为了方便起见，可以将gp_toolkit模式添加到搜索路径中。例如：

```
=> ALTER ROLE myrole SET search_path TO myschema,gp_toolkit;
```

本章描述了此模式中最有用的一些视图。此视图中还有一些其他的对象(视图、函数、外部表等)未在本章中讲述。

需要日常维护的表的检查

下面的视图可以帮助识别哪些表需要维护(VACUUM和ANALYZE)。

- gp_bloat_diag
- gp_stats_missing

VACUUM或VACUUM FULL命令回收被删除或废弃的行占用的磁盘空间。由于GPDB采用的是MVCC事务并发机制，被删除或更新的记录会依然占用磁盘上的物理空间，虽然这些记录对于新的事务已经不可见。过期的记录会增加磁盘尺寸甚至降低表的扫描速度。

ANALYZE命令收集用于查询规划器的列级统计信息。GPDB使用依赖数据库统计信息的基于成本的查询规划器。精确的统计信息使得查询规划器可以更好的评估选择性和检索行数以达到选择最佳查询计划的目的。

---

gp_bloat_diag

此视图显示哪些表有膨胀(实际占用的磁盘页数超过统计信息所显示的预期磁盘页数)。膨胀表需要通过VACUUM或者VACUUM FULL以回收被删除或废弃的行占用的磁盘空间。此视图对于所有数据库用户都是可访问的。

字段	描述
bdirelid	表对象标识符
bdinspname	模式名称
bdirelname	表对象名字
bdirelpages	实际的磁盘页面数
bdiexppages	统计信息所显示的预期磁盘页数
bdidiag	膨胀诊断信息

---

gp_stats_missing

此视图显示那些没有统计信息且可能需要ANALYZE的表。

字段	描述
smischema	模式的名字
smitable	表的名字
smisize	此表是否有统计信息？如果系统表中没有记录统计或者行尺寸信息，

	则为假，这表明表需要被分析。如果表中没有任何记录也为假。例如：上层分区表总是空的，且该值总是为假。
smicols	表中的字段数
smirecs	表中的记录数

锁检查

当一个事务访问一个关系时(比如一张表)，其获得一个锁。根据获得锁的类型，后续的事务在可以访问同一个关系之前必须等待。更多关于锁类型的信息参见"GPDB中的锁模式"。GPDB资源队列(用作工作负载管理)同样使用锁来管理查询的准入。

gp_locks_*系列的视图可以帮助我们诊断查询及会话是否由于某个锁而等待访问特定对象。

- gp_locks_on_relation
- gp_locks_on_resqueue

gp_locks_on_relation

此视图显示当前正被锁定的关系，与该锁相关的查询会话信息。更多关于锁类型的信息参见"GPDB中的锁模式"。 此视图对于所有数据库用户都是可访问的。

字段	描述
lorlocktype	被锁的关系的类型：relation、extend、page、tuple、transactionid、object、userlock、resource queue、advisory
lordatabase	对象所在数据库的对象标识符，如果是共享对象则为 0
lorrelname	关系的名字
lorrelation	关系的对象标识符
lortransaction	锁影响到的事务标识符
lorpid	获取或等待锁的服务器进程号。如果为准备型事务则为 0
lormode	锁模式的名字
lorgranted	锁是(True)否(False)被授权
lorcurrentquery	会话当前的查询

gp_locks_on_resqueue

此视图显示当前正被锁定的资源队列，与该锁相关的查询会话信息。此视图对于所有数据库用户都是可访问的。

字段	描述
lorusename	执行当前会话的用户名字
lorrsqname	资源队列的名字
lorlocktype	可锁对象的类型：resource queue
lorobjid	锁定的事务对象标识符
lortransaction	锁影响到的事务标识符
lorpid	锁影响到的事务的服务器进程号

lormode	锁模式的名字
lorgranted	锁是(True)否(False)被授权
lorwaiting	会话是否正在等待

查看 GPDB 服务器日志文件

GPDB系统的每个组成部分(Master、Standby、Segment、Mirror)都有自己的日志文件。gp_log_*系列的视图允许通过SQL的方式来查询服务器日志文件以找到感兴趣的记录。访问这些视图需要超级用户权限。

- gp_log_command_timings
- gp_log_database
- gp_log_master_concise
- gp_log_system

gp_log_command_timings

此视图使用一个外部表来读取Master上的日志文件，报告数据库会话上SQL命令的执行时间。访问该视图需要超级用户权限。

字段	描述
logsession	会话标识符(以 con 作为前缀)
logcmdcount	一个会话中的命令编号(以 cmd 作为前缀)
logdatabase	数据库的名字
loguser	数据库用户的名字
logpid	进程号(以 p 作为前缀)
logtimemin	第一个日志信息的时间
logtimemax	最后一个日志信息的时间
logduration	语句从开始时间到最后时间的时间段

gp_log_database

此视图使用一个外部表来读取整个GPDB系统(Master、Segment、Mirror)的服务器日志文件，列出与当前数据库有关的日志记录。日志记录可以通过会话标识符(logsession)和命令标识符(logcmdcount)来辨识。访问该视图需要超级用户权限。

字段	描述
logtime	日志信息的时间戳
loguser	数据库用户的名字
logdatabase	数据库的名字
logpid	相关的进程号(以 p 作为前缀)
logthread	相关的线程号(以 th 作为前缀)
loghost	客户端的主机名或者 IP 地址
logport	客户端的端口号
logsessiontime	会话连接打开的时间

logtransaction	全局事务标识符
logsession	会话标识符(以 con 作为前缀)
logcmdcount	一个会话中的命令编号(以 cmd 作为前缀)
logsegment	节点数据标识符(对于 Primary 和 Mirror 分别以 seg 和 mir 作为前缀，Master 的数据标识符为-1)
logslice	分片的标识符(执行计划的分片)
logdistxact	分布式事务标识符
loglocalxact	本地事务标识符
logsubxact	子事务标识符
logseverity	LOG、ERROE、FATAL、PANIC、DEBUG1 或 DEBUG2
logstate	日至信息相关的 SQL 状态码
logmessage	日志或出错信息文本
logdetail	错误信息相关的详细信息
loghint	错误信息相关的提示信息
logquery	内部生成的查询文本
logquerypos	进入内部生成的查询文本的游标索引
logcontext	消息生成的上下文
logdebug	为调试用的带细节的查询字符串
logcursorpos	进入查询字符窜的游标索引
logfunction	生成该消息所在的函数
logfile	生成该消息所在的文件
logline	生成该消息所在的文件的行号
logstack	消息相关的堆栈跟踪信息的完整文本

gp_log_master_concise

此视图使用一个外部表来读取Master日志文件的字段的子集。访问该视图需要超级用户权限。

字段	描述
logtime	日志信息的时间戳
logdatabase	数据库的名字
logsession	会话标识符(以 con 作为前缀)
logcmdcount	一个会话中的命令编号(以 cmd 作为前缀)
logmessage	日志或出错信息文本

gp_log_system

此视图使用一个外部表来读取整个GPDB系统(Master、Segment、Mirror)的服务器日志文件，列出所有的日志记录。日志记录可以通过会话标识符(logsession)和命令标识符(logcmdcount)来辨识。访问该视图需要超级用户权限。

字段	描述
logtime	日志信息的时间戳
loguser	数据库用户的名字

logdatabase	数据库的名字
logpid	相关的进程号(以 p 作为前缀)
logthread	相关的线程号(以 th 作为前缀)
loghost	客户端的主机名或者 IP 地址
logport	客户端的端口号
logsessiontime	会话连接打开的时间
logtransaction	全局事务标识符
logsession	会话标识符(以 con 作为前缀)
logcmdcount	一个会话中的命令编号(以 cmd 作为前缀)
logsegment	节点数据标识符(对于 Primary 和 Mirror 分别以 seg 和 mir 作为前缀，Master 的数据标识符为-1)
logslice	分片的标识符(执行计划的分片)
logdistxact	分布式事务标识符
loglocalxact	本地事务标识符
logsubxact	子事务标识符
logseverity	LOG、ERROE、FATAL、PANIC、DEBUG1 或 DEBUG2
logstate	日至信息相关的 SQL 状态码
logmessage	日志或出错信息文本
logdetail	错误信息相关的详细信息
loghint	错误信息相关的提示信息
logquery	内部生成的查询文本
logquerypos	进入内部生成的查询文本的游标索引
logcontext	消息生成的上下文
logdebug	为调试用的带细节的查询字符串
logcursorpos	进入查询字符窜的游标索引
logfunction	生成该消息所在的函数
logfile	生成该消息所在的文件
logline	生成该消息所在的文件的行号
logstack	消息相关的堆栈跟踪信息的完整文本

检查服务器配置文件

GPDB系统的每个组成部分(Master、Standby、Segment、Mirror)都有自己的服务器配置文件(postgresql.conf)。下面的gp_toolkit对象可用来在全部的主节点检查postgresql.conf文件的参数设置：

- gp_param_setting('parameter_name')
- gp_param_settings_seg_value_diffs

gp_param_setting('parameter_name')

此函数根据服务器配置参数的名字，返回Master和每个激活的Segment的postgresql.conf中的值。该函数对所有用户都可以访问。

字段	描述

paramsegment	节点数据标识符(仅激活的节点被显示)。Master 的数据标识符为-1
paramname	参数的名字
paramvalue	参数的值

例如：

SELECT * FROM gp_param_setting('max_connections');

---

gp_param_settings_seg_value_diffs

被作为本地参数(这意味着每个Segment从自己的postgresql.conf文件获取参数值)的服务器配置参数，应该在所有Segment之间保持一致。此视图显示那些不一致的本地参数。不过那些被认为具备不同值(如port(端口))的参数不包含在内。该视图对所有用户都可以访问。

字段	描述
psdname	参数的名字
psdvalue	参数的值
psdcount	参数为该值的 Segment 数量

失败节点检查

视图gp_pgdatabase_invalid可用于检查失败的节点。

---

gp_pgdatabase_invalid

此视图显示在系统表中被标记为掉线的Segment的信息。该视图对所有用户都可以访问。

字段	描述
pgdbidbid	Segment 的 dbid。每个 Segment 有一个唯一的 dbid
pgdbiisprimary	此 Segment 当前是否作为 Primary 在运行
pgdbicontent	数据在节点上的标识符。主节点与镜像节点具有相同的 content 值
pgdbivalid	该节点当前是否启动且可用
pgdbidefinedprimary	此节点是否在系统初始化时被设置为主节点

资源队列活动和状态检查

资源队列的目的是限制系统中同时执行的查询的数量，以避免自动资源(如内存、CPU、磁盘IO)耗尽。所有的数据库用户都分配到一个资源队列，每个用户提交的语句都需要先针对资源队列的限制进行评估才能得到运行。gp_resq_*系列视图可用于检查通过各自资源队列提交到系统的语句的状态。需注意的是，超级用户提交的语句是不受资源队列限制的。

- gp_resq_activity
- gp_resq_activity_by_queue
- gp_resq_priority_statement
- gp_resq_role
- gp_resqueue_status

gp_resq_activity

对于那些有活动工作负载的资源队列，此视图为每个通过资源队列提交的活动语句显示一条记录。该视图对所有用户都可以访问。

字段	描述
resqprocpid	该语句相关的进程号(在 Master 上)
resqrole	用户的名字
resqoid	资源队列的对象标识符
resqname	资源队列的名字
resqstart	语句提交到系统的时间
resqstatus	语句的状态：running、waiting、cancelled

gp_resq_activity_by_queue

对于那些有活动工作负载的资源队列，此视图显示队列活动的一个概要。该视图对所有用户都可以访问。

字段	描述
resqoid	资源队列的对象标识符
resqname	资源队列的名字
resqlast	最后发到该资源队列的语句的时间
resqstatus	最后一条语句的状态：running、waiting、cancelled
resqtotal	此资源队列中的语句总数

gp_resq_priority_statement

此视图显示GPDB系统中正在运行的语句的资源队列优先级、会话标识符、以及一些其他信息。该视图对所有用户都可以访问。

字段	描述
rqpdatname	该会话连接到的数据库名字
rqpusename	提交语句的用户名字
rqpsession	会话标识符
rqpcommand	会话中的命令编号(命令编号结个会话标识符唯一识别语句)
rqppriority	该语句的资源队列优先级(MAX、HIGH、MEDIUM、LOW)
rqpweight	该语句的优先级对应的一个 Integer 值
rqpquery	查询语句的文本内容

gp_resq_role

此视图显示了资源队列与相关的角色。该视图对所有用户都可以访问。

字段	描述
rrrolname	角色(用户)的名字

rrrsqname	分配给该角色的资源队列的名字。如果一个角色没有明确的分配资源队列，其将属于缺省资源队列(pg_default)

gp_resqueue_status

此视图允许管理员查看工作负载管理资源队列的状态和活动。其显示了对于系统中特定的资源队列来说，多少语句处于等待状态、多少语句正在执行。

字段	描述
queueid	资源队列的标识符
rsqname	资源队列的名字
rsqcountlimit	资源队列的活动语句限制阀值。-1 意味着无限制
rsqcountvalue	资源队列中当前正被使用的资源槽位数量
rsqcostlimit	资源队列的成本限制阀值。-1 意味着无限制
rsqcostvalue	资源队列中当前正被使用的成本总和
rsqmemorylimit	资源队列的内存限制
rsqmemoryvalue	资源队列中当前正被使用的内存总和
rsqwaiters	资源队列中处于等待状态的语句的数量
rsqholders	资源队列中处于正在执行状态的语句的数量

查看用户和组(角色)

将用户分组一起管理对象权限会更方便，这样的话，一个组可以一起被授权或者撤销权限。在GPDB中通过创建作为组的角色来实现，将特定的用户角色设置为该组角色的成员。

视图gp_roles_assigned可用以查看系统中的所有角色，以及分配给他们的成员(如果该角色同样作为组角色的话)。

gp_roles_assigned

此视图显示系统中的所有角色，以及分配给他们的成员(如果该角色同样作为组角色的话)。该视图对所有用户都可以访问。

字段	描述
raroleid	角色对象标识符。如果该角色有成员(用户)，则被作为组角色
rarolename	角色(用户或者组)的名字
ramemberid	该角色的成员的角色对象标识符
ramembername	该角色的成员的角色的名字

检查数据库对象尺寸和磁盘空间

gp_size_*系列视图可用于确定Database、Schema、Table或Index的磁盘空间使用情况。下面的视图计算所有Primary(Mirror不包含在内)实例上的对象尺寸总和。

- gp_size_of_all_table_indexes
- gp_size_of_database

- gp_size_of_index
- gp_size_of_partition_and_indexes_disk
- gp_size_of_schema_disk
- gp_size_of_table_and_indexes_disk
- gp_size_of_table_and_indexes_licensing
- gp_size_of_table_disk
- gp_size_of_table_uncompressed
- gp_disk_free

表和索引的尺寸视图是按照关系的对象标识符显示，而不是名字。要通过名字确定表和索引的尺寸，必须通过pg_class表的关系名字(relname)。例如：

```
SELECT relname as name, sotdsize as size, sotdtoastsize as toast, sotdadditionalsize as other
FROM gp_size_of_table_disk as sotd, pg_class
WHERE sotd.sotdoid=pg_class.oid ORDER BY relname;
```

gp_size_of_all_table_indexes

此视图显示每个表的所有索引的尺寸之和。该视图对所有用户都可以访问，不过非超级用户只能查看到那些有访问权限的关系。

字段	描述
soatioid	表的对象标识符
soatisize	表的所有索引尺寸之和(字节为单位)
soatischemaname	模式的名字
soatitablename	表的名字

gp_size_of_database

此视图显示数据库的总的尺寸。该视图对所有用户都可以访问，不过非超级用户只能查看到那些有访问权限的关系。

字段	描述
soddatname	数据库的名字
soddatsize	数据库的尺寸(字节为单位)

gp_size_of_index

此视图显示索引的尺寸。该视图对所有用户都可以访问，不过非超级用户只能查看到那些有访问权限的关系。

字段	描述
soioid	索引的对象标识符
suitableoid	索引所属的表的对象标识符
soisize	所引导尺寸(字节为单位)
soiindexschemaname	索引所属的模式的名字
soiindexname	索引的名字
suitableschemaname	索引所属的表的模式的名字
suitablename	索引所属的表的名字

gp_size_of_partition_and_indexes_disk

此视图显示分区表及其索引的尺寸。该视图对所有用户都可以访问，不过非超级用户只能查看到那些有访问权限的关系。

字段	描述
sopaidparentoid	父表的对象标识符
sopaidpartitionoid	分区表的对象标识符
sopaidpartitiontablesize	分区表的尺寸(字节为单位)
sopaidpartitionindexessize	分区表的所有索引的尺寸总和
sopaidparentschemaname	父表的模式的名字
sopaidparenttablename	父表的名字
sopaidpartitionschemaname	分区表的模式的名字
sopaidpartitiontablename	父表的名字

gp_size_of_schema_disk

此视图显示当前数据库中的模式的尺寸。该视图对所有用户都可以访问，不过非超级用户只能查看到那些有访问权限的关系。

字段	描述
sosdnsp	模式的名字
sosdschematablesize	模式中表尺寸的总和(字节为单位)
sosdschemaidxsize	模式中索引尺寸的总和(字节为单位)

gp_size_of_table_and_indexes_disk

此视图显示表及其索引的尺寸。该视图对所有用户都可以访问，不过非超级用户只能查看到那些有访问权限的关系。

字段	描述
sotaidoid	表的对象标识符
sotaidtablesize	表的尺寸
sotaididxsize	表的所有索引的尺寸总和
sotaidschemaname	模式的名字
sotaidtablename	表的名字

gp_size_of_table_and_indexes_licensing

此视图针对许可目的显示表及其索引的尺寸。该视图需要超级用户才可以访问。

字段	描述
sotailoid	表的对象标识符
sotailtablesizedisk	表的尺寸
sotailtablesizeuncompressed	如果该表为 AO 压缩表，显示该表未压缩的尺寸
sotailindexessize	表的所有索引的尺寸总和

sotailschemaname	模式的名字
sotailtablename	表的名字

gp_size_of_table_disk

此视图显示表在磁盘上的尺寸。该视图对所有用户都可以访问，不过非超级用户只能查看到那些有访问权限的关系。

字段	描述
sotdoid	表的对象标识符
sotdsize	表是总尺寸(字节为单位，包括超大对象属性，AO 表的额外存储对象等)
sotdtoastsize	TOAST 表(超大属性存储的位置)的尺寸，如果存在的话
sotdadditionsize	AO 表的额外存储信息的尺寸
sotdschemaname	模式的名字
sotdtablename	表的名字

gp_size_of_table_uncompressed

此视图显示压缩的AO表的未压缩尺寸，否则显示的是表在磁盘上的尺寸。该视图需要超级用户才可以访问。

字段	描述
sotuoid	表的对象标识符
sotusize	如果是压缩的 AO 表则为未压缩的尺寸，否则显示的是表在磁盘上尺寸
sotuschemaname	模式的名字
sotutablename	表的名字

gp_disk_free

此外部表在活动的节点上运行df(disk free)命令并汇总结果。未活动的镜像不包含在计算中。该视图需要超级用户才可以访问。

字段	描述
dfsegment	数据在节点上标识符(仅有活动节点被显示)
dfhostname	实例所在的主机的主机名
dfdevice	设备的名字
dfspace	实例所在的文件系统的空闲磁盘空闲空间(千字节为单位)

检查不平坦的数据分布

在GPDB中的所有表都是分布的，这意味着它们的数据被分割到系统的所有节点上。如果数据分布的不平坦，查询的性能可能会受到影响。下面的视图可以帮助诊断一张表是否出现了数据不平坦分布。

- gp_skew_coefficients
- gp_skew_idle_fractions

gp_skew_coefficients

此视图通过计算各实例之间的差异系数显示数据分布的倾斜。该视图对所有用户都可以访问，不过非超级用户只能查看到那些有访问权限的关系。

字段	描述
skcoid	表的对象标识符
skcnamespace	表定义的名字空间
skcrelname	表的名字
skccoeff	差异系统是通过标准差除以平均值得到的。这样既考虑到了平均数也考虑到了差异性。值越小越好。越大的值表示数据倾斜越严重

gp_skew_idle_fractions

此视图通过计算表扫描期间的系统空闲百分比显示数据分布的倾斜，其作为数据处理倾斜的指标。该视图对所有用户都可以访问，不过非超级用户只能查看到那些有访问权限的关系。

字段	描述
sifoid	表的对象标识符
sifnamespace	表定义的名字空间
sifname	表的名字
siffraction	表扫描期间的系统空闲百分比，其作为数据分布或者查询处理的倾斜指标。例如，0.1 表示 10%的倾斜，0.5 表示 50%的倾斜，等等。对于出现 10%倾斜的表，应该对其分布策略进行评估

第廿三章：经验分享

搞了这么久以翻译为主的事情，是该添加一些不一样的内容了。该章节将不定期的添加自己的经验分享，如果有同等爱好的朋友可以联系本人补充发布。

查看数据分布情况

不合理的分布策略会导致数据分布的倾斜(SKEW)，这对于GP来说是很糟糕的。分析数据的分布情况是诊断性能问题一个不可缺少的环节。

多数使用过GP的技术人员都知道，GP保留了一个名为gp_segment_id的虚拟字段用以区分不同实例之间的数据标识，而gp_segment_id对应gp_segment_configuration表的content字段。
于是我们可以使用这样的语句来查看数据在instance上的分布情况：

```
select gp_segment_id,count(*) fom $tablename group by 1 order by 1;
```

然而，有时我们还很想知道分区表中各分区之间数据的分布情况，关于partition的标识，我们没有太好的办法，GP基于Postgresql而来，GP中partition表实现了类似PG中的inherits，在parent级别的表中并没有存储实际的数据，数据存储在最底层的表中，也就是我们最关心的分区表，这样我们就可以用表信息包含的一个虚拟字段tableoid来区分数据所在的分区，比如：

```
select tableoid::regclass,count(*) from tbname group by 1 order by 1;
```

还可以将两者结合使用：

```
select gp_segment_id,tableoid::regclass,count(*) from tbname group by 1,2 order by 1,2;
```

RAID 条带科学化

这里需要提醒所有的用户，RAID的条带大小(STRIP SIZE)属于硬件级别的设置。如果要修改存有数据的RAID条带大小将会导致数据丢失，因此务必确保在生产系统安装之初就选择好最合理的条带大小。个人认为，对不同的硬件环境，没有所谓的通用的最佳值可以选择，最好通过实验的方式获得这个最佳值。目前诸多比较高级的RAID卡已经可以从4KB支持到1MB的带宽，这中间有256倍的差异，因此务必经过实验的方式验证最佳条带大小。

实验方式可以参考：
设置特定条带大小，然后安装好OS，GPDB以及GPCC，使用GPCC监测gpcheckperf的执行。按照如下步骤筛选出最佳的条带大小。

- 将条带设置到最大值，记录下gpcheckperf获得的IO值，此为最大IO带宽
- 将条带设置到最小值，记录下gpcheckperf获得的IOPS值，此为最大IOPS
- 采用二分法设置条带大小，直至gpcheckperf获得的IO值和IOPS值与前两步测试的结果都比较吻合为止，此时获得的条带大小为最佳值

注意：因为GPDB更多应用于OLAP类型的操作，建议在有两种最佳值选择时，考虑倾向确保IO带宽的一项，但也不能过多牺牲IOPS性能，个中原因暂不做细说。不过切记不要盲目听从伪专家的指导，以实测为准。