

Greenplum数据库® 6.0 文档

管理员指南

安全性配置指南

最佳实践指南

工具指南

参考指南

Greenplum 平台扩展框架(PXF)

Greenplum database 5 管理员指南

Greenplum database 4 管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“[反馈](#)”按钮提交详细信息

Greenplum数据库文档的源代码在Greenplum数据库和Pivotal Greenplum数据库。 Pivotal Greenplum数据库的某些特性未在开源Greenplum数据库代码，但在Greenplum数据库文档中进行了描述。

通常，术语“Pivotal Greenplum数据库”是指Pivotal的商业支持版本的Greenplum数据库，在描述仅商业Greenplum数据库软件支持的特性时在本文档中使用。 术语“Greenplum数据库”是指开源Greenplum数据库和Pivotal Greenplum数据库中都可用的特性。

在包含主要特性文档的部分中提到了仅Pivotal Greenplum数据库支持的特性。

以下是Pivotal Greenplum数据库特性，这些特性在开源Greenplum数据库中不受支持，但在本文档中可能会记录或引用。

- 开源Greenplum数据库存储库中不包含用于生成Pivotal Greenplum数据库二进制安装程序包的代码。
- Greenplum数据库不包括EMC DD Boost集成。 仅当在Greenplum主机上将Data Domain系统作为NFS共享挂载时，才支持备份到EMC Data Domain设备。
- Greenplum数据库不包括Symantec NetBackup集成。
- QuickpZ压缩方法在Greenplum数据库中不可用。 在CREATE TABLE [AS]操作中指定QuickLZ compressstype 值时将被忽略，但它将在随后的对该表的INSERT 或SELECT 操作中引起错误。
- 仅Pivotal Greenplum数据库支持不推荐使用的gpcheck 管理工具及其替代gpsupport 。
- 要将Greenplum平台扩展框架（PXF）与开源Greenplum数据库一起使用，必须分别构建和安装PXF服务器软件。 请参阅Greenplum数据库和PXF存储库中的PXF README文件中的构建说明。
- 本文档中有关联系Pivotal技术支持的建议仅适用于Pivotal Greenplum数据库客户。

Greenplum数据库® 6.0 文档

[管理员指南](#)[安全性配置指南](#)[最佳实践指南](#)[工具指南](#)[参考指南](#)[Greenplum平台扩展框架\(PXF\)](#)[Greenplum database 5 管理员指南](#)[Greenplum database 4 管理员指南](#)[FAQ](#)

如果您发现翻译不妥之处，欢迎点击“[反馈](#)”按钮提交详细信息

Greenplum数据库文档的源代码在Greenplum数据库和Pivotal Greenplum数据库。 Pivotal Greenplum数据库的某些特性未在开源Greenplum数据库代码，但在Greenplum数据库文档中进行了描述。

通常，术语“Pivotal Greenplum数据库”是指Pivotal的商业支持版本的Greenplum数据库，在描述仅商业Greenplum数据库软件支持的特性时在本文档中使用。 术语“Greenplum数据库”是指开源Greenplum数据库和Pivotal Greenplum数据库中都可用的特性。

在包含主要特性文档的部分中提到了仅Pivotal Greenplum数据库支持的特性。

以下是Pivotal Greenplum数据库特性，这些特性在开源Greenplum数据库中不受支持，但在本文档中可能会记录或引用。

- 开源Greenplum数据库存储库中不包含用于生成Pivotal Greenplum数据库二进制安装程序包的代码。
- Greenplum数据库不包括EMC DD Boost集成。 仅当在Greenplum主机上将Data Domain系统作为NFS共享挂载时，才支持备份到EMC Data Domain设备。
- Greenplum数据库不包括Symantec NetBackup集成。
- QuickpZ压缩方法在Greenplum数据库中不可用。 在CREATE TABLE [AS]操作中指定QuickLZ compressstype 值时将被忽略，但它将在随后的对该表的INSERT 或SELECT 操作中引起错误。
- 仅Pivotal Greenplum数据库支持不推荐使用的gpcheck 管理工具及其替代gpsupport 。
- 要将Greenplum平台扩展框架（PXF）与开源Greenplum数据库一起使用，必须分别构建和安装PXF服务器软件。 请参阅Greenplum数据库和PXF存储库中的PXF README文件中的构建说明。
- 本文档中有关联系Pivotal技术支持的建议仅适用于Pivotal Greenplum数据库客户。

Greenplum数据库® 6.0文档

管理员指南

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

有关配置、管理和监控Greenplum数据库安装以及管理、监控和使用数据库的信息。该指南还包含有关Greenplum数据库架构和概念（例如并行处理）的信息。

- **Greenplum数据库概念**

这一节给出了Greenplum数据库组件和特性的概述，例如高可用性、并行数据装载特性以及管理工具。

- **管理一个Greenplum系统**

这一节描述了一个Greenplum数据库系统管理员所执行的基本系统管理任务。

- **管理Greenplum数据库访问**

保护Greenplum数据库，包括通过网络配置、数据库用户身份验证、加密来保护对数据库的访问。

- **定义数据库对象**

这一节包括Greenplum数据库中的数据定义语言 (DDL) 以及如何创建和管理数据库对象。

- **分布与倾斜**

Greenplum数据库依赖于跨节点的均匀数据分布。

- **插入, 更新, 和删除数据**

这一节提供了Greenplum数据库中有关操纵数据和并发访问的信息。

- **查询数据**

本主题提供在Greenplum数据库中使用SQL的信息。

- **使用外部数据**

外部表和外表都可以访问存储在Greenplum数据库之外的数据源中的数据，就好像数据存储在常规数据库表中一样。您可以从外部表和外表读取和写入数据。

- **装载和卸载数据**

这一节中的主题描述了Greenplum数据库中将数据装载进来和写出去的方法，以及如何格式化数据文件。

- **性能管理**

这一节的内容是Greenplum数据库的性能管理，其中包含了如何监控，以及如何通过配置工作量来进行资源调用的优先级管理。



Greenplum数据库® 6.0文档

□ 安全性配置指南

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“[反馈](#)”按钮提交详细信息

这份指南描述了如何保护一个Greenplum数据库系统。这份指南假定读者有Linux/UNIX系统管理和数据库管理系统的知识。熟悉结构查询语言（SQL）也很有帮助。

因为Greenplum数据库是基于PostgreSQL 9.4的，这份指南假定读者对PostgreSQL比较熟悉。这份指南中通篇都为Greenplum数据库中类似的特性提供了对PostgreSQL文档的引用。[PostgreSQL documentation](#)

这份指南为负责管理Greenplum数据库系统的系统管理员提供信息。

- [保护数据库](#)
介绍Greenplum数据库的安全性主题。
- [Greenplum数据库端口和协议](#)
列出Greenplum集群内使用的网络端口和协议。
- [配置客户端认证](#)
描述认证Greenplum数据库客户端的可用方法。
- [配置数据库授权](#)
描述如何通过使用角色和权限在用户级别限制对数据库数据的授权访问。
- [审计](#)
描述被记录并且应该被监控以检测安全性威胁的Greenplum数据库事件。
- [加密数据和数据库连接](#)
描述如何在数据库中或者在网络上传输时加密数据以防止窃听攻击或者中间人攻击。
- [安全性最佳实践](#)
这一章描述用户应该遵循的基本安全性最佳实践，它可以确保最高级别的系统安全性。



Greenplum数据库® 6.0文档

□ Greenplum数据库最佳实践

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

最佳实践是一种方法或技术，它得到的结果始终比用其他方法得到的要好。最佳实践通过实践得来并且被证明能可靠地得到想要的结果。通过利用所有可用来确保成功的知识和技能，最佳实践是一种正确且最佳使用任意产品的保证。

本文档不教授用户如何使用Greenplum数据库的特性。关于如何使用和实现特定的Greenplum数据库特性，请参考Greenplum数据库文档其他部分提供的参考连接。这份文档主要致力于描述在设计、实现和使用Greenplum数据库时可遵循的最佳实践。

本文档的目的并非覆盖整个产品或者特性的概要，而是提供Greenplum数据库中要点的摘要。本文档不会介绍边缘用例。当然边缘用例能更好的利用Greenplum数据库特性并从中受益，但是这些用例要求熟练运用用户已有的Greenplum知识和技能，以及对环境（包括SQL访问、查询执行、并发、负载和其他因素）的深入理解。

通过掌握这些最佳实践，用户将能提升其Greenplum数据库集群在维护、支持、性能和扩展性领域的成功率。

- **最佳实践概要**

本章包含Greenplum数据库的最佳实践概要。

- **系统配置**

这一节的主题描述配置Greenplum数据库集群主机的要求和最佳实践。

- **模式设计**

这一节包含设计Greenplum数据库模式的最佳实践。

- **采用资源组管理内存和资源**

采用资源组管理Greenplum数据库资源。

- **采用资源队列管理内存和资源**

避免内存错误，管理Greenplum数据库资源。

- **系统监控和维护**

本节介绍日常运维相关的最佳实践，关注这些内容可以确保Greenplum数据库日常高可用性和性能保持最佳状态。

- **装载数据**

本节描述了将数据装载到Greenplum数据库的不同方式。

- **安全性**

最佳实践可以确保最高级别的系统安全性。



- **加密数据和数据库连接**

本节描述了有关实现加密和管理密钥的最佳实践。

- **SQL查询调优**

Greenplum数据库基于代价的优化器会权衡所有执行查询的策略并选择代价最小的策略去执行。

- **高可用性**

当用户启用并且正确地配置Greenplum高可用特性时，Greenplum数据库支持高度可用的、容错的数据库服务。要保证达到要求的服务等级，每个组件都必须有一个备用组件以保证在它失效时及时顶上。

Greenplum数据库® 6.0文档

工具指南

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“[反馈](#)”按钮提交详细信息

本文档介绍命令行工具和客户端程序的参考信息。

- [管理工具参考](#)

本章描述了Greenplum数据库提供的命令行管理工具相关内容

- [客户端工具参考](#)

描述Greenplum数据库随附的命令行客户端工具。

- [附加程序](#)

本节介绍Greenplum数据库安装后的可用附加程序



Greenplum数据库® 6.0文档

参考指南

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

Greenplum数据库系统的参考信息，包括SQL命令，系统catalog，环境变量，服务器配置参数，字符集支持，数据类型和Greenplum数据库扩展。

- [SQL Command Reference](#)
- [SQL 2008可选特性兼容性](#)
- [Greenplum环境变量](#)
- [保留标识符和SQL关键字](#)
- [系统目录参考](#)
- [gp_toolkit管理模式](#)
- [gpperfmon 数据库](#)
- [Greenplum 数据库数据类型](#)
- [字符集支持](#)
- [服务器配置参数](#)
- [内置函数摘要](#)
- [Greenplum MapReduce规范](#)
- [Greenplum PL/pgSQL过程语言](#)
- [Greenplum PL/R 语言扩展](#)
- [Greenplum PL/Python语言扩展](#)
- [Greenplum PL/Container语言扩展](#)
- [Greenplum的PL/Java语言扩展](#)
- [Greenplum的PL/Perl语言扩展](#)
- [用于分析的Greenplum MADlib扩展](#)
- [附加提供的模块](#)
- [Greenplum Partner Connector API](#)
- [后台工作进程](#)
- [Greenplum 特点概要](#)



Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

- [PXF介绍](#)
- [PXF管理手册](#)
- [使用PXF访问hadoop](#)
- [使用PXF访问Azure, Google云端存储, Minio和S3对象存储](#)

使用PXF访问SQL数据库
(JDBC)

PXF故障排除

- [PXF实用程序手册](#)

[Back to the Administrator Guide](#)

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“[反馈](#)”按钮提交详细信息

Greenplum平台扩展框架(PXF) 通过内置连接器提供跨异构数据源的并行、高吞吐量的数据访问和联合跨异构数据源的查询，该连接器将Greenplum数据库外部表定义映射到外部数据源。PXF源于Apache HAWQ项目。

- [PXF介绍](#)

本主题介绍PXF概念和用法。

- [PXF管理手册](#)

介绍了PXF的管理，包括安装，配置，初始化，升级和管理过程。

- [使用PXF访问hadoop](#)

介绍了PXF Hadoop连接器，它们支持的数据类型以及可用于读取和写入HDFS的配置文件。

- [使用PXF访问Azure, Google云端存储, Minio和S3对象存储](#)

介绍了PXF对象存储连接器，它们支持的数据类型以及可用于从对象库读取数据和向对象库写入数据的配置文件。

- [使用PXF访问SQL数据库\(JDBC\)](#)

介绍如何使用PXF JDBC连接器读取和写入外部SQL数据库，如Postgres或MySQL等。

- [PXF故障排除](#)

介绍了PXF的服务和数据库级日志记录配置过程。它还标识了一些常见的PXF错误，并描述了如何解决PXF内存问题。

- [PXF实用程序手册](#)

介绍了工具命令的使用信息

Greenplum数据库® 6.0文档

 管理员指南 **Greenplum数据库概念** 管理一个Greenplum系统 管理Greenplum数据库访问 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

 查询数据 使用外部数据 装载和卸载数据 性能管理

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

这一节给出了Greenplum数据库组件和特性的概述，例如高可用性、并行数据装载特性以及管理工具。

- **关于Greenplum的架构**

Greenplum数据库是一种大规模并行处理（MPP）数据库服务器，其架构特别针对管理大规模分析型数据仓库以及商业智能工作负载而设计。

- **关于管理和监控工具**

Greenplum数据库提供了标准的命令行工具来执行通常的监控和管理任务。

- **关于Greenplum数据库中的并发控制**

Greenplum数据库使用了PostgreSQL的多版本并发控制（MVCC）模型来管理对于堆表的并发事务。

- **关于并行数据装载**

这一主题给出了对于Greenplum数据库数据装载特性的简短介绍。

- **关于Greenplum数据库中的冗余和故障切换**

这一主题给出了Greenplum数据库高可用性特性的一个高层次的概述。

- **关于Greenplum数据库中的数据库统计信息**

Parent topic: Greenplum数据库管理员指南



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

□ 性能管理

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

这一节描述了一个Greenplum数据库系统管理员所执行的基本系统管理任务。

- **关于Greenplum数据库发布版本号**

Greenplum的版本号和其组织方式的改变，界定了从一个版本到下一个版本做了那些修改。

- **启动和停止Greenplum数据库**

在一个Greenplum数据库管理系统中，所有主机上的数据库实例（Master和所有的Segment）一起被启动或者停止，启停操作统一由Master实例发起，它们步调一致，在外界看来是一个完整的数据库管理系统。

- **访问数据库**

这个主题描述了几种可用来连接Greenplum数据库的客户端工具以及如何建立一个数据库会话。

- **配置Greenplum数据库系统**

服务器配置参数会影响Greenplum数据库的行为。

- **启用压缩**

可以利用Greenplum数据库自身的特性或工具来配置启用数据压缩。

- **启用高可用和数据持久化特征**

Greenplum数据库的容错和高可用特征可以通过配置启用

- **备份和恢复数据库**

这个主题描述如何使用Greenplum的备份和恢复特性。

- **扩容Greenplum系统**

为了放大性能和存储能力，通过向集群增加主机来扩容用户的Greenplum系统。

- **监控Greenplum系统**

可以用系统中包含的各种工具以及附加组件来监控一个Greenplum数据库系统。

- **日常系统维护任务**

要保持一个Greenplum数据库系统高效运行，必须对数据库定期清理过期数据并且更新表统计信息，这样查询优化器才能有准确的信息。

- **Recommended Monitoring and Maintenance Tasks**

This section lists monitoring and maintenance activities recommended to ensure high availability and consistent performance of your Greenplum Database cluster.

Parent topic: Greenplum

Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统
 - **管理Greenplum数据库访问**
 - 定义数据库对象
- 分布与倾斜
- 插入, 更新, 和删除数据
- 查询数据
- 使用外部数据
- 装载和卸载数据
- 性能管理

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

保护Greenplum数据库，包括通过网络配置、数据库用户身份验证、加密来保护对数据库的访问。

- 配置客户端认证

本主题讲解如何为Greenplum数据库配置客户端访问和身份验证。

- 管理角色与权限

Greenplum数据库授权机制存储访问数据库中数据库对象的角色和权限，并使用SQL语句或命令行实用程序进行管理。

Parent topic: [Greenplum数据库管理员指南](#)



Greenplum数据库® 6.0文档

 管理员指南 Greenplum数据库概念 管理一个Greenplum系统 管理Greenplum数据库访问 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

 查询数据 使用外部数据 装载和卸载数据 性能管理

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

这一节包括Greenplum数据库中的数据定义语言 (DDL) 以及如何创建和管理数据库对象。

在Greenplum数据库中创建对象包括为数据分布、存储选项、数据装载以及其他影响数据库系统性能的其他Greenplum特征做出预先的选择。理解可用的选项以及如何使用数据库将有助于用户做出正确的决定。

大部分的Greenplum高级特性都是SQL CREATE DDL语句的扩展。

- [创建和管理数据库](#)
- [创建和管理表空间](#)
- [创建和管理SCHEMA](#)
- [创建和管理表](#)
- [选择表存储模型](#)
- [对大型表分区](#)
- [创建和使用序列](#)
- [在Greenplum数据库中使用索引](#)
- [创建和管理视图](#)

Parent topic: [Greenplum数据库管理员指南](#)



Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统
 - 管理Greenplum数据库访问
 - 定义数据库对象
 - 分布与倾斜**
- 插入, 更新, 和删除数据
 - 查询数据
 - 使用外部数据
 - 装载和卸载数据
 - 性能管理
- Greenplum database 5管理员指南
- Greenplum database 4管理员指南
- FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

Greenplum数据库依赖于跨节点的均匀数据分布。

在MPP无共享环境中，查询的总体响应时间由所有节点的完成时间来度量。系统只能与最慢的节点一样快。如果数据偏斜，具有更多数据的节点将花费更多时间来完成，因此每个节点必须具有大致相等的行数并执行大致相同的处理量。如果一个节点具有比其他节点更多的处理数据，则可能导致性能不佳和内存不足。

大表做连接操作时，最佳分布至关重要。要执行连接，匹配的行必须位于同一节点上。如果数据未在同一连接列上分发，则其中一个表所需的行将动态重新分发到其他节点。有些情况下，执行广播动作，每个节点将其各个行发送到所有其他节点上，而不是每个节点重新哈希数据并根据哈希值将行发送到适当的节点的重新分配。

Parent topic: [Greenplum数据库管理员指南](#)

本地(Co-located)连接

使用在所有节点上均匀分布表并导致本地连接的哈希分布可以提供显着的性能提升。当连接的行位于同一节点时，可以在节点实例中完成大部分处理。这些连接被称为本地或*co-located*连接。本地连接最小化数据移动；每个节点独立于其他节点运营，没有节点之间的网络流量或通信。

要做到经常进行连接的大型表进行本地连接，请在同一列上分布表。本地连接要求连接的两端分布在相同的列上（并且顺序相同），并且在连接表时使用分布子句中的所有列。分发列也必须是相同的数据类型 - 尽管不同数据类型的某些值可能看起来具有相同的表示，但它们以不同方式存储并且哈希值不同，因此它们存储在不同的节点上。

数据倾斜

数据倾斜可能是由于错误选择分布键或单个元组表插入或复制操作导致的数据分布不均匀。出现在表级别的数据倾斜通常是查询性能不佳和内存不足情况的根本原因。倾斜的数据会影响扫描（读取）性能，它也会影响所有其他查询执行操作，例如，连接和按操作。

验证分布以确保数据在初始加载后均匀分布非常重要。在增量加载后继续验证分布同样重要。

以下查询显示每个节点的行数以及最小和最大行数的方差：

```
SELECT 'Example Table' AS "Table Name",
       max(c) AS "Max Seg Rows", min(c) AS "Min Seg Rows",
       (max(c)-min(c))*100.0/max(c) AS "Percentage Difference
Between Max & Min"
FROM (SELECT count(*) c, gp_segment_id FROM facts GROUP BY
2) AS a;
```

`gp_toolkit` schema里有两个view可以供你检查倾斜。

- `gp_toolkit.gp_skew_coefficients` view通过计算存储在每个节点上的数据的变异系数 (CV) 来显示数据分布倾斜。`skccoeff`列显示变异系数 (CV)，计算为标准差除以平均值。它考虑了数据系列平均值附近的平均值和可变性。值越低越好。值越高表示数据倾斜越大。
- `gp_toolkit.gp_skew_idle_fractions` view通过计算表扫描期间空闲的系统百分比来显示数据分布倾斜，这是计算倾斜的指示。`siffraction`列显示在表扫描期间空闲的系统百分比。这是数据分布不均匀或查询处理倾斜的指标。例如，值0.1表示10%偏斜，值0.5表示50%偏斜，依此类推。倾斜超过10%的表应评估其分配策略。

随机分布表的注意事项

当创建一个随机分布表(DISTRIBUTED RANDOMLY)，Greenplum数据库以轮询方式将您插入或复制的数据分发到表中。Greenplum数据库为每个插入或复制操作启动轮询周期的新实例。因此，将随机分布的表中的单个元组插入分配给第一个节点，将导致数据偏斜。Greenplum数据库更均匀地分布来自批量插入的数据。

复制表的注意事项

当您创建复制表 (CREATE TABLE子句DISTRIBUTED REPLICATED) 时，Greenplum数据库会将每个表行分发到每个节点实例。复制表的数据均匀分布，因为每个节点具有相同的行。使用复制表上的`gp_segment_id`系统列来验证均匀分布的数据的查询将失败，因为Greenplum数据库不允许查询引用复制表的系统列。

处理倾斜

当不成比例的数据流向一个或几个节点并由其处理时，则要处理倾斜结果。它往往是Greenplum数据库性能和稳定性问题的罪魁祸首。它可能发生在诸如连接，排序，聚合和各种OLAP操作之类的操作中。处理倾斜在执行查询中发生，并且不容易检测数据倾斜。

如果单个节点失败，即不是主机上的所有节点，则可能是处理倾斜问题。识别处理倾斜目前是一个手动过程。首先查看分裂文件。如果存在倾斜，但不足以导致分裂，则不会成为性能问题。如果确定存在倾斜，则找到导致倾斜的查询。以下是要使用的步骤和命令。（更改名称，例如传递给gpssh的主机文件名）：

1. 找到要监视倾斜处理的数据库的OID：

```
SELECT oid, datname FROM pg_database;
```

示例输出：

oid	datname
17088	gpadmin
10899	postgres
1	template1
10898	template0
38817	pws
39682	gpperfmon
(6 rows)	

2. 运行gpssh命令检测集群所有节点的文件大小。使用先前命令将<OID>替换为数据库的OID：

```
[gpadmin@mdw kend]$ gpssh -f ~/hosts -e \
"du -b /data[1-
2]/primary/gpseg*/base/<OID>/pgsql_tmp/*" | \
grep -v "du -b" | sort | awk -F" " '{ arr[$1] =
arr[$1] + $2 ; tot = tot + $2 } ; END \
{ for ( i in arr ) print "Segment node" i, arr[i],
"bytes (" arr[i]/(1024**3)" GB)"; \
print "Total", tot, "bytes (" tot/(1024**3)" GB)" }'
```

示例输出：

```
Segment node[sdw1] 2443370457 bytes (2.27557 GB)
Segment node[sdw2] 1766575328 bytes (1.64525 GB)
Segment node[sdw3] 1761686551 bytes (1.6407 GB)
Segment node[sdw4] 1780301617 bytes (1.65804 GB)
Segment node[sdw5] 1742543599 bytes (1.62287 GB)
Segment node[sdw6] 1830073754 bytes (1.70439 GB)
Segment node[sdw7] 1767310099 bytes (1.64594 GB)
```

```
Segment node[sdw8] 1765105802 bytes (1.64388 GB)
Total 14856967207 bytes (13.8366 GB)
```

如果磁盘使用率存在显着且持续的差异，则应调查正在执行的查询是否存在可能的倾斜（上面的示例输出未显示出明显的倾斜）。在监控系统中，总会存在一些倾斜，但通常是短暂的，持续时间很短。

3. 如果出现显着且持续的偏斜，则下一个任务是识别违规查询。

上一步中的命令汇总了全部节点。这次，找到实际的节点目录。您可以从master执行此操作，也可以登录到上一步中标识的特定节点。以下是从master运行的示例。

此示例专门查看排序文件。并非所有分裂文件或倾斜情况都是由排序文件引起的，因此您需要自定义命令：

```
$ gpssh -f ~/hosts -e
"ls -l /data[1-
2]/primary/gpseg*/base/19979/pgsql_tmp/*"
| grep -i sort | awk '{sub(/base.*tmp\//, "...",
$10); print $1,$6,$10}' | sort -k2 -n
```

如下是该命令的输出：

```
[sdw1] 288718848
/data1/primary/gpseg2/.../pgsql_tmp_slice0_sort_17758_0001
291176448

/data2/primary/gpseg5/.../pgsql_tmp_slice0_sort_17764_0001
924581888

/data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_00
980582400

/data1/primary/gpseg18/.../pgsql_tmp_slice10_sort_29425_00
986447872

/data2/primary/gpseg35/.../pgsql_tmp_slice10_sort_29602_00
[sdw5] 999620608

/data1/primary/gpseg26/.../pgsql_tmp_slice10_sort_28637_00
999751680

/data2/primary/gpseg9/.../pgsql_tmp_slice10_sort_3969_0001
1000112128

/data1/primary/gpseg13/.../pgsql_tmp_slice10_sort_24723_00
1000898560

/data2/primary/gpseg28/.../pgsql_tmp_slice10_sort_28641_00
[sdw8] 1008009216

/data1/primary/gpseg44/.../pgsql_tmp_slice10_sort_15671_00
```

```
1008566272
```

```
/data1/primary/gpseg24/.../pgsql_tmp_slice10_sort_28633_00
1009451008

/data1/primary/gpseg19/.../pgsql_tmp_slice10_sort_29427_00
1011187712

/data1/primary/gpseg37/.../pgsql_tmp_slice10_sort_18526_00
1573741824

/data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_00
1573741824

/data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_00
1573741824

/data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_00
1573741824

/data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_00
1573741824

/data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_00
1573741824

/data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_00
1573741824
```

扫描此输出显示主机sdw8上的节点gpseg45是罪魁祸首，因为其排序文件比输出中的其他文件大。

4. 使用ssh登录到有问题的节点并切换到root。使用lsof命令找到排序文件所属进程的PID：

```
[root@sdw8 ~]# lsof
/data2/primary/gpseg45/base/19979/pgsql_tmp/pgsql_tmp_slic

COMMAND PID USER FD TYPE DEVICE SIZE
NODE NAME
postgres 15673 gpadmin 11u REG 8,48 1073741824
64424546751
/data2/primary/gpseg45/base/19979/pgsql_tmp/pgsql_tmp_slic
```

PID, 15673, 也是文件名的一部分，但并不是总是这样。

5. 使用ps命令找到PID并确认数据库和连接信息：

```
[root@sdw8 ~]# ps -eaf | grep 15673
gpadmin 15673 27471 28 12:05 ?          00:12:59
postgres: port 40003, sbaskin bdw
172.28.12.250(21813) con699238 seg45 cmd32
slice10 MPPEXEC SELECT
root      29622 29566  0 12:50 pts/16    00:00:00 grep
15673
```

6. 在master上，检查上一个命令（sbaskin），连接（con699238）和命令（cmd32）中用户的pg_log日志文件。具有这三个值的日志文件中的行应该是包含查询的行，但有时，命令编号可能略有不同。例如，ps输出可能显示cmd32，但在日志文件中它是cmd34。如果查询仍在运行，则用户和连接的最后一个查询是违规查询。

几乎在所有情况下处理倾斜的解决方法是重写查询。创建临时表可以消除倾斜。临时表可以随机分配以强制进行两阶段聚合。

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

□ 性能管理

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

这一节提供了Greenplum数据库中有关操纵数据和并发访问的信息。

这个主题包括下列小标题：

- [关于Greenplum数据库中的并发控制](#)
- [插入行](#)
- [更新现有行](#)
- [删除行](#)
- [使用事务](#)
- [全局死锁检测](#)
- [清理数据库](#)

Parent topic: [Greenplum数据库管理员指南](#)

关于Greenplum数据库中的并发控制

Greenplum数据库和PostgreSQL不为并发控制使用锁。它们使用一种多版本模型来维护数据一致性，即多版本并发控制（MVCC）。MVCC为每一个数据库会话实现了事务隔离，并且每一个查询事务会看到一个数据的快照。这保证该事务会看到一致的不受其他并发事务影响的数据。

因为MVCC不会为并发控制使用显式锁，锁竞争被最小化并且Greenplum数据库在多用户环境中维持了合理的性能。为查询（读取）数据获得的锁不与为写数据获得的锁冲突。

Greenplum数据库提供了多种锁模式来控制对表中数据的并发访问。大部分Greenplum数据库的SQL命令自动获取适当的锁来确保在命令执行期间被引用的表不会被删除或者被以不兼容的方式被修改。对于不能轻易适应于MVCC行为的应用，可以使用 `LOCK` 命令来获取显式锁。不过，MVCC的正确使用通常能提供更好的性能。

Table 1. Greenplum数据库中的锁模式

锁模式	相关的SQL命令	冲突模式
ACCESS SHARE	<code>SELECT</code>	ACCESS EXCLUSIVE
ROW SHARE	<code>SELECT FOR SHARE</code> , <code>SELECT...FOR</code>	EXCLUSIVE, ACCESS EXCLUSIVE

	UPDATE	
ROW EXCLUSIVE	INSERT, COPY See Note .	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, SELECT...FOR UPDATE See Note .	ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE
ACCESS EXCLUSIVE	ALTER TABLE, DROP TABLE, TRUNCATE, REINDEX, CLUSTER, VACUUM FULL	ACCESS SHARE, ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE

Note: 注意: Greenplum数据库会为 UPDATE DELETE 和 SELECT...FOR UPDATE 获取更加严格的EXCLUSIVE锁。(而不是PostgreSQL中的 ROW EXCLUSIVE).

当分布式死锁检测开启时, 运行在堆表上的 `DELETE`, `UPDATE` 和 `SELECT...FOR UPDATE` 的锁模式是 `ROW EXCLUSIVE` 的。详情请见[全局死锁检测](#).

插入行

使用 `INSERT` 命令在一个表中创建行。这个命令要求该表的名称和表中每一个列的值; 可以选择性地以任意顺序指定列名。如果没有指定列名, 以那些列在表中的顺序列出数据值, 用逗号分隔它们。

例如, 指定要插入的列名和值:

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

只指定要插入的值:

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

通常, 数据值都是常量, 但也可以使用标量表达式。例如:

```
INSERT INTO films SELECT * FROM tmp_films WHERE date_prod <  
'2016-05-07';
```

可以在单个命令中插入多行。例如:

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

要插入数据到一个分区表, 应指定根分区表, 即用 `CREATE TABLE` 命令创建的表。也可以在一个 `INSERT` 命令中指定该分区表的一个叶子子表。如果数据对于指定的叶子子表无效, 会返回一个错误。不支持在 `INSERT` 命令中指定一个不是叶子子表的子表。

要插入大量数据, 使用外部表或者 `COPY` 命令。对于插入大量行, 这些装载机制比 `INSERT` 更加有效。更多有关批量数据装载的信息请见[装载和卸载数据](#)。[装载和卸载数据](#)

追加优化表的存储模型是为批量数据装载而优化。Greenplum不推荐对追加优化表的单行 `INSERT` 语句。对于追加优化表, Greenplum数据

库支持最多127个并发INSERT 事务插入到一个追加优化表。

更新现有行

UPDATE命令在一个表中更新行。可以更新一个表中所有的行、所有行的一个子集或者单个行。可以单独更新每一列而不影响其他列。

要执行一次更新，需要：

- 要更新的表和列的名称
- 这些列的新值
- 指定要更新的行的一个或者更多条件。

例如，下面的命令把所有价格为5 的产品更新为价格为 10:

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

在Greenplum数据库中使用 UPDATE由下列限制:

- GPORCA可以为Greenplum分布键列提供更新支持，Postgres planner则不会。
- 如果使用了镜像，不能在UPDATE语句中使用STABLE或VOLATILE 函数。
- Greenplum数据库的分区列不能被更新.

删除行

DELETE命令从一个表中删除行。指定一个WHERE子句可以删除满足特定条件的行。如果不指定WHERE子句，该表中所有的行都会被删除。其结果是一个合法的但为空的表。例如，从产品表中删除所有价格为10的行：

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

要从一个表中删除所有行：

```
DELETE FROM products;
```

在Greenplum数据库中使用 DELETE具有和使用UPDATE类似的限制:

- 如果使用了镜像, 不能在 UPDATE 语句中使用STABLE 或VOLATILE 函数。

截断一个表

使用TRUNCATE命令可以快速地移除一个表中的所有行。例如:

```
TRUNCATE mytable;
```

这个命令在一次操作中清空一个表的所有行。注意TRUNCATE不扫描该表, 因此它不会处理继承的子表或者ON DELETE 的重写规则。该命令只截断所提到的表中的行。

使用事务

事务允许用户把多个SQL语句捆绑在一个要么全做要么全不做的操作中。

下面是Greenplum数据库的SQL事务命令:

- BEGIN 或者 START TRANSACTION 开始一个事务块。
- END 或者 COMMIT 提交一个事务的结果。
- ROLLBACK 放弃一个事务而不做任何更改。
- SAVEPOINT 在一个事务中标记一个位置并且允许做部分回滚。用户可以回滚在一个保存点之后执行的命令但保留该保存点之前执行的命令。
- ROLLBACK TO SAVEPOINT 回滚一个事务到一个保存点。
- RELEASE SAVEPOINT 销毁一个事务内的保存点。

事务隔离级别

Greenplum数据库接受下列标准SQL事务级别:

- 读未提交 和 读已提交 的行为像标准的 读已提交.
- 可重复读 和 可序列化 的行为像behave like 可重复读.

下列信息描述了Greenplum事务级别的行为:

读已提交/读未提交

Greenplum 数据库不允许任何命令来看其他并发事务的未提交的更新, 因此 读未提交 会和 读已提交 的行为一样。读已提交 提供快速、简单、部分的事务隔离。使用读已提交和读未提交事务隔离, SELECT, UPDATE 和DELETE 事务在一个查询开始时取得的数据库快照上操作。

一个 SELECT 查询:

- 看得见该查询开始前被提交的数据。
- 看得见在该事务内执行的更新。
- 看不见事务外未提交的数据。
- 如果并发事务在该查询所在事务最初的读操作之前就被提交, 该查询可能会看到这个并发事务所作的更改。

如果其他并发事务在同一个事务中后续的SELECT查询开始前提交更改, 这些查询能够看到不同的数据。 UPDATE和 DELETE命令只找在该命令开始前提交的行。

读已提交事务隔离允许并发事务在UPDATE或 DELETE找到行之前修改或者锁定该行。 读已提交或读未提交事务隔离可能不适合执行复杂查询和更新并且要求该数据库的一致性视图的应用。 读已提交 事务隔离可能不适合执行 复杂查询和更新并且要求该数据库的一致性视图的应用。

可重复读和可序列化

根据SQL标准规定, 可序列化 事务隔离确保在其中事务的并行化执行结果就像一个接一个执行一样。 如果指定 可序化时, Greenplum数据库会退而采用可重复读· 可重复读事务能防止脏读、不可重复读和幻读, 而且不需要昂贵的锁定, 但是在Greenplum数据库的一些SERIALIZABLE事务之间可能发生其他的相互影响而阻止它们变成真正地可序列化。 并发运行的事务应该被检查来识别出不会因为不允许对同一数据的并发更新而被阻止的相互影响。 通过使用显式表锁或者要求冲突事务更新一个被引入来表示该冲突的虚拟行可以阻止所发现的问题。

对于 可重复读 事务, 一个 SELECT查询:

- 看得到一个事务开始时（不是该事务中当前查询开始时）的数据快照。
- 只看得到在查询开始前被提交的数据。
- 看得到该事务内执行的更新。
- 看不到该事务外部的未提交数据。
- 看不到并发事务所作的更改。
- 一个单一事务中的后续 SELECT 命令总是看到相同的数据。
- UPDATE, DELETE, SELECT FOR UPDATE, 和 SELECT FOR SHARE 命令只会发现在该命令开始前被提交的行。如果一个目标行被找到时一个并发事务已经更新、删除或者锁定该行，可重复读事务会等待该并发事务执行或者回滚。如果并发事务执行变化，那么可重复读事务会回滚。如果可重复读事务回滚，那么可序列化或者可重复读事务执行变化。

Greenplum数据库中的默认事务隔离级别是读已提交。要为一个事务更改隔离级别，在BEGIN 该事务时声明隔离级别或者在事务开始后使用SET TRANSACTION命令设置隔离级别。

全局死锁检测

Greenplum数据库全局死锁检测器后端进程会收集所有segment上的锁信息，并使用有向算法来检测本地死锁和全局死锁是否存在。该算法使Greenplum数据库放宽对堆表的并发更新和删除限制。

(Greenplum数据库仍然在AO / CO表上使用表级锁定，对UPDATE, DELETE 和 SELECT...FOR UPDATE 的并发操作进行限制。)

默认情况下，全局死锁检测器是被禁用的，Greenplum数据库以串行方式对堆表执行并发更新和删除操作。可以通过设置配置参数`gp_enable_global_deadlock_detector`，开启并发更新并让全局死锁检测器检测死锁是否存在。

启用全局死锁检测器后，当启动Greenplum数据库时，master 主机上会自动启动一个后端进程。可以通过`gp_global_deadlock_detector_period` 配置参数，来设置采集和分析锁等待数据的时间间隔。

如果全局死锁检测器发现了死锁，它会通过取消最新的事务所关联的一个或多个后端进程来避免死锁。

当全局死锁检测器发现了以下事物类型的死锁时，只有一个事务将成功。其他事务将失败，并打印错误指出不允许对同一行进行并发更

新。

- 在同一行堆表的并发事务中, 第一个事务是更新操作, 下一个事务执行更新或删除, 并且查询计划包含一个动作操作符。
- 堆表的同一分发键上的并发更新事务由Greenplum Database Postgres查询优化器执行
- 哈希表的同一行上的并发事务更新由GPORCA优化器执行。

Note: Greenplum数据库通过配置参数 `deadlock_timeout` 指定本地死锁检测的间隔。由于本地死锁检测和全局死锁检测算法的不同, 被死锁检测器终止的进程也不同, 这取决于本地死锁检测和全局死锁检测哪个先被触发。

Note: 若打开配置参数 `lock_timeout`, 且将数值设定为小于 `deadlock_timeout` 和 `gp_global_deadlock_detector_period`, 查询会在死锁检测被触发之前就被终止。

用户可以通过自定义函数, `pg_dist_wait_status()` 查看所有segment上的锁等待信息: 哪些事务在等待锁, 哪些事务在持有锁, 以及事务运行在哪个segment上。`pg_dist_wait_status()` 的输出如下例所示:

```
SELECT * FROM pg_dist_wait_status();
  segid | waiter_dxid | holder_dxid | holdTillEndXact
-----+-----+-----+-----
      -1 |        29 |        28 | t
       0 |        28 |        26 | t
       0 |        27 |        29 | t
       1 |        26 |        27 | t
(4 rows)
```

全局死锁检测器会终止某个事务以打破死锁, 并且打印如下的错误信息:

```
ERROR: canceling statement due to user request: "cancelled by global deadlock detector"
```

清理数据库

虽然新事务看不到被删除或者被更新的数据行, 但是它们仍然在磁盘上占用物理空间。周期性地运行VACUUM命令可以移除这些过期的行。例如:

```
VACUUM mytable;
```

VACUUM命令会收集表级别的统计信息，例如行数和页数。在装载数据后清理所有的表，包括追加优化表。有关推荐的例行清理操作的信息，请见 [例行清理和分析](#)。

Important: 如果在数据库数据上频繁地执行更新和删除，VACUUM、VACUUM FULL和 VACUUM ANALYZE命令应该被用来维护Greenplum数据库中的数据。有关使用 VACUUM命令的信息请见*Greenplum*数据库参考指南。

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ **查询数据**

□ 使用外部数据

□ 装载和卸载数据

□ 性能管理

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

本主题提供在Greenplum数据库中使用SQL的信息。

用户可以使用`psql`交互式SQL客户端和其他客户端工具输入SQL语句来查看、更改以及分析数据库中的数据。

- **关于Greenplum的查询处理**

这个主题给出了Greenplum数据库如何处理查询的概述。理解这一处理有助于编写和调优查询。

- **关于GPORCA**

在Greenplum数据库中，默认的GPORCA优化器与传统查询优化器共存。

- **定义查询**

Greenplum数据库基于PostgreSQL的SQL标准实现。

- **WITH查询（公用表表达式）**

`WITH`子句提供在一个更大的`SELECT`查询中，使用子查询或执行数据修改操作的方式。你可以在`INSERT`, `UPDATE`, 或 `DELETE`命令中使用`WITH`子句。

- **使用函数和操作符**

用户定义和内置函数和运算符的说明。

- **使用JSON数据**

Greenplum数据库支持`json`和`jsonb`数据类型来存储JSON (JavaScript Object Notation)数据。

- **使用XML数据**

Greenplum数据库支持存储XML数据的`xml`数据类型。

- **使用全文搜索**

Greenplum数据库提供用于查询自然语言文档的数据类型，函数，运算符，索引类型和配置。

- **查询性能**

Greenplum数据库动态地消除一个表中的不相关分区并且为查询中的不同操作符最优化地分配内存。

- **管理查询生成的溢出文件**

如果没有足够的内存让一个SQL查询在内存中运行，Greenplum数据库会在磁盘上创建溢出文件（也被称为工作文件）。

- **查询分析**

检查性能不好的查询的查询计划，来确定可能的性能调优机会。

Parent topic: Greenplum数据库管理员指南

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

□ 性能管理

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

外部表和外表都可以访问存储在Greenplum数据库之外的数据源中的数据，就好像数据存储在常规数据库表中一样。您可以从外部表和外表读取和写入数据。

外部表是Greenplum数据库可以支持存储在数据库之外的数据的表。

您可以创建一个可读的外部表来从外部数据源读取数据，也可以创建一个可写的外部表以将数据写入外部数据源。您可以像在常规数据库表中一样在SQL命令中使用外部表。例如，您可以使用SELECT（可读外部表），INSERT（可写外部表）以及将外部表与其他Greenplum表连接。外部表通常用于加载和卸载数据库数据。有关使用外部表访问外部数据的更多信息，请参阅[定义外部表](#)[定义外部表](#)。

[使用PXF访问外部数据](#) 描述了使用PXF和外部表来访问外部数据源。

外表是Greenplum数据库可以支持存储在数据库之外的数据的表。您既可以读取也可以写入同一个外表。您可以类似如上所述的外部表一样，在SQL命令中使用外表。有关使用外部访问外部数据的更多信息，请参阅[使用外部表访问外部数据](#)。

基于Web的外部表提供对HTTP服务器或操作系统进程所服务的数据的访问。有关基于Web的表的更多信息，请参阅[创建和使用外部Web表](#)。

- [定义外部表](#)

外部表允许把外部文件当作常规数据库表来访问。它们常常被用来把数据移进或者移出Greenplum数据库。

- [使用PXF访问外部数据](#)

您组织管理的数据可能已存在于外部源中，例如Hadoop，对象存储库和其他SQL数据库。Greenplum平台扩展框架（PXF）通过内置连接器提供对此外部数据的访问，该连接器将外部数据源映射到Greenplum数据库表定义。

- [使用外部表访问外部数据](#)

- [使用Greenplum的并行文件服务器（gpfdist）](#)

gpfdist 协议用于CREATE EXTERNAL TABLE SQL命令，以访问Greenplum Database gpfdist文件服务器实用程序提供的外部数据。当外部数据由gpfdist 提供时，Greenplum数据库系统中的所有节点都可以并行读取或写入外部表数据。



Parent topic: [Greenplum数据库管理员指南](#)

Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统
 - 管理Greenplum数据库访问
 - 定义数据库对象
 - 分布与倾斜
 - 插入, 更新, 和删除数据
 - 查询数据
 - 使用外部数据
 - 装载和卸载数据**
 - 性能管理

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

这一节中的主题描述了Greenplum数据库中将数据装载进来和写出去的方法，以及如何格式化数据文件。

Greenplum数据库支持高性能的并行数据装载和卸载，并且支持对较少数据、单个文件的非并行数据导入和导出。

Greenplum数据库可以对多种类型的外部数据源读出和写入，包括文本文件、Hadoop文件系统、亚马逊S3以及Web服务器。

- SQL命令COPY在Master主机上的一个外部文本文件或在Segment主机上的多个文本文件和一个Greenplum数据库表之间传输数据。
- 外部表允许用户使用如SELECT、JOIN或者SORT EXTERNAL TABLE DATA等SQL命令并行地直接查询位于数据库之外的数据。用户可以为外部表创建视图。外部表常常被用在一个如CREATE TABLE *table* AS SELECT * FROM *ext_table*之类的命令中，用于装载外部数据到一个普通的数据库表中
- 外部Web表提供对动态数据的访问。它们背后可以是来自于用HTTP协议访问的URL的数据或者一个运行在一个或者更多Segment上的OS脚本的输出数据。
- gpfdist工具是Greenplum数据库的并行文件分发程序。它是一个配合外部表使用的HTTP服务器，它允许Greenplum数据库的Segment并行地从多个文件系统装载外部数据。用户可以在不同的主机和网络接口上运行多个gpfdist的实例并且并行访问它们。
- gupload工具使用gpfdist和一个YAML格式的控制文件使装载任务的步骤自动化。
- 用户可以使用Greenplum Platform Extension Framework (PXF) 创建可读写的外部表，并使用这些表将数据加载到Greenplum数据库或从中卸载数据。如想了解如何使用PXF，请参阅[使用PXF访问外部数据](#)。

装载数据的方法选择取决于源数据的特性—它的位置、尺寸、格式以及是否需要转换。

在最简单的情况下，SQL命令COPY从一个对Greenplum数据库Master实例可访问的文本文件中装载数据到一个表中。这样做不需要设置并且可以为较少的数据提供很好的性能。通过COPY命令，数据可以在Master主机上的单个文件和数据库之间来回复制。限制数据集的总尺寸为外部文件所在文件系统的容量并且把数据限制为一个单一的文件写流。

对于大型数据集的更高效的数据装载选项利用了Greenplum数据库

的MPP架构，使用Greenplum数据库的Segment并行装载数据。这些方法允许数据从多个文件系统通过多个主机上的多个NIC被装载，达到非常高的数据传输率。外部表允许用户从数据库中像访问普通数据库表一样访问外部文件。在与Greenplum数据库的并行文件分发程序gpfdist一起使用时，外部表通过使用所有的Greenplum数据库Segment来装载或卸载数据达到完全并行。

Greenplum数据库利用了Hadoop分布式文件系统的并行架构来访问其上的文件。

- [使用外部表装载数据](#)
- [装载和写入非HDFS自定义数据](#)
- [处理装载错误](#)
- [用gupload装载数据](#)
- [使用PXF访问外部数据](#)

您组织管理的数据可能已存在于外部源中，例如Hadoop，对象存储库和其他SQL数据库。Greenplum平台扩展框架（PXF）通过内置连接器提供对此外部数据的访问，该连接器将外部数据源映射到Greenplum数据库表定义。

- [使用gpfdist和gupload转换外部数据](#)

gpfdist并行文件服务器允许用户设置转换，使Greenplum数据库外部表能够以CREATE EXTERNAL TABLE命令的FORMAT子句不支持的格式读取和写入文件。*input*转换以外部数据格式读取文件，并以CSV或外部表的FORMAT子句中指定的其他文本格式将行输出到gpfdist。*output*转换以文本格式从gpfdist接收行，并将它们转换为外部数据格式。
- [用COPY装载数据](#)
- [在单行错误隔离模式中运行COPY](#)
- [优化数据装载和查询性能](#)
- [从Greenplum数据库卸载数据](#)
- [格式化数据文件](#)
- [自定义数据访问协议实例](#)

Parent topic: [Greenplum数据库管理员指南](#)

Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统
 - 管理Greenplum数据库访问
 - 定义数据库对象
 - 分布与倾斜
 - 插入, 更新, 和删除数据
 - 查询数据
 - 使用外部数据
 - 装载和卸载数据
 - 性能管理

这一节的内容是Greenplum数据库的性能管理，其中包含了如何监控，以及如何通过配置工作量来进行资源调用的优先级管理。

- **管理性能**

系统性能管理包含了性能计算，确定性能问题的造成原因，以及使用可用的工具和技术来解决问题的应用。

- **性能问题的常见原因**

这一节解释了常见性能问题的排查以及对这些问题可能的解决方案。

- **Greenplum数据库内存总览**

在Greenplum数据库系统中，内存是一项关键的资源，对其的高效使用可以保证优秀的性能和输出，这一节描述了Segment主机内存是在Segment之间的分配方式，以及管理员可用的配置方法。

- **管理资源**

Greenplum数据为用户提供了根据业务需求对查询进行排序以及提供资源的功能，该功能的主要目的是为了防止在资源不可用时仍然发起查询的行为。

- **检修性能问题**

这一节为在一个Greenplum数据库系统中确定和解决性能问题提供了指导。

Parent topic: [Greenplum数据库管理员指南](#)

[Greenplum database 5管理员指南](#)

[Greenplum database 4管理员指南](#)

[FAQ](#)

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息



Greenplum数据库® 6.0文档

管理员指南

安全性配置指南

最佳实践指南

工具指南

参考指南

Greenplum平台扩展框架(PXF)

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“[反馈](#)”按钮提交详细信息

Greenplum数据库文档的源代码在Greenplum数据库和Pivotal Greenplum数据库。 Pivotal Greenplum数据库的某些特性未在开源Greenplum数据库代码，但在Greenplum数据库文档中进行了描述。

通常，术语“Pivotal Greenplum数据库”是指Pivotal的商业支持版本的Greenplum数据库，在描述仅商业Greenplum数据库软件支持的特性时在本文档中使用。 术语“Greenplum数据库”是指开源Greenplum数据库和Pivotal Greenplum数据库中都可用的特性。

在包含主要特性文档的部分中提到了仅Pivotal Greenplum数据库支持的特性。

以下是Pivotal Greenplum数据库特性，这些特性在开源Greenplum数据库中不受支持，但在本文档中可能会记录或引用。

- 开源Greenplum数据库存储库中不包含用于生成Pivotal Greenplum数据库二进制安装程序包的代码。
- Greenplum数据库不包括EMC DD Boost集成。 仅当在Greenplum主机上将Data Domain系统作为NFS共享挂载时，才支持备份到EMC Data Domain设备。
- Greenplum数据库不包括Symantec NetBackup集成。
- QuickpZ压缩方法在Greenplum数据库中不可用。 在CREATE TABLE [AS]操作中指定QuickLZ compressstype 值时将被忽略，但它将在随后的对该表的INSERT 或SELECT 操作中引起错误。
- 仅Pivotal Greenplum数据库支持不推荐使用的gpcheck 管理工具及其替代gpsupport 。
- 要将Greenplum平台扩展框架（PXF）与开源Greenplum数据库一起使用，必须分别构建和安装PXF服务器软件。 请参阅Greenplum数据库和PXF存储库中的PXF README文件中的构建说明。
- 本文档中有关联系Pivotal技术支持的建议仅适用于Pivotal Greenplum数据库客户。

Greenplum数据库® 6.0文档

□ 安全性配置指南

保护数据库

Greenplum数据库端口和协议

配置客户端认证

配置数据库授权

审计

加密数据和数据库连接

安全性最佳实践

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

[如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息](#)

介绍Greenplum数据库的安全性主题。

安全性配置的目的是配置Greenplum数据库服务器以消除尽可能多的安全性缺陷。这份指南为最小安全性需求提供了底线，并且有额外的安全性文档对其进行补充。

根本的安全性需求有下面几类：

- [认证](#) 覆盖Greenplum数据库服务器支持并且用来建立客户端应用标识的机制。
- [授权](#) 与数据库用来授权客户端访问使用的特权和权限模型相关。
- [审计](#), 或者日志设置覆盖Greenplum数据库中跟踪成功或者失败的用户动作的可用日志选项。
- [数据加密](#) 阐述可用来保护静态数据以及传输中数据的加密功能。这包括与Greenplum数据库相关的安全性证书。

访问Kerberos化的Hadoop集群

用户可以使用Greenplum平台扩展框架（PXF）进行读写Hadoop文件系统中的外部表。如果Hadoop集群受到Kerberos的保护

（Kerberos化），Greenplum数据库必须被配置为允许外部表拥有者用Kerberos认证。执行这种设置的步骤请见[为保护HDFS配置PXF](#)。

平台加固

平台加固涉及通过遵循最佳实践和实施联邦安全性标准来评估以及最小化系统弱点。加固产品基于US国防部的指导方针“安全性模板实现指南（STIG）”。加固工作移除不必要的包、禁用不需要的服务、设置严格的文件和目录权限、移除无主的文件和目录、为单用户模式执行认证并且为最终用户提供选项以配置服从最新STIG的包。

Parent topic: [安全性配置指南](#)



Greenplum数据库® 6.0文档

□ 安全性配置指南

保护数据库

Greenplum数据库端口和协议

配置客户端认证

配置数据库授权

审计

加密数据和数据库连接

安全性最佳实践

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮
提交详细信息

列出Greenplum集群内使用的网络端口和协议。

Greenplum数据库客户端用TCP在客户端连接端口（默认为5432）上连接到Greenplum的Master实例。这个监听端口可以在 `postgresql.conf` 配置文件中重新配置。客户端连接使用PostgreSQL的libpq API。`psql`命令行接口、一些Greenplum工具以及特定语言的编程API也将直接使用libpq库或者在内部实现libpq协议。

每一个Segment实例也有一个客户端连接端口，它被Master实例单独用来与Segment协调数据库操作。在Greenplum的Master上执行的 `gpstate -p` 命令，会列出Greenplum的Master以及主Segment和镜像Segment的端口分配。例如：

```
[gpadmin@mdw ~]$ gpstate -p
20190403:02:57:04:011030 gpstate:mdw:gpadmin-[INFO]:-Starting gpstate with
args: -p
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-local Greenplum
Version: 'postgres (Greenplum Database) 5.17.0 build
commit:fc9a9d4cad8dd4037b9bc07bf837c0b958726103'
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-master Greenplum
Version: 'PostgreSQL 8.3.23 (Greenplum Database 5.17.0 build
commit:fc9a9d4cad8dd4037b9bc07bf837c0b958726103) on x86_64-pc-linux-gnu,
compiled by GCC gcc (GCC) 6.2.0, 64-bit compiled on Feb 13 2019 15:26:34'
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-Obtaining Segment
details from master...
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:--Master segment
instance /data/master/gpseg-1 port = 5432
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:--Segment instance port
assignments
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-----
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:- Host Datadir
Port
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/primary/gpseg0 20000
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/mirror/gpseg0 21000
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/primary/gpseg1 20001
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/mirror/gpseg1 21001
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/primary/gpseg2 20002
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/mirror/gpseg2 21002
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/primary/gpseg3 20000
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/mirror/gpseg3 21000
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/primary/gpseg4 20001
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/mirror/gpseg4 21001
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/primary/gpseg5 20002
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/mirror/gpseg5 21002
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/primary/gpseg6 20000
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/mirror/gpseg6 21000
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/primary/gpseg7 20001
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/mirror/gpseg7 21001
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/primary/gpseg8 20002
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-/data/mirror/gpseg8 21002
```

诸如后备复制、Segment镜像、统计信息收集以及Segment之间的数据交换等特性也会创建额外的Greenplum数据库网络连接。数据库启动时会创建一些持久连接而在查询执行之类的操作期间会创建临时连接。用于查询执行处理、数据移动以及统计信息收集的临时连接使用TCP和UDP协议在1025到65535范围内的可用端口。

Note: 为避免在初始化Greenplum数据库时Greenplum数据库与其他应用程序之间的端口冲突, 请不要在操作系统参数net.ipv4.ip_local_port_range指定的范围内指定Greenplum数据库端口。例如, 如果net.ipv4.ip_local_port_range = 10000-65535, 用户可以将Greenplum数据库基本端口号设置为该值以外的范围:

```
PORT_BASE = 6000
MIRROR_PORT_BASE = 7000
REPLICATION_PORT_BASE = 8000
MIRROR_REPLICATION_PORT_BASE = 9000
```

用于Greenplum数据库的某些附加产品和服务有额外的网络需求。下面的表格列出了在Greenplum集群内使用的端口和协议, 并且包括与Greenplum数据库集成的服务和应用。

Table 1. Greenplum数据库端口和协议

服务	协议/端口	描述
Master的SQL客户端连接	TCP 5432, libpq	Greenplum的Master主机上的SQL客户端连接端口。使用PostgreSQL的libpq API支持客户端。可配置。
Segment的SQL客户端连接	可变, libpq	Segment实例的SQL客户端连接端口。主机上的每一个主Segment和镜像Segment都必须有唯一的端口。端口在Greenplum系统初始化或者扩展时分配。gp_segment_configuration系统目录在port列中为每个Segment记录端口号。运行gpstate -p可以查看使用中的端口。
Segment镜像端口	可变, libpq	Segment从其主Segment接收镜像块的端口。该端口在镜像被设置时分配。端口号存储在gp_segment_configuration系统目录的mirror_port列中。
Greenplum数据库的Interconnect	UDP 1025-65535, 动态分配	在查询执行期间, Interconnect在Greenplum的Segment之间传输数据库元组。
后备Master的客户端监听器	TCP 5432, libpq	后备Master主机上的SQL客户端连接端口。通常和Master的客户端连接端口相同。可以用gpinitstandby工具的-p选项配置。
后备Master复制器	TCP 1025-65535, gpsyncmaster	Master主机上的gpsyncmaster进程会建立一个到第二Master主机的连接来把Master的日志复制到后备Master上。
Greenplum数据库文件装载和传输工具: gpfdist, gupload, gptransfer	TCP 8080, HTTP TCP 9000, HTTPS	gpfdist文件服务工具能够在Greenplum主机或者外部主机上运行。在启动该服务器时用-p选项指定连接端口。 gupload和gptransfer工具会用一个配置文件中指定的端口或者端口范围运行一个或者更多gpfdist。
Gpperfmon代理	TCP 8888	执行在每一台Greenplum主机上的gpperfmon代理(gpmon 和 gpmmon)的连接端口。通过Master和Segment主机上postgresql.conf中的配置变量gpperfmon_port设置。
备份完成通知	TCP 25, TCP 587, SMTP	gpbackup备份工具可以选择在备份完成时向一个email地址列表发送邮件。SMTP服务必须在Greenplum的Master主机上被启用。
Greenplum	TCP 22, SSH	Greenplum scp ssh

数据库的安全shell (SSH) : gpssh、gpscp、gpssh-exkeys、gppkg、gpseginstall		很多的工具使用和在主机之间传输文件并且管理集群中的Greenplum系统。
Greenplum平台扩展框架(PXF)	TCP 5888	PXF Java服务运行在每个Greenplum数据库segment主机上的5888端口。
Pgbouncer连接池	TCP, libpq	pgbouncer连接池运行在libpq客户端和Greenplum (或者PostgreSQL) 数据库之间。它可以运行在Greenplum的Master主机上，但推荐将它运行在Greenplum集群之外的一台主机上。当它运行在一台单独的主机上时，pgbouncer可以当作Greenplum的Master主机的温备机制，切换到Greenplum后备主机不要求重新配置客户端。在pgbouncer.ini配置文件中设置客户端连接端口和Greenplum的Master主机地址。

Parent topic: [安全性配置指南](#)

Greenplum数据库® 6.0文档

□ 安全性配置指南

保护数据库

Greenplum数据库端口和协议

配置客户端认证

配置数据库授权

审计

加密数据和数据库连接

安全性最佳实践

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

描述认证Greenplum数据库客户端的可用方法。

在Greenplum数据库系统第一次被初始化时，系统含有一个预定义的超级用户角色。这个角色将具有与初始化该Greenplum数据库系统的操作系统用户相同的名称。这个角色被称作gpadmin。默认情况下，系统被配置为只允许来自gpadmin角色的对数据库的本地连接。如果用户想要允许任

- [允许到Greenplum数据库的连接](#)
- [编辑pg_hba.conf文件](#)
- [认证方法](#)
- [限制并发连接](#)
- [加密客户端/服务器连接](#)

Parent topic: 安全性配置指南

允许到Greenplum数据库的连接

客户端访问和认证受到配置文件pg_hba.conf（标准的PostgreSQL基于主机认证文件）的控制。有关该文件的详细信息请见PostgreSQL文档中的[pg_hba.conf文件](#)

在Greenplum数据库中，Master实例的pg_hba.conf文件控制着对Greenplum系统的客户端访问和认证。Segment也有自己的pg_hba.conf文件，但是它们已经被正确地配置为仅允许来自Master主机的客户端连接。Segment从不接受外部的客户端连接，因此没有必要修改Segment上的pg_hba.conf文件。

pg_hba.conf文件的一般格式是一组记录，每个记录一行。空行会被忽略，任何#号注释字符之后的文本也同样会被忽略。记录由若干个被空格或者制表符分隔的域构成。如果域值被加上引号，其中可以包含空格。记录不能跨行。每个远程客户端访问记录都是如下格式：

host	database	role	address	authentication-method
------	----------	------	---------	-----------------------

每个UNIX域套接字访问记录是如下格式：

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

pg_hba.conf域的含义如下：

local

匹配使用UNIX域套接字的连接尝试。如果没有这类记录，就不允许UNIX域套接字连接。

host

匹配使用TCP/IP的连接尝试。除非服务器使用服务器配置参数`listen_addresses`的合适值启动，远程TCP/IP连接将不可能成功。

hostssl

匹配使用TCP/IP的连接尝试，但是只匹配使用SSL加密建立的连接。SSL必须在服务器启动时通过设置`ssl`配置参数来启用。对SSL认证的要求可以在`postgresql.conf`中配置。
[见为SSL认证配置postgresql.conf](#).

hostnossl

匹配使用TCP/IP但不用SSL的连接尝试。对SSL认证的要求可以在`postgresql.conf`中配置。
[见为SSL认证配置postgresql.conf](#).

database

指定这个记录匹配哪些数据库名。值`all`指定它匹配所有数据库。通过用逗号分隔数据库名，可以提供多个数据库名。可以指定一个包含数据库名的单独文件，方法是在这个域中给出加上前缀@的文件名。

role

指定这个记录匹配哪些数据库角色名。值`all`指定它匹配所有的角色。如果指定的角色是一个组并且想包括其中的所有成员，可在角色名前面放一个+。通过用逗号分隔角色名，可以提供多个角色名。可以指定一个包含角色名的单独文件，方法是在这个域中给出加上前缀@的文件名。

address

指定这个记录匹配的客户端机器地址。该字段可以包含IP地址，IP地址范围或主机名。

IP地址范围是使用标准点分指定范围的起始地址，然后是斜杠(/)和CIDR掩码长度。掩码长度表明客户端IP地址必须匹配的高位二进制位数。给定IP地址中在这一长度右边的二进制位必须为零。在IP地址、/、CIDR掩码长度之间不能有任何空格。

IPv4地址范围的典型例子有 172.20.143.89/32 是单台主机，或 172.20.143.0/24 是一个小型网络，或 10.6.0.0/16 是一个大型网络。对于单个主机，IPv6地址范围可能类似于::1/128 (在本例中为IPv6环回地址)

或fe80::7a31:c1ff:0000:0000/96用于小型网络。

0.0.0.0/0 代表所有IPv4地址，::0/0 代表所有IPv6地址。要指定单台主机，对于IPv4和IPv6分别使用CIDR掩码32和128.在网络地址中，不要省略拖尾的零。

以IPv4格式给出的条目将仅匹配IPv4连接，以IPv6格式给出的条目将仅匹配IPv6连接，即使所表示的地址在IPv4-in-IPv6范围

内。

Note: 如果主机系统C库不支持IPv6地址，则IPv6格式的条目将被拒绝。

如果指定了主机名（将非IP地址或IP范围的地址视为主机名），则将该名称与反向名称的结果进行比较客户端IP地址的分辨率（例如，如果使用DNS，则为反向DNS查找）。主机名比较不区分大小写。如果匹配，则转发名称对主机名执行解析（例如，正向DNS查找）以检查其解析为的地址是否等于客户端IP地址。如果两个方向都匹配，则认为该条目匹配。

某些主机名数据库允许将IP地址与多个主机名相关联，但是当要求系统解析IP时，操作系统仅返回一个主机名地址。`pg_hba.conf`中使用的主机名必须是客户端IP地址的地址到名称解析返回的主机名，否则行将不被视为匹配项。

在`pg_hba.conf` 中指定了主机名时，应确保名称解析相当快。设置本地名称解析缓存（例如`nscd`）可能是有利的。另外，用户可以启用服务器配置参数`log_hostname`以查看客户端主机名而不是日志中的IP地址。

IP-地址

IP-掩码

这些域可以被用作CIDR-地址记法的替换选择。与指定掩码长度不同，真实的掩码在一个单独的列中指定。例如，`255.0.0.0`表示一个IPv4的CIDR掩码长度8，而`255.255.255.255`表示CIDR掩码长度32。

authentication-method

指定连接时要使用的认证方法。详见 [认证方法](#)

CAUTION:

对于一个更安全的系统，考虑从Master的`pg_hba.conf`中移除所有使用trust认证的连接。trust认证意味着不做任何认证就授予角色访问，因此会绕过所有安全性。如果系统具有可用的ident服务，可以把trust项替换为ident认证。

编辑`pg_hba.conf`文件

最初，`pg_hba.conf`文件被设置成`gpadmin`用户对数据库具有完全访问权限，而其他Greenplum Database角色则没有数据库访问权限。用户需要编辑`pg_hba.conf`文件，以使用户能够访问数据库并保证`gpadmin`用户的安全。需要考虑删除具有信任身份验证的条目，因为它们允许有权访问服务器的任何人以他们选择的任何角色进行连接。对于本地（UNIX套接字）连接，可使用ident身份验证，这要求操作系统用户匹配指定的角色。对于本地和远程TCP连接，身份验证要求客户端的主机运行ident服务。用户可以在主控主机上安装ident服务，然后对本地TCP连接使用ident身份验证，例如`127.0.0.1/28`。将身份

验证用于远程TCP连接的安全性较差，因为它要求用户信任客户端主机上身份服务的完整性。

这个例子展示如何编辑Master的pg_hba.conf文件来允许对从所有角色访问所有数据库的远程客户端使用加密口令认证。

要编辑pg_hba.conf：

1. 在一个文本编辑器中打开文件\$MASTER_DATA_DIRECTORY/pg_hba.conf。
2. 为想要允许的每一类连接在文件中增加一行。记录会被顺序读取，因此记录的顺序是有意义的。通常，较早出现的记录将有比较严格的连接匹配参数以及较弱的认证方法，而较晚出现的记录将有较宽松的匹配参数和较强的认证方法。例如：

```
# 允许gpadmin用户使用ident认证本地访问所有数据库
local all gpadmin ident sameuser
host all gpadmin 127.0.0.1/32 ident
host all gpadmin ::1/128 ident
# 允许'dba'角色从任何具有IP地址192.168.x.x的主机访问任何数据库
# 并且是用md5加密的口令来认证用户
# 注意要使用SHA-256加密，用password替换下面行中的md5
host all dba 192.168.0.0/32 md5
```

认证方法

- [基本认证](#)
- [Kerberos认证](#)
- [LDAP认证](#)
- [SSL客户端认证](#)
- [基于PAM的认证](#)
- [Radius认证](#)

基本认证

支持下列基本认证方法：

Password 或者 MD5

要求客户端提供一个口令，可以为两者之一：

- Md5 – 口令作为MD5哈希传输。
- Password – 口令以明文传输。传输过程中应该总是使用SSL连

接来防止口令嗅探。这是可配置的，更多信息请见*Greenplum*数据库管理员指南中的“加密口令”。

Reject

拒绝有匹配参数的连接。用户通常应该使用这种方法来限制来自特定主机或者不安全连接的访问。

Ident

基于客户端的操作系统用户名的认证。用户只对本地连接使用这种方法。对来自远程主机的TCP连接使用ident要求客户端的主机正在运行ident服务。身份认证方法仅应与受信任的封闭网络上的远程主机一起使用。

下面是pg_hba.conf基本认证项的一些例子： entries:

```
hostnossl    all    all        0.0.0.0      reject
hostssl      all    testuser   0.0.0.0/0    md5
local        all    gpuser     ident
```

Kerberos认证

用户可以用一台Kerberos服务器 (RFC 2743, 1964) 认证。

pg_hba.conf 文件中Kerberos认证的格式是：

```
servicename/hostname@realm
```

可以把下列选项加到该项中：

Map

映射系统和数据库用户。

Include_realm

在ident映射文件中指定系统用户名中包括的realm名称的选项。

Krb_realm

为匹配的主体指定realm名称。

Krb_server_hostname

服务主体的主机名。

下面是一个pg_hba.conf的Kerberos项的例子：

```
host      all  all  0.0.0.0/0    krb5
                           hostssl all  all  0.0.0.0/0    krb5
map=krbmap
```

下列Kerberos服务器设置在postgresql.conf中被指定：

`krb_server_key_file`
设置Kerberos服务器密钥文件的位置。

`krb_srvname string`
Kerberos服务名。

`krb_caseins_users boolean`
大小写敏感性。默认为off。

下面的客户端设置作为连接参数被指定：

`Krb_srvname`
要用于认证的Kerberos服务名。

LDAP认证

用户可以用一个LDAP目录认证。

- LDAPS和TLS上的LDAP选项可以加密到LDAP服务器的连接。
- 除非SSL被启用，从客户端到服务器的连接不会被加密。配置客户端连接以使用SSL加密来自该客户端的连接。
- 要配置或者自定义LDAP设置，将LDAPCONF环境变量设置为到ldap.conf文件的路径并且将它加入到greenplum_path.sh脚本。

下面是为用户系统配置LDAP认证的推荐步骤：

1. 用要通过LDAP认证的数据库用户/角色设置LDAP服务器。
2. 在数据库上：
 - a. 验证要通过LDAP认证的数据库用户存在于数据库上。LDAP仅被用于验证用户名/口令对，因此角色应该存在于数据库中。
 - b. 更新\$MASTER_DATA_DIRECTORY中的pg_hba.conf文件为相应的用户使用LDAP作为认证方法。注意pg_hba.conf文件中第一个匹配用户/角色的项将被用作认证机制，因此项在该文件中的位置很重要。
 - c. 重载服务器让pg_hba.conf配置设置生效(gpstop -u)。

在auth-options中指定下面的参数。

`ldapserver`
要连接的LDAP服务器的名称或者IP地址。可以指定多个服务器，用空格分隔。

`ldapprefix`
在做简单绑定认证时，形成要绑定为的DN时自动加在用户名前面的字符串。

ldapsuffix

在做简单绑定认证时，形成要绑定为的DN时自动加在用户名后面的字符串。

ldapport

要连接的LDAP服务器上的端口号。如果没有指定端口，将使用LDAP库的默认端口设置。

ldaptls

设置为1让PostgreSQL和LDAP服务器之间的连接是用TLS加密。注意这只加密到LDAP服务器的流量 — 到客户端的连接仍将 是未加密的（除非使用了SSL）。

ldapbasedn

在做搜索+绑定认证时，要开始在其中搜索用户的根DN。

ldapbinddn

在做搜索+绑定认证时，绑定到目录以执行搜索的用户的DN。

ldapbindpasswd

在做搜索+绑定认证时，绑定到目录以执行搜索的用户的口令。

ldapsearchattribute

在做搜索+绑定认证时，匹配正在搜索的用户名的属性。

Example:

```
ldapserver=ldap.greenplum.com prefix="cn=" suffix="",
dc=greenplum, dc=com"
```

下面是LDAP认证的pg_hba.conf文件项的例子：

```
host all testuser 0.0.0.0/0 ldap ldap
ldapserver=ldapserver.greenplum.com ldapport=389
ldapprefix="cn="
ldapsuffix=",ou=people,dc=greenplum,dc=com"
hostssl all ldaprole 0.0.0.0/0 ldap
ldapserver=ldapserver.greenplum.com ldaptls=1
ldapprefix="cn="
ldapsuffix=",ou=people,dc=greenplum,dc=com"
```

SSL客户端认证

SSL认证将正在连接的客户端在SSL握手期间提供的SSL证书的公用名 (cn) 属性与被请求的数据库用户名进行比较。数据库用户应该在数据库中存在。映射文件可以被用来在系统用户名和数据库用户名之间进行映射。

SSL认证参数

认证方法:

- Cert

认证选项:

Hostssl

连接类型必须是hostssl。

map= mapping

映射。

这在pg_ident.conf文件中被指定，或者在ident_file服务器设置中指定的文件中指定。

下面是SSL客户端认证的pg_hba.conf项的例子:

```
Hostssl testdb certuser 192.168.0.0/16 cert
          Hostssl testdb all
192.168.0.0/16 cert map=gpuser
```

OpenSSL配置

Greenplum数据库默认会读取\$GP_HOME/etc/openssl.cnf中指定的OpenSSL配置文件。用户可以通过修改或者更新这个文件并且重启服务器对OpenSSL的默认配置做更改。

创建一个自签名证书

自签名证书可以被用来测试，但是生产中应该使用由证书颁发机构(CA)（全球CA或者本地CA）签发的证书，这样客户端就能够验证服务器的身份。如果所有的客户端都是组织本地的，推荐使用本地CA。

要为服务器创建一个自签名证书:

1. 输入下列openssl命令: command:

```
openssl req -new -text -out server.req
```

2. 在提示中输入要求的信息。

确保为公共名输入本地主机名。挑战口令可以留空。

3. 这个程序生成受口令保护的密钥，它不接受短于四字符的口令。要

移除口令（如果想让服务器自动启动还必须移除），运行下面的命令：

```
openssl rsa -in privkey.pem -out server.key rm  
privkey.pem
```

4. 输入旧口令解锁现有的密钥。然后输入下面的命令：

```
openssl req -x509 -in server.req -text -key server.key -  
out server.crt
```

这会把证书转变成一个自签名的证书并且把密钥和证书拷贝到服务器将查找它们的地方。

5. 最后，运行下列命令： command:

```
chmod og-rwx server.key
```

更多如何创建服务器私钥和证书的细节，请参考OpenSSL的文档。

为SSL认证配置postgresql.conf

下面的服务器设置需要在postgresql.conf配置文件中被指定：

- `ssl boolean`。 启用SSL连接。
- `ssl_renegotiation_limit integer`。 指定密钥重新协商之前的数据限制。
- `ssl_ciphers string`。 列出允许的SSL加密算法。

下列SSL服务器文件可以在Master的数据目录下找到：

- `server.crt`。 服务器证书。
- `server.key`。 服务器私钥。
- `root.crt`。 可信的证书机构。
- `root.crl`。 被证书机构撤销的证书。

配置SSL客户端连接

SSL选项：

`require`

仅使用SSL连接。如果存在一个根CA文件，以指定`verify-`

ca时同样的方式验证证书。

verify-ca

仅使用SSL连接。验证服务器证书由一个可信的CA发出。

verify-full

仅使用SSL连接。验证服务器证书由一个可信的CA发出并且该服务器主机名匹配证书中的名称。

sslcert

客户端SSL证书的文件名。默认是`~/.postgresql/postgresql.crt`。

sslkey

用于客户端证书的密钥。默认是`~/.postgresql/postgresql.key`。

sslrootcert

包含SSL证书机构证书的文件名。默认是`~/.postgresql/root.crt`。

sslcrl

SSL证书撤销列表的名称。默认是`~/.postgresql/root.crl`。

客户端连接参数可以使用下列环境变量设置：

- `sslmode` – PGSSLMODE
- `sslkey` – PGSSLKEY
- `sslrootcert` – PGSSLROOTCERT
- `sslcert` – PGSSLCERT
- `sslcrl` – PGSSLCRL

基于PAM的认证

“PAM”（可插拔认证模块）被用来验证用户名/口令对，类似于基本认证。PAM认证只在用户已经在数据库中存在时才有用。

参数

pamservice

默认的PAM服务是`postgresql`。注意如果PAM被设置为读取`/etc/shadow`，认证将会失败因为PostgreSQL服务器由一个非根用户启动。

下面是PAM客户端认证的`pg_hba.conf`项的例子：

```
local      all  gpuser  am  pamservice=postgresql
```

Radius认证

RADIUS（远程认证拨号用户服务）认证通过向一台配置好的RADIUS服务器发送'Authenticate Only'类型的Access Request消息工作。它包括用于用户名、口令（加密）以及网络访问服务器（NAS）标识符的参数。这个请求使用由radiussecret选项中指定的共享秘密加密。RADIUS服务器会响应Access Accept或者Access Reject。

Note: 不支持RADIUS计费。

只有用户已经在数据库中存在时，RADIUS认证才有用。

为了强密码，RADIUS加密向量要求SSL被启用。

RADIUS认证选项

radiusserver

RADIUS服务器的名称。

radiussecret

RADIUS共享秘密。

radiusport

RADIUS服务器上要连接的端口。

radiusidentifier

RADIUS请求中的NAS标识符。

下面是RADIUS客户端认证的pg_hba.conf项的例子：

```
hostssl  all  all  0.0.0.0/0  radius  radiusserver=servername
radiussecret=sharedsecret
```

限制并发连接

要限制对Greenplum数据库系统的活动并发会话的数量，用户可以配置max_connections服务器配置参数。这是一个本地参数，意味着用户必须在Master、后备Master和每个Segment实例（主Segment和镜像Segment）的postgresql.conf文件中设置

Segment	max connections
---------	-----------------

Master

5-

它。 上 的值必须是 上该值的 10倍。

在用户设置max_connections时，还必须设置依赖参数max_prepared_transactions。在Master上，这个值必须被设置为至少和max_connections值一样大，而在Segment实例上应该设置为和Master上一样的值。

在\$MASTER_DATA_DIRECTORY/postgresql.conf (包括后备Master) 中：

```
max_connections=100
max_prepared_transactions=100
```

在所有Segment实例的SEGMENT_DATA_DIRECTORY/postgresql.conf中：

```
max_connections=500
max_prepared_transactions=100
```

Note: 注意：调高这些参数的值可能会导致Greenplum数据库请求更多共享内存。为了缓和这种影响，考虑降低其他内存相关的参数，例如gp_cached_segworkers_threshold。

要更改允许的连接数：

1. 停止Greenplum数据库系统：

```
$ gpstop
```

2. 在Master主机上，编

辑\$MASTER_DATA_DIRECTORY/postgresql.conf并且更改下面两个参数：

- max_connections – 想要允许的活动用户会话数加上superuser_reserved_connections的数量
- max_prepared_transactions – 必须大于等于max_connections.

3. 在每个Segment实例上，编

辑SEGMENT_DATA_DIRECTORY/postgresql.conf并且更改下面两个参数：

- max_connections – 必须是Master上值的5-10倍。
- max_prepared_transactions – 必须等于Master上的值。

4. 重启Greenplum数据库系统：

```
$ gpstart
```

加密客户端/服务器连接

Greenplum数据库对于客户端和Master服务器之间的SSL连接有原生支持。SSL连接可以防止第三方进行包嗅探，还能防止中间人攻击。只要客户端连接会通过不安全的链接，就应该使用SSL，并且只要使用客户端证书认证也应该使用SSL。

Note: 有关在gpfdist服务器和Greenplum数据库Segment主机之间加密数据的信息，请见[加密gpfdist连接](#).

启用SSL要求在客户端和Master服务器系统上都安装OpenSSL。可通过在Master的postgresql.conf中设置服务器配置参数ssl=on让Greenplum以启用SSL的方式启动。在以SSL模式启动时，服务器将在Master的数据目录中查找server.key（服务器私钥）文件和server.crt（服务器证书）文件。在启用SSL的Greenplum系统能启动之前，这些文件必须被正确地设置好。

Important: 不要用口令保护私钥。服务器不会为私钥提示要求口令，而不提供口令数据库启动会失败并且报出错误。

自签名证书可以被用来测试，但是生产中应该使用由证书颁发机构(CA)（全球CA或者本地CA）签发的证书，这样客户端就能够验证服务器的身份。如果所有的客户端都是组织本地的，推荐使用本地CA。创建自签名证书的步骤请见[创建一个自签名证书](#)

Greenplum数据库® 6.0文档

□ 安全性配置指南

保护数据库

Greenplum数据库端口和协议

配置客户端认证

配置数据库授权

审计

加密数据和数据库连接

安全性最佳实践

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

[如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息](#)

描述如何通过使用角色和权限在用户级别限制对数据库数据的授权访问。

Parent topic: [安全性配置指南](#)

访问权限和角色

Greenplum数据库使用角色管理数据库访问权限。角色的概念包括了用户和组的概念。一个角色可以是一个数据库用户、组或者两者皆有。角色可以拥有数据库对象（例如表）并且可以把那些对象上的特权指派给其他角色以控制对那些对象的访问。角色可以是其他角色的成员，因此成员角色可以继承其父角色的对象特权。

每一个Greenplum数据库系统都包含一组数据库角色（用户和组）。那些角色独立于服务器所运行的操作系统管理的用户和组。不过，为了方便用户可能想要维护操作系统用户名和Greenplum数据库角色名之间的关系，因为很多客户端应用使用当前的操作系统用户名作为默认的数据库用户名。

在Greenplum数据库中，用户通过Master实例登入和连接，Master实例会验证它们的角色和访问特权。然后Master将在幕后用当前登入的角色向Segment实例发出命令。

角色被定义在系统层面上，因此它们对系统中的所有数据库都有效。

要让Greenplum数据库系统自举，刚刚初始化好的系统总是包含一个预定义的超级用户角色（也被称作系统该用户）。这个角色将和初始化Greenplum数据库系统的操作系统用户具有相同的名称。习惯上，这个角色被命名为gpadmin。要创建更多角色，用户首先必须作为这个初始角色连接。

管理对象特权

当一个对象（表、视图、序列、数据库、函数、语言、方案或表空间）被创建时，它会被指派一个拥有者。拥有者通常就是执行创建语句的角色。对于大部分种类的对象，初始状态只有拥有者（或者超级用户）可以对该对象做任何事情。要允许其他角色使用对象，必须授予特权。对每一类对象，Greenplum数据库支持下面的特权：

对象类型	特权
表、视图、序列	<ul style="list-style-type: none"> • SELECT • INSERT • UPDATE • DELETE • RULE • ALL
外部表	<ul style="list-style-type: none"> • SELECT • RULE • ALL
数据库	<ul style="list-style-type: none"> • CONNECT • CREATE • TEMPORARY TEMP • ALL
函数	EXECUTE
Procedural Languages	USAGE
方案	<ul style="list-style-type: none"> • CREATE • USAGE • ALL

特权必须为每个对象单独授予。例如，在一个数据库上授予ALL并不会为该数据库中的对象授予完全的访问。它只授予所有的数据库级别特权（CONNECT, CREATE, TEMPORARY）给数据库本身。

使用SQL命令GRANT把对象上的特权给一个特定角色。例如：

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

T要收回特权，可使用REVOKE命令。例如：

```
=# REVOKE ALL PRIVILEGES ON mytable FROM jsmith;
```

用户还可以使用DROP OWNED以及REASSIGN OWNED命令来管理弃用角色所拥有的对象（注意：只有对象的拥有者或者超级用户可以删除一个对象或者重新指派拥有关系）。例如：

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

使用SHA-256加密

Greenplum数据库的访问控制大概能对应桔皮书的“C2”级安全性，而不是“B1”级。Greenplum数据库当前支持对象级别的访问特权。行级或者列级访问不被支持，标记安全性也不被支持。

行级和列级访问可以使用限制被选择的行列的视图来模拟。行级标签可以通过为表增加存储敏感度信息的额外列模拟，然后使用视图来控制基于该列的行级访问。然后可以为角色授予对视图的访问而不是对基表的访问。虽然这些变通方案不能提供和“B1”级安全性相同的安全性，但对于很多组织来说它们仍然是一种可行的替代方案。

要使用SHA-256加密，用户必须在系统或者会话级别上设置一个参数。这一节介绍如何使用一个服务器参数来实现SHA-256加密的口令存储。注意为了使用SHA-256加密进行存储，客户端认证方法必须被设置为password而不是默认的MD5（详见[配置SSL客户端连接](#)）。这意味着口令在网络上以明文方式传输，因此我们高度推荐用户设置SSL来加密客户端和服务器之间的通信信道。

用户可以在系统范围或者以会话为基础设置选择的加密方法。可用的加密方法是SHA-256以及MD5（为了向后兼容性）。

在系统范围设置加密方法

要在整个Greenplum系统（Master及其Segment）上设置password_hash_algorithm服务器参数：

1. 作为超级用户登入Greenplum数据库实例。
2. 执行gpconfig把password_hash_algorithm设置为SHA-256：

```
$ gpconfig -c password_hash_algorithm -v 'SHA-256'
```

3. 验证设置：

```
$ gpconfig -s
```

将看到：

```
Master value: SHA-256
Segment value: SHA-256
```

为单个会话设置加密方法

要为单个会话设置password_hash_algorithm服务器参数：

1. 作为超级用户登入Greenplum数据库实例。
2. 设置password_hash_algorithm为SHA-256：

```
# set password_hash_algorithm = 'SHA-256'
```

3. 验证设置：

```
# show password_hash_algorithm;
```

将会看到：

```
SHA-256
```

下面是展示新设置效果的例子：

1. 作为超级用户登入并且验证口令哈希算法设置：

```
# show password_hash_algorithm
password_hash_algorithm
-----
SHA-256
```

2. 创建一个带有口令且有登录特权的新角色。

```
create role testdb with password 'testdb12345#' LOGIN;
```

3. 更改客户端认证方法以允许SHA-256加密口令的存储：
在Master上打开pg_hba.conf文件并且加入下面的行：

```
host all testdb 0.0.0.0/0 password
```

4. 重启集群。
5. 作为刚创建好的用户testdb登入数据库。

```
psql -U testdb
```

6. 在提示下输入正确的口令。
7. 验证口令被存储为SHA-256哈希。
口令的哈希被存储在pg_authid.rolpassword中。
8. 作为超级用户登入。
9. 执行下列查询：

```
# SELECT rolpassword FROM pg_authid WHERE rolname = 'testdb';
   Rolpassword
  -----
sha256<64 hexadecimal characters>
```

用时间限制访问

Greenplum数据库让管理员能够限制角色在特定时间的访问。可使用CREATE ROLE或者ALTER ROLE命令指定基于时间的约束。

可以用日期或者日期和时间限制访问。无需删除和重建角色就可以移除这些约束。

基于时间的约束只适用于它们指派给的角色。如果角色是另一个有时间约束的角色的成员，时间约束不会被继承。

基于时间的约束仅在登录时被实施。SET ROLE以及SET SESSION AUTHORIZATION命令不受任何基于时间的约束的影响。

要为角色设置基于时间的约束，要求超级用户或者CREATEROLE特权。没有人可以为超级用户增加基于时间的约束。

有两种方法增加基于时间的约束。在CREATE ROLE或者ALTER ROLE命令中使用关键词DENY，后面接上下面的一种形式：。

- 访问被限制的一个日子，以及可选的时间。例如，在周三不能访问。
- 一个区间——也就是一个开始日期和结束日期以及可选的时

间——在其间访问被限制。例如，从周三下午10点到周四上午8点期间不能访问。

用户可以指定多个限制，例如，周三的任何时间都不能访问并且在周五的下午3点到5点之间不能访问。

有两种方法指定一个日子。使用后面跟着带单引号的英语中平日术语的DAY或者0到6之间的一个数字，如下表所示。

英语术语	数字
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

日子中的时间可以以12小时制或者24小时制指定。在词TIME后面接上带单引号的说明。只能指定小时和分钟并且用分号分隔 (:)。如果使用12小时制，在最后加上AM或者PM。下面的例子展示了多种时间说明。

```
TIME '14:00'      # 24-hour time implied
TIME '02:00 PM'   # 12-hour time specified by PM
TIME '02:00'       # 24-hour time implied. This is equivalent
to TIME '02:00 AM'.
```

Important: 基于时间的约束根据服务器时间实施。不考虑时区。

要指定一个时间区间，在其间访问被禁止，可以用词BETWEEN和AND指定两个日子/时间说明。如下所示。DAY总是需要的。

```
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'
      DAY 2 TIME '02:00'
```

最后三个语句等效。

Note: 区间不能在周六之后回卷。

下面的语法不正确：

```
DENY BETWEEN DAY 'Saturday' AND DAY 'Sunday'
```

正确的说明是使用两个DENY子句，如下所示：

```
DENY DAY 'Saturday'  
DENY DAY 'Sunday'
```

下面的例子展示了创建带有基于时间约束的角色并且修改角色以增加基于时间的约束。只有基于时间约束所需的语句被显示。更多有关创建和修改角色的细节请见 *Greenplum*数据库参考指南中CREATE ROLE 和 ALTER ROLE 的描述。

例 1 – 创建带有基于时间约束的新角色

在周末不允许访问。

```
CREATE ROLE generaluser  
DENY DAY 'Saturday'  
DENY DAY 'Sunday'  
...
```

例 2 – 修改角色以增加基于时间的约束

每晚的凌晨2点到4点之间不允许访问。

```
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 – 修改角色以增加基于时间的约束

在周三或者周五下午的3点到5点间不允许访问。

```
ALTER ROLE generaluser  
DENY DAY 'Wednesday'  
DENY BETWEEN DAY 'Friday' TIME '15:00' AND DAY 'Friday'  
TIME '17:00'
```

删除基于时间的约束

要移除基于时间的约束，请使用ALTER ROLE命令。输入后面跟着要删除的日子/时间说明的关键词DROP DENY FOR。

```
DROP DENY FOR DAY 'Sunday'
```

任何含有DROP子句中全部或者部分条件的约束都会被移除。例如，如果一条现有的约束否定周一和周二的访问，并且DROP子句移除周一的约束，则这一条现有的约束会被完全删除。DROP子句会完全移除所有与DROP子句中约束交叠的约束。即便发生交叠的约束包含有比DROP子句更多的限制，它们也会被完全删除。

例 1 - 从一个角色移除一条基于时间的限制

```
ALTER ROLE generaluser  
DROP DENY FOR DAY 'Monday'  
...
```

这个语句将为例2中的generaluser角色移除所有与周一约束交叠的约束，即便其中有额外的约束。

Greenplum数据库® 6.0文档

□ 安全性配置指南

保护数据库

Greenplum数据库端口和协议

配置客户端认证

配置数据库授权

审计

加密数据和数据库连接

安全性最佳实践

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮
提交详细信息

描述被记录并且应该被监控以检测安全性威胁的Greenplum数据库事件。

Greenplum数据库有能力审计很多事件，包括系统的启动和关闭、Segment数据库失效、导致错误的SQL语句以及所有的连接尝试和断开连接。Greenplum数据库还记录SQL语句和关于SQL语句的信息，并且可以以多种方式配置来记录不同细节的审计信息。`log_error_verbosity`配置参数控制写入到服务器日志中的每一条消息的细节多少。类似地，`log_min_error_statement`参数允许管理员配置为SQL语句记录的细节层次，而`log_statement`参数确定要被审计的SQL语句的种类。当可审计事件由Greenplum数据库外部的主体发起时，Greenplum数据库会为所有可审计事件记录用户名。

Greenplum数据库通过只允许具有适当角色的管理员在日志文件上执行操作来防止对审计记录的未授权修改和删除。日志以一种使用逗号分隔值（CSV）的专有格式存储。每个Segment和Master都存储有自己的日志文件，不过这些都可以由管理员从远程访问。Greenplum数据库还通过`log_truncate_on_rotation`参数授权对旧日志文件的覆盖。这是一个本地参数并且必须在每个Segment和Master的配置文件中设置。

Greenplum提供了一个名为`gp_toolkit`的管理方案，用户可以用它来查询日志文件、系统目录和操作环境得到系统状态信息。包括用法在内的更多信息，请参考Greenplum数据库参考指南中的`gp_toolkit`管理方案附录。

查看数据库服务器的日志文件

Greenplum数据库中的每一个数据库实例（Master和Segment）都是一个运行着的PostgreSQL数据库服务器，它们都有自己的服务器日志文件。每天的日志文件被创建在Master和每个Segment的数据目录下的`pg_log`目录中。

服务器日志文件被写为逗号分隔值（CSV）格式。不是所有的日志项在所有的日志域中都有值。例如，只有与查询工作者进程相关的日志项才会有`slice_id`值。一个特定查询的相关日志项可以通过其会话标识符（`gp_session_id`）和命令标识符（`gp_command_count`）确定。

#	域名	数据类型	描述
1	event_time	timestamp with time zone	日志项被写到日志中的时间
2	user_name	varchar(100)	数据库用户名
3	database_name	varchar(100)	数据库名
4	process_id	varchar(10)	系统进程ID（带前缀“P”）
5	thread_id	varchar(50)	线程计数（带前缀“TH”）
6	remote_host	varchar(100)	在Master上，是客户端机器的主机名/地址。在Segment上，是Master的主机名/地址。
7	remote_port	varchar(10)	Segment或Master的端口号
8	session_start_time	timestamp with time zone	会话连接打开的时间
9	transaction_id	int	Master上的顶层事务ID。这个ID是任何子事务的父级。

10	gp_session_id	text	会话标识符号（带前缀"con"）
11	gp_command_count	text	会话内部的命令编号（带前缀"cmd"）
12	gp_segment	text	Segment内容标识符（对主Segment带前缀"seg"，镜像Segment带前缀"mir"）。Master的内容id总是-1。
13	slice_id	text	切片id（查询计划被执行的部分）
14	distr_trnx_id	text	分布式事务ID
15	local_trnx_id	text	本地事务ID
16	sub_trnx_id	text	子事务ID
17	event_severity	varchar(10)	值包括：LOG、ERROR、FATAL、PANIC、DEBUG1、DEBUG2
18	sql_state_code	varchar(10)	与日志消息相关的SQL状态代码
19	event_message	text	日志或者错误消息文本
20	event_detail	text	与错误或者警告消息相关的详细消息文本
21	event_hint	text	与错误或者警告消息相关的提示消息文本
22	internal_query	text	内部产生的查询文本
23	internal_query_pos	int	指向内部产生的查询文本中的光标
24	event_context	text	产生消息的上下文
25	debug_query_string	text	带有完整细节的用户提供的查询字符串，用于调试。这个字符串可能会由于内部使用而修改。
26	error_cursor_pos	int	指向查询字符串中的光标
27	func_name	text	产生这个消息的函数
28	file_name	text	产生消息的内部代码文件
29	file_line	int	产生消息的内部代码文件的行号
30	stack_trace	text	与这个消息相关的栈跟踪文本

Greenplum提供一个名为`gplogfilter`的工具，它能被用来在一个Greenplum数据库日志文件中搜索匹配指定条件的项。这个工具默认会搜索位于默认日志位置的Greenplum的Master日志文件。例如，要显示Master日志文件的最后三行：

```
$ gplogfilter -n 3
```

用户还可以通过gpssh工具运行gplogfilter来立刻搜索所有的Segment日志文件。例如，要显示每个Segment日志文件的最后三行：

```
$ gpssh -f seg_host_file
=> source /usr/local/greenplum-db/greenplum_path.sh
=> gplogfilter -n 3 /gpdata/gp*/pg_log/gpdb*.csv
```

下面是Greenplum与安全相关的审计（或者日志）服务器配置参数，它们可以在postgresql.conf配置文件中设置：

域名	值范围	默认	描述
log_connections	Boolean	off	这会对服务器日志输出一行详细描述每个成功的连接。某些客户端程序（如psql）在决定是否要求口令时会尝试连接两次，因此重复的“connection received”消息并非总是表示问题。
log_disconnections	Boolean	off	在一个客户端会话终止时，这会在服务器日志中输出一行，其中会包括该会话的持续时间。
log_statement	NONE DDL MOD ALL	ALL	控制那些SQL语句会被记录。DDL记录所有数据定义命令，如CREATE、ALTER和DROP命令。MOD记录所有DDL语句外加INSERT、UPDATE、DELETE、TRUNCATE以及COPY FROM。如果PREPARE和EXPLAIN ANALYZE语句中如果包含有适当类型的命令，它们也会被日志记录。
log_hostname	Boolean	off	连接日志消息默认只显示连接主机的IP地址。把这个选项打开会导致主机名也被记录。注意这依赖于用户的主机名解析设置，而且这有可能会带来不可忽视的性能损失。
log_duration	Boolean	off	致使每一个满足log_statement的完成语句的持续时间被记录。
log_error_verbosity	TERSE DEFAULT VERBOSE	DEFAULT	为被记录的每条消息控制写入到服务器日志的细节多少。
log_min_duration_statement	number of milliseconds, 0, -1	-1	如果语句的持续时间大于等于指定的毫秒数，则在一个日志行中记录该语句和它的持续时间。将这个参数设置为0将打印出所有的语句及其持续时间。-1禁用这一特性。例如，如果用户将它设置为250，那么所有运行时间大于等于250ms的SQL语句将被记录。在跟踪应用中的未优化查询时，启用这一选项非常有用。
log_min_messages	DEBUG5 DEBUG4 DEBUG3 DEBUG2 DEBUG1	NOTICE	控制哪些消息级别会被写入到服务器日志。每个级别包括其后的所有级别。级别越靠后，发送到日志的消息就越少。

	INFO NOTICE WARNING ERROR LOG FATAL PANIC		
log_rotation_size	0 - INT_MAX/1024 kilobytes	1048576	大于0时，将这个千字节数写入日志后，将创建一个新的日志文件。设置为零可禁用基于大小的新日志文件的创建。
log_rotation_age	Any valid time expression (number and unit)	1d	决定个体日志文件的最大生存时间。在这个时间过去之后，一个新的日志文件将被创建。设置为零可禁用新日志文件基于时间创建。
log_statement_stats	Boolean	off	对每个查询，写入查询解析器、规划器和执行器的整体性能统计信息到服务器日志中。这是一种粗糙的画像手段。
log_truncate_on_rotation	Boolean	off	截断（重写）而不是追加到任何现存的同名日志文件。仅当一个新文件由于基于时间的轮转而被打开时，截断才会发生。例如，使用这个设置配合gpseg#-%H.log这样的log_filename会导致产生24个每小时的日志文件，然后循环地重写它们。关闭这一设置时，预先已经存在的文件在所有的情况下都会被追加内容。

Parent topic: [安全性配置指南](#)

Greenplum数据库® 6.0文档

□ 安全性配置指南

保护数据库

Greenplum数据库端口和协议

配置客户端认证

配置数据库授权

审计

加密数据和数据库连接

安全性最佳实践

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

描述如何在数据库中或者在网络上传输时加密数据以防止窃听攻击或者中间人攻击。

- 客户端和Master数据库之间的连接可以用SSL加密。这可以用ssl服务器配置参数启用，该参数默认为off。将ssl参数设置为on允许客户端与Master的通信被加密。Master数据库必须为SSL设置好。更多有关用SSL加密客户端连接的信息请见[OpenSSL配置](#) OpenSSL配置。
- Greenplum数据库允许在Greenplum并行文件分发服务器、gpf dist以及Segment主机之间传输的数据的SSL加密。更多信息请见[加密gpf dist连接](#) 加密gpf dist连接。
- pgcrypto包中的加密/解密函数可以保护数据库中静止的数据。级别的加密可以保护敏感信息，例如社会保险号码或信用卡号。更多信息请见[用pgcrypto加密静止数据](#) 用pgcrypto加密静止数据。

Parent topic: [安全性配置指南](#)

加密gpf dist连接

gpf dists协议是gpf dists协议的一个安全版本，它安全地标识文件服务器和Greenplum数据库并且加密它们之间的通信。使用gpf dists可以防止窃听攻击和中间人攻击。

gpf dists协议用下列显著的特性实现了客户端/服务器的SSL安全性：

- 要求客户端证书。
- 不支持多语言证书。
- 不支持证书撤销列表（CRL）。
- TLSv1协议被用于TLS_RSA_WITH_AES_128_CBC_SHA加密算法。这些SSL参数不能更改。
- 支持SSL重新协商。
- SSL的忽略主机失配参数被设置为假。
- gpf dist文件服务器（server.key）和Greenplum数据库（client.key）不支持含有口令的私钥。
- 发出适合于使用中的操作系统的证书是用户的责任。通常，支持将证书转换成必要的格式，例如使用<https://www.sslshopper.com/ssl-tools.html>



[converter.html](#) 上的SSL转换器。

用--ssl选项启动的gpfdist服务器只能用gpfdist协议通信。没有用--ss1选项启动的gpfdists服务器只能用gpfdists协议通信。更多gpfdists的细节请参考*Greenplum数据库管理员指南*。

有两种方法启用gpfdists协议：

- 用--ss1选项运行gpfdist然后在CREATE EXTERNAL TABLE语句的LOCATION子句中使用gpfdists协议。
- 使用一个SSL选项被设置为真的YAML控制文件并且运行gpload。运行gpload会用--ss1选项启动gpfdist服务器然后使用gpfdist协议。

在使用gpfdists时，下列客户端证书必须位于每一个Segment的\$PGDATA/gpfdists目录中：

- 客户端证书文件client.crt
- 客户端私钥文件client.key
- 可信证书机构root.crt

Important: 不要用口令保护私钥。服务器不会为私钥提示要求口令，因此数据装载会在要求口令的情况下失败并且报出错误。

在使用带有SSL的gpload时，用户在YAML控制文件中指定服务器证书的位置。在使用带有SSL的gpfdist时，用户用--ss1选项指定服务器证书的位置。

下面的例子展示如何安全地把数据装载到一个外部表中。这个例子从所有扩展名为txt的文件使用gpfdists协议创建了一个名为ext_expenses的可读外部表。这些文件被格式化为使用竖线(|)作为列定界符并且使用空格作为空值。

1. 在Segment主机上用--ss1选项运行gpfdist。
2. 登入数据库并执行下面的命令：

```
=# 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 '' ) ;
```

用pgcrypto加密静止数据

Greenplum数据库的pgcrypto包提供加密静止在数据库中数据的函数。管理员可以加密有敏感信息的列以提供额外的保护层，例如社会保险号码或者信用卡号。存储为加密形式的数据库数据不能被不掌握加密密钥的用户读取，并且数据也无法直接从磁盘上读出。

当用户安装Greenplum数据库时，默认情况下会安装pgcrypto。用户必须在要使用该模块的每个数据库中显式启用pgcrypto。

pgcrypto允许使用对称和非对称加密的PGP加密。对称加密使用相同的密钥加密和解密数据，并且比非对称加密更快。在密钥交换不成为问题的环境中，对称加密是首选方法。对于非对称加密，公钥被用来加密数据而私钥被用来解密数据。这种方法比对称加密要慢一些并且要求更强的密钥。

使用pgcrypto总是会带来性能和可维护性方面的代价。有必要只对需要的数据使用加密。还有，记住无法通过索引来搜索加密数据。

在实现数据库内加密之前，考虑下列PGP限制。

- 不支持签名。这也意味着检查加密子密钥是否属于主密钥。
- 不支持加密密钥作为主密钥。通常不鼓励这种实践，因为这类限制不应该成为问题。
- 不支持多个子密钥。这可能看起来像是一个问题，因为这是常见的做法。在另一方面，用户不应将自己的常规GPG/PGP密钥用于pgcrypto，而是创建新的密钥，因为使用场景很不相同。

Greenplum数据库默认编译了zlib，这允许PGP加密函数在加密前压缩数据。在编译有OpenSSL时，将有更多算法可用。

因为pgcrypto函数在数据库服务器内部运行，数据和口令在pgcrypto和客户端应用之间以明文形式移动。为了最好的安全性，用户应该使用本地连接或者SSL连接并且用户应该信任系统管理员和数据库管理员。

pgcrypto会根据主PostgreSQL配置脚本的发现配置其自身。

当编译有zlib时，pgcrypto加密函数能够在加密前压缩数据。

pgcrypto有多种加密级别，范围从基本加密到高级内建函数。下面的表格显示所支持的加密算法。

Table 1. 表 1. pgcrypto支持的加密函数

值功能	内建	带有OpenSSL
MD5	yes	yes
SHA1	yes	yes

SHA224/256/384/512	yes	yes ¹
其他摘要算法	no	yes ²
Blowfish	yes	yes
AES	yes	yes ³
DES/3DES/CAST5	no	yes
Raw Encryption	yes	yes
PGP Symmetric-Key	yes	yes
PGP Public Key	yes	yes

创建PGP密钥

要在Greenplum数据库中使用PGP非对称加密，用户必须首先创建公私钥并且安装它们。

这一节假定用户正在一台Linux机器上用GNU隐私保护（gpg）命令行工具安装Greenplum数据库。使用最新版本的GPG来创建密钥。从<https://www.gnupg.org/download/>为用户的操作系统下载和安装GNU隐私保护（GPG）。在GnuPG网站用户将找到流行的Linux发布的安装器以及Windows和Mac OS X安装器的链接。

1. 作为root，执行下面的命令并且从菜单中选择选项1：

```
# gpg --gen-key
gpg (GnuPG) 2.0.14; Copyright (C) 2009 Free Software
Foundation, Inc.
This is free software: you are free to change and
redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: directory '/root/.gnupg' created
gpg: new configuration file '/root/.gnupg/gpg.conf'
created
gpg: WARNING: options in '/root/.gnupg/gpg.conf' are not
yet active during this run
gpg: keyring '/root/.gnupg/secring.gpg' created
gpg: keyring '/root/.gnupg/pubring.gpg' created
Please select what kind of key you want:
(1) RSA and RSA (default)
(2) DSA and Elgamal
(3) DSA (sign only)
(4) RSA (sign only)
Your selection? 1
```

2. 如这个例子中所示，对提示做出响应并且遵照指示：

```

RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) Press enter to accept
default key size
Requested keysize is 2048 bits
Please specify how long the key should be valid.
  0 = key does not expire
  <n> = key expires in n days
  <n>w = key expires in n weeks
  <n>m = key expires in n months
  <n>y = key expires in n years
  Key is valid for? (0) 365
Key expires at Wed 13 Jan 2016 10:35:39 AM PST
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: John Doe
Email address: jdoe@email.com
Comment:
You selected this USER-ID:
  "John Doe <jdoe@email.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
You need a Passphrase to protect your secret key.
(For this demo the passphrase is blank.)
can't connect to '/root/.gnupg/S.gpg-agent': No such
file or directory
You don't want a passphrase - this is probably a *bad*
idea!
I will do it anyway. You can change your passphrase at
any time,
using this program with the option "--edit-key".

We need to generate a lot of random bytes. It is a good
idea to perform
some other action (type on the keyboard, move the mouse,
utilize the
disks) during the prime generation; this gives the
random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good
idea to perform
some other action (type on the keyboard, move the mouse,
utilize the
disks) during the prime generation; this gives the
random number
generator a better chance to gain enough entropy.
gpg: /root/.gnupg/trustdb.gpg: trustdb created
gpg: key 2027CC30 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdbgpg:
      3 marginal(s) needed, 1 complete(s) needed, PGP
trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q,
0n, 0m, 0f, 1u

```

```

gpg: next trustdb check due at 2016-01-13
pub 2048R/2027CC30 2015-01-13 [expires: 2016-01-13]
      Key fingerprint = 7EDA 6AD0 F5E0 400F 4D45 3259
      077D 725E 2027 CC30
uid                  John Doe <jdoe@email.com>
sub 2048R/4FD2EFBB 2015-01-13 [expires: 2016-01-13]

```

3. 通过输入下面的命令列出PGP密钥：

```

gpg --list-secret-keys
/root/.gnupg/secring.gpg
-----
sec 2048R/2027CC30 2015-01-13 [expires: 2016-01-13]
uid          John Doe <jdoe@email.com>
ssb 2048R/4FD2EFBB 2015-01-13

```

2027CC30是公钥并且将被用于加密数据库中的数据。4FD2EFBB是私（秘密）钥并且将被用来解密数据。

4. 使用下面的命令导出密钥：

```

# gpg -a --export 4FD2EFBB > public.key
# gpg -a --export-secret-keys 2027CC30 > secret.key

```

更多有关PGP加密函数的信息，请见[pgcrypto](#) 的文档。

使用PGP加密表中的数据

这一节显示如何使用刚生成的PGP密钥加密插入到列中的数据。

1. 转储public.key文件的内容然后把它拷贝到剪切板：

```

# cat public.key
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.14 (GNU/Linux)

mQENBFS1zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jt
2HislojSP6LI0cSkIqMU9LAlnceczhRIhBhuVgK1GSgd9texg2nnSL9Adm
R5syVKG+qcdWuvyZg9oO0meyjhc3n+kkbRTEMuM3f1bMs8shOwzMvstCUV
vG5rJAE8PuYDSJCJ74I6w7SOH3RiRIC7IfL6xYddV4213ctd44b18/i71h
/Hbsji2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFNFnE6
ic8MD01K5/GA1R6Hc0ZIHf8KEcavruQlikjnABEBAAG0HHR1c3Qga2V5ID
a2V5QGVtYWlsLmNvbT6JAT4EEwECACgFAls1zf0CGwMFCQHhM4AGCwkIBw

```

```

AgkKCwQWAgMBAh4BAheAAAoJEAd9c14gJ8wwbfwH/3VyVsPkQl1owRJNxv
7BfrvU52yk+PPZYoes9UpdL3CMRk8gAM9bx5Sk08q2UXSZLC6ffOpEW4uW
JR0C3ooezTkmCBW8I1bU0qGetzVxopdXLuPGCE7hVWQe9HcSntiTLxGov1
TAoccXLbyuZh9Rf5vLoQdKzcCyOHh5IqXaQOT100TeFeEpb9TIiwcntg3W
DGoUAOanjDZ3KE8Qp7V74fhG1EZVzHb8FajR62CXSHFKqpBgiNxnTOk45N
eTUXPSnwPi46qoAp9UQogsfGyB1XDOTB2UoqhutAMECaM7VtpePv79i0Z/
AQ0EVLV1/QEIAbFdQ+8QMCADOipM1bF/JrQt3zUoc4BTqICaxdyzAfz0
Zro2us99G1ARqLwd8EqJcl/xmfcJiZyUam6ZAzzFXCgnH5Y1sdtMTJZdLp
gCWG/ZLu4wzxOFFzDkiPv9RDw6e5MNLTJrSp4hS5o2apKdbO4Ex83O4mJY
iDDCWU4T01hv3hSKCpke6LcwsX+7liozp+aNmP0Ypwfi4hR3UUMP70+v1b
bVLz31LLouHRgpCzla+PzzbEKs16jq77vG9kqZTCIzXoWaLljuitRlfJko
v/8yAnkcAmowZrIBlyFg2KBzhunYmN2YvkUAEQEAAYkBjQQYAQIADwUCVL
DAUJAeEzgAAKCRAHfxJeICfMMOHYCACFhInZA9uAM3TC441+MrgMUJ3rW9
WrdTsxR8WkSNbIxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRrI61FPHQN
WH+N21asoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8Hkk
HMUC55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNLGWE
/UUZB+dYqCwtvX0nnBu1KNcmk2AkEcFK3YoliCxomdOxhFOv9AKjjojDyC
Pv2MikPS2fKOAg1R3LpMa8zDET14w3vckPQNrQNnYuUtfj6ZoCxx
=XZ8J
-----END PGP PUBLIC KEY BLOCK-----

```

- 创建一个名为userssn的表并且插入一些敏感数据，这个例子中是Bob和Alice的社会保险号码。在"dearmor("之后粘贴public.key的内容。

```

CREATE TABLE userssn( ssn_id SERIAL PRIMARY KEY,
                      username varchar(100), ssn bytea);

INSERT INTO userssn(username, ssn)
SELECT robotccs.username, pgp_pub_encrypt(robotccs.ssn,
                                             keys.pubkey) AS ssn
FROM (
    VALUES ('Alice', '123-45-6788'), ('Bob', '123-
45-6799'))
AS robotccs(username, ssn)
CROSS JOIN (SELECT dearmor('-----BEGIN PGP PUBLIC KEY
BLOCK-----'

```

Version: GnuPG v2.0.14 (GNU/Linux)

mQENBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jt
2HislojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgK1GSgd9texg2nnSL9Adm
R5syVKG+qcdWuvyZg9oO0meyjhc3n+kkbRTEMuM3f1bMs8sh0wzMvstCUV
vG5rJAe8PuYDSJCJ74I6w7SOH3RiRIC7IfL6xYddV4213ctd44b18/i71h
/Hbsjii2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFfnFnE6
ic8MD0lK5/GAlR6Hc0ZIHf8KEcavruQlikjnABEBAAG0HHRlc3Qga2V5ID
a2V5QGVtYWlsLmNvbT6JAT4EEwECACgFAlS1Zf0CGwMFCQHhM4AGCwkIBw
AgkKCwQWAgnMBAh4BAheAAAoJEAd9c14gJ8wwbfwH/3VyVsPkQ11owRJNxv
7BfrvU52yk+PPZYoes9UpdL3CMRk8gAM9bx5Sk08q2UXSZLC6ffOpEW4uW
JR0C3ooezTkmCBW8I1bU0qGetzVxopdXLuPGCE7hVWQe9HcSntiTLxGov1
TAoccXLbyuZh9Rf5vLoQdKzcCyOHh5IqXaQOT100TeFeEpb9TIiwcntg3W
DGOUAOanjDZ3KE8Qp7V74fhG1EZVzHb8FajR62CXSHFKqpBgiNxntOk45N
eTUXPSnwPi46qoAp9UQogsfgYB1XD0TB2UOqhutAMECaM7VtpePv79i0Z/
AQ0EVLV1/QEIANabFdQ+8QMCADOipM1bF/JrQt3zUoc4BTqICaxdyzAfz0
Zro2us99G1ARqLwd8EqJcl/xmfcJiZyUam6ZAzzFXCgnH5Y1sdMTJZdLp
gCWG/ZLu4wzxOFFzDkiPv9RDw6e5MNltJrSp4hS5o2apKdbO4Ex8304mJY
iDDCWU4T01hv3hSKCpke6LcwsX+7liozp+aNmP0Ypwfi4hR3UUMP70+v1b
bVLz31LLouHRgpCzla+PzzbEKs16jq77vG9kqZTCIzXoWaLljiutRlfJko
v/8yAnkcAmowZrIBlyFg2KBzhunYmN2YvkUAEQEAAYkBjQQYAQIADwUCVL
DAUJAeEzgAAKCRAHfXJeICfMMOHYCACFhInZA9uAM3TC441+MrgMUJ3rW9
WrdTsxR8WkSNbIxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRrI61FPHQN
WH+N21asoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8Hkk
HMUC55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sN1GWE
/UUZB+dYqCwtvX0nnBu1KNcmk2AkEcFK3YoliCxomdOxhFOv9AKjjojDyC
Pv2MikPS2fKOAg1R3LpMa8zDET14w3vcKPQNrQNnYuUtfj6ZoCvx
=XZ8J
-----END PGP PUBLIC KEY BLOCK-----' AS pubkey) AS keys;

3. 验证ssn列被加密。

\213\032\226\$\2751\256XR\346k\266\030\234\267\201vUh\004\2
 20\316\306|\203+\010\261;\232\254tp\255\243\261\373Rq;\316
 \322\347ea\220\0151\212g\337\264\336b\263\004\311\210.4\34
 12\342y^202\262|A7\202t\240\333p\345G\373\253\243oCO\011\
 \347\240\005\213\0078\036\210\307\$317\322\311\222\035\354
 3270\013c\327\272\212%\363\033\252\322\337\354\276\225\232



4. 从数据库中提取public.key的ID：

```
SELECT pgp_key_id(dearmor('-----BEGIN PGP PUBLIC KEY
BLOCK-----
Version: GnuPG v2.0.14 (GNU/Linux)

mQENBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jt
2HislojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgK1GSgd9texg2nnSL9Adm
R5syVKG+qcdWuvyZg9oO0meyjhc3n+kkbRTEMuM3f1bMs8shOwzMvstCUV
vG5rJAe8PuYDSJCJ74I6w7SOH3RiRIC7IfL6xYddV4213ctd44bl8/i71h
/Hbsjii2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFfnFnE6
ic8MD0lK5/GAlR6Hc0ZIHf8KEcavruQlikjnABEBAAG0HHRlc3Qga2V5ID
a2V5QGVtYWlsLmNvbT6JAT4EEwECACgFAls1Zf0CGwMFCQHm4AGCwkIBw
AgkKCwQWAQMBAh4BAheAAoJEAd9c14gJ8wwbfwH/3VyVsPkQ11owRJNxv
7BfrvU52yk+PPZYoes9UpdL3CMRk8gAM9bx5Sk08q2UXSZLC6ffOpEW4uW
JRoc3ooezTkmCBW8I1bU0qGetzVxopdXLuPGCE7hvWQe9HcSntiTlxGov1
TAoccXLbyuZh9Rf5vLoQdKzcCyOHh5IqXaQOT100TeFeEpb9TIiwcntg3W
DGouAOanjDZ3KE8Qp7V74fhG1EZVzHb8FajR62CXSHFKqpBgiNxntOk45N
eTUXPSnwPi46qoAp9UQogsfGyB1XD0TB2U0qhutAMECaM7VtpePv79i0Z/
AQ0EVLV1/QEIANabFdQ+8QMCADOipM1bF/JrQt3zUoc4BTqICaxdyzAfz0
Zro2us99GLARqLWd8EqJcl/xmfCJiZyUam6ZAzzFXCgnH5Y1sdtMTJZdLp
gCWG/ZLu4wzxOFFzDkiPv9RDw6e5MNLtJrSp4hS5o2apKdbO4Ex83O4mJY
iDDCWU4T01hv3hSKCpke6LcwsX+7liozp+aNmP0Ypwfi4hR3UUMP70+v1b
bVLz31LLouHRgpCzla+PzzbEKs16jq77vG9kqZTCIzXoWaLljuitRlfJko
v/8yAnkcAmowZrIBlyFg2KBzhunYmN2YvkUAEQEAAYkBQQYAQIADwUCVL
```

```

DAUJAeEzgAAKCRAHfxJeICfMMOHYCACFhInZA9uAM3TC441+MrgMUJ3rW9
WrdTsxR8WkSNbIxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRRI61FPHQN
WH+N21asoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8Hkk
HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNlGWE
/UUZB+dYqCwtvX0nnBu1KNcmk2AkEcFK3YoliCxomdOxhFOv9AKjjojDyC
Pv2MikPS2fKOAg1R3LpMa8zDETl4w3vckPQNrQNnYuUtfj6ZoCxv
=XZ8J
-----END PGP PUBLIC KEY BLOCK-----' )) ;
pgp_key_id | 9D4D255F4FD2EFBB

```

这显示用于加密ssn列的PGP密钥ID是9D4D255F4FD2EFBB。每当新密钥被创建时推荐执行这一步，然后把ID存起来便于跟踪。用户可以使用这个密钥来查看哪个密钥对被用来加密数据：

```

SELECT username, pgp_key_id(ssn) As key_used
FROM userssn;
username | Bob
key_used | 9D4D255F4FD2EFBB
-----
username | Alice
key_used | 9D4D255F4FD2EFBB

```

Note: 不同的密钥可能有相同的ID。这很少见，但是是一种正常事件。客户端应用应该尝试用每一个来解密看看哪个适合 — 就像处理ANYKEY一样。请见pgcrypto文档中的[pgp_key_id\(\)](#)。

5. 使用私钥解密数据。

```

SELECT username, pgp_pub_decrypt(ssn, keys.privkey)
      AS decrypted_ssn FROM userssn
  CROSS JOIN
  (SELECT dearmon('-----BEGIN PGP PRIVATE
KEY BLOCK-----'
Version: GnuPG v2.0.14 (GNU/Linux)

1QOYBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jt
2HislojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgK1GSgd9texg2nnSL9Adm
R5syVKG+qcdWuvyZg9oO0meyjhC3n+kkbRTEMuM3f1bMs8sh0wzMvstCUV
vG5rJAe8PuYDSJCJ74I6w7SOH3RiRIC7IfL6xYddV4213ctd44bl8/i71h
/Hbsji2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFfnFnE6
ic8MD0lK5/GAlR6Hc0ZIHf8KEcavruQlikjnABEBAAEAB/wNfjjvP1brRF

```

XwUNm+sI4v2Ur7qZC94VTukPGf671vqcYZJuqXxvZrZ8bl6mv165xEUiZY
fe0PaM4Wy+Xr94Cz2bPbWgawnRNN3GAQy4rlBTrvqQWy+kmpbd87iTjwZi
02iSzraq41Rt0Zx21Jh4rkpF67ftmzOH0v1rS0bW0vHuEMy7tCwmdPe9Hb
n9C11UqBn4/acTtCClWAjREZn0zXAsNixtTIPC1V+9n09YmecMkVwNfIPk
OPFnuZ/Dz1rCRHjNHb5j6ZyUM5zDqUVnnezktxqrOENSxm0gfMGcpxHQog
6UyBBADSCXHPfo/VPVtMm5p1yGrNOR2jR2rUj9+poZzD2gjkt5G/xIKRlk
emu27wr9dVEX7ms0nvDq58iutbQ4d0JIDlcHMeSRQZluErblB75Vj3HtIm
4Jx6SWRXPUJPGXGI87u0UoBH0Lwij7M2PW711ao+MLEA9jaJQwQA+sr9BK
r5nE72gsbCCLowkC0rdldf1RGtobwYDMpmYZhOaRKjkOTMG6rCXJxrf6Lq
/gNziTmch35MCq/MZzA/bN4VMPyeIlwzxVZkJLsQ7yyqX/A7ac7B7DH0Kf
MSOAJhMmk1W1Q1RRNw3cnYi8w3q7X40EAL/w54FVvvPqp3+sCd86SAapM
tIsuNVemMWdgNXwvK8AJsz7VreVU5yZ4B8hvCuQj1C7geaN/LXhiT8foRs
Bf+iHC/VNEv4k4uDb4l0gnHJYYyifB1wC+nn/EnXCZYQINMiala4M6Vqc/
nwkZt/89LsAiR/20HHR1c3Qga2V5IDx0ZXN0a2V5QGVtYWlsLmNvbT6JAT
ACgFAls1zf0CGwMFCQHhM4AGCwkIBwMCBhUIAgkKCwQWAh4BAheAAA
c14gJ8wwbfwH/3VyVsPkQ11owRJNxvXGt1bY7BfrvU52yk+PPZYoes9Upd
8gAM9bx5Sk08q2UXSZLC6ffOpEW4uWgmGYf8JR0C3ooezTkmCBW8I1bU0q
opdXLuPGCE7hVWQe9HcSntiTLxGov1mJAw07TAoccXLbyuZh9Rf5vLoQdK
h5IqXaQOT100TeFeEpb9TIiwcntg3WCSU5P0DG0UAoanjDZ3KE8Qp7V74f
zHb8Fa jR62CXSHFKqpBgiNxntOk45NbXADn4eTUXPSnwPi46qoAp9UQogs
DOTB2UOqhutAMECaM7VtpEpV79i0Z/NfnBedA5gEVLV1/QEIAnabFdQ+8Q
pM1bF/JrQt3zUoc4BTqICaxdyzAfz0tUSf/7Zro2us99GlARqLWd8EqJcl
iZyUam6ZAzzFXCgnH5Y1sdtMTJZdLp5We0jwgCWG/ZLu4wzxOFFzDkiPv9
MNLtJrSp4hS5o2apKdbO4Ex83O4mJYnav/rEiDDCWU4T0lhv3hSKCpke6L
liozp+aNmP0Ypwfi4hR3UUMP70+V1beFqW2JbVLz31LLouHRgpCzla+Pzz
jq77vG9kqZTCIzXoWaLljuitrlfJkO3vQ9h0v/8yAnkcAmowZrIBlyFg2K
mN2YvkUAEQEAAQAH/A7r4hDrnmzX3QU6FAzePlRB7niJtE2IEN8AufF05Q
c1S72WjtqMAIAgYasDkOhfhcxanTneGuFVYggKT3eSDm1RFKpRjX22m0zK
Mu95V20klul6OCm8d06+2fmkGxGqgc4zsKy+jQxtxK3HG9YxMC0dvA2v2C5

```
Utc7zh//k6IbmaLd7F1d7DXt7Hn2Qsmo8I1rtgPE8grDToomTnRUodToye
ORwsp8n8g2CSFaXSrEyU6HbFYXSxZealhQJGYLF0ZdR0MzVtZQCn/7n+IH
Nd2a8DVx3yQS3dAmvLzhFacZdjXi3lwvj0moFOkEAOCz1E63SKNNksniQ1
gaov6Ux/zGLMstwTzNouI+Kr8/db0G1SAy1Z3UoAB4tFQXEApox9A4AJ2K
cZVULenfDZaxrb9Lid7ZnTDXKVyGTWDF7ZHavHJ4981mCW17lU11zHBB9
dhFvb0gdy0jSLaFMFr/JBAD0fz3RrhP7e6X112zdBqGthjC5S/IoKwwBgw
LoxqBr2pl9PotJJ/JUMPhD/LxuTcOZtYjy8PKgm5jhNBdq3Ss0kNKAY1f5
6I4iAX/NekqSyF+OgBfC9aCgS5RG8hYoOCbp8na5R3bgiuS8IzmVmm5OhZ
nQP7BzmR0p5BahpZ8r3Ada7FcK+0ZLLRdLmOYF/yUrZ53SoYCZRzU/GmtQ
Gjqied9Bs1MHdNUolq7GaexcjZmOWHEf6w9+9M4+vxtQqlnkIWqtaphewE
EP3sIY0EAE3mmiLmHLqBju+UJKMNwFNeyMTqgchg50ISH8J9FRIkBJQQYAQ
VLVl/QIBDAUJAeEzgAAKCRAHFxJeICfMMOHYCACFhInZA9uAM3TC44l+Mr
W9izrO48WrdTsxR8WkSNbIxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRr
QNPSvz62WH+N21asoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQ
kk8qb5x/HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy
WE8pvgeEx/UUZB+dYqCwtvX0nnBu1KNcmk2AkEcFK3YoliCxomdOxhFOv9A
yC65KJciPv2MikPS2fKOAg1R3LpMa8zDETl4w3vckPQNrQNnYuUtfj6ZoC
=fa+6
-----END PGP PRIVATE KEY BLOCK-----') AS privkey) AS
keys;

username | decrypted_ssn
-----
Alice    | 123-45-6788
Bob      | 123-45-6799
(2 rows)
```

如果用户创建了一个带有口令的密钥，用户可能不得不在这里输入它。不过对于这个例子的目的，口令为空。

密钥管理

根据用户是使用对称（单私钥）还是非对称（公私钥）加密算法，有必要安全地存放主密钥或私钥。存储加密密钥有很多种选项，例如放

在文件系统中、密钥保险箱、加密USB、可信平台模块 (TPM) 或者硬件安全性模块 (HSM) 。

在规划密钥管理时，考虑下列问题：

- 密钥将被存储在哪里？
- 密钥应该何时过期？
- 如何保护密钥？
- 如何访问密钥？
- 密钥如何能被恢复和收回？

开放Web应用安全性项目 (OWASP) 为保护加密密钥提供了一份非常全面的指南。 [guide to securing encryption keys](#) 

¹ SHA2算法在版本0.9.8中被加入到OpenSSL，pgcrypto将使用内建代码。

² 任何OpenSSL支持的摘要算法都被自动的取用。但对加密不能这样，加密需要被明确地支持。

³ 从版本0.9.7开始OpenSSL中就包括了AES。对于较老的版本，pgcrypto将使用内建代码。

Greenplum数据库® 6.0文档

□ 安全性配置指南

保护数据库

Greenplum数据库端口和协议

配置客户端认证

配置数据库授权

审计

加密数据和数据库连接

安全性最佳实践

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

[如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息](#)

这一章描述用户应该遵循的基本安全性最佳实践，它可以确保最高级别的系统安全性。

Greenplum数据库的默认安全性配置：

- 仅允许本地连接。
- 为超级用户 (`gpadmin`) 配置的基本认证。
- 超级用户被授权做任何事情。
- 只有数据库角色的口令被加密。

系统用户 (`gpadmin`)

保护和限制对`gpadmin`系统用户的访问。

Greenplum要求一个UNIX用户ID来安装和初始化Greenplum数据库系统。这个系统用户在Greenplum文档中被称作`gpadmin`。`gpadmin`用户是Greenplum数据库中的默认数据库超级用户，也是Greenplum安装及其底层数据文件的文件系统拥有者。默认的管理员账户是Greenplum数据库设计的根本。没有它系统无法运行，并且也没有办法限制`gpadmin`用户ID的访问。

这个`gpadmin`用户可以绕过Greenplum数据库的所有安全性特性。任何人通过这一用户ID登入到Greenplum主机，就可以读取、修改或者删除任何数据，包括系统目录数据和数据库访问权限。因此，非常有必要保护好`gpadmin`用户ID并且只允许必要的系统管理员可以接触到它。

只有在执行特定系统维护任务（例如升级或扩展）时，管理员才应该作为`gpadmin`登入到Greenplum。

数据库用户绝不应作为`gpadmin`登录，并且也绝不应以`gpadmin`运行ETL或者生产负载。

超级用户

被授予SUPERUSER属性的角色是超级用户。超级用户会绕过所有访问特权检查和资源队列。只有系统管理员应该被给予超级用户权

请见Greenplum数据库管理员指南中的“修改角色属性”。

登录用户

为每个登入系统的用户指派一个不同的角色并且设置LOGIN属性。

为了登录和审计的目的，每个被允许登入Greenplum数据库的用户应该被给予他们自己的数据库角色。对于应用或者Web服务，考虑为每一种应用或服务创建一个不同的角色。请见*Greenplum*数据库管理员指南中的“[创建新角色（用户）](#)”。

每一个登录角色应该被指派给一个单一的、非默认的资源队列。

组

使用组管理访问特权。

为每个对象/访问权限的逻辑分组创建一个组。

每个登录用户应该属于一个或者更多角色。使用GRANT语句给角色增加组访问。使用REVOKE语句从角色移除组访问。

不应该为组角色设置LOGIN属性。

见*Greenplum*数据库管理员指南中的“[创建组（角色成员关系）](#)”。

对象特权

只有拥有者和超级用户对新对象拥有完全的权限。权限必须被授予给其他角色（用户或组）以允许它们访问对象。每种类型的数据库对象都有不同的可以被授予的特权。使用GRANT语句给角色增加权限，使用REVOKE语句来移除权限。

用户可以使用REASSIGN OWNED BY语句更改对象的拥有者。例如，要准备删除一个角色，先要更改属于该角色的对象的拥有者。使用DROP OWNED BY删除角色拥有的对象，包括依赖对象。

方案可以被用于实施额外的对象权限检查层，但是方案权限不会覆盖包含在该方案中对象上设置的对象特权。

操作系统用户和文件系统

要防止网络入侵，系统管理员应该验证在组织内使用的口令足够强。下面的推荐可以强化口令：

- 最小口令强度推荐：至少9个字符。MD5口令应该为至少15个字符。
- 混合大小写字母。

- 混合字母和数字。
- 包括非字母数字字符。
- 选择一个用户可以记住的口令。

下面推荐了一些可以用来确定口令强度的口令破解软件。

- John The Ripper。一种快速灵活的口令破解程序。它允许使用多个单词列表并且可以进行蛮力口令破解。该程序可以从<http://www.openwall.com/john/>得到。
- Crack。可能是最著名的口令破解软件，Crack也非常快，但是可能不如John The Ripper那么易用。该软件可以在<https://dropsafe.crypticide.com/alecm/software/crack/c50-faq.html>得到。

整个系统的安全性依赖于root口令的强度。该口令应该至少长达12个字符并且包括大写字母、小写字母、特殊字符和数字的组合。它不能基于任何词典中的词。

应该配置口令过期参数。

确保下面的行存在于文件/etc/libuser.conf的[import]小节中。

```
login_defs = /etc/login.defs
```

确保在[userdefaults]小节中没有以下列文本开头的行，因为这些词会覆盖来自/etc/login.defs的设置：

- LU_SHADOWMAX
- LU_SHADOWMIN
- LU_SHADOWWARNING

确保下面的命令不会产生输出。通过这一命令列出的任何账号都应该被锁定。

```
grep ":+:" /etc/passwd /etc/shadow /etc/group
```

注意：我们强烈推荐客户在初始设置后更改他们的口令。

```
cd /etc
chown root:root passwd shadow group gshadow
chmod 644 passwd group
chmod 400 shadow gshadow
```

找出所有全域可写的文件以及没有设置其粘滞位的文件。

```
find / -xdev -type d \(\ -perm -0002 -a ! -perm -1000 \) -print
```

为前一个命令结果中的所有目录设置其粘滞位 (# chmod +t {dir})。

找出所有全域可写的文件并且修正每一个被列出的文件。

```
find / -xdev -type f -perm -0002 -print
```

为前述命令给出的所有文件设置正确的权限 (# chmod o-w {file})。

找出所有不属于有效用户或组的文件，然后为它们分配一个拥有者或者移除文件。

```
find / -xdev \(-nouser -o -nogroup \) -print
```

找出所有全域可写的目录并且确认它们属于root或者某个系统账户（假定只有系统账户的用户ID低于500）。如果该命令生成输出，验证其分配是否正确或者将它们重新分配给root。

```
find / -xdev -type d -perm -0002 -uid +500 -print
```

口令质量、口令过期策略、口令重用、口令重试尝试以及更多认证设置可以通过可插拔认证模块（PAM）框架配置。PAM在目录/etc/pam.d中查找应用相关的配置信息。运行authconfig或者system-config-authentication将重写PAM配置文件，这会毁掉任何手工更改并且将它们替换为系统默认配置。

默认的PAM模块pam_cracklib提供了口令的强度检查。要配置pam_cracklib以要求至少一个大写字符、小写字符、数字和特殊字符（U.S.国防部指导方针推荐），可编辑文件/etc/pam.d/system-auth并且在对应于口令前置条件pam_cracklib.so try_first_pass的行中包括下列参数。

```
retry=3:  
dcredit=-1. Require at least one digit  
ucredit=-1. Require at least one upper case character  
ocredit=-1. Require at least one special character  
lcredit=-1. Require at least one lower case character  
minlen=14. Require a minimum password length of 14.
```

例如：

```
password required pam_cracklib.so try_first_pass  
retry=3\minlen=14 dcredit=-1 ucredit=-1 ocredit=-1 lcredit=-1
```

可以设置这些参数来反映用户的安全性策略需求。注意口令限制不适用于root口令。

PAM模块pam_tally2提供了在指定失败次数的失败登录尝试之后锁定用户账户的功能。要实施口令封锁，可编辑文件/etc/pam.d/system-auth来包括下列行：

- 第一个认证行应该包括：

```
auth required pam_tally2.so deny=5 onerr=fail  
unlock_time=900
```

- 第一个账户行应该包括：

```
account required pam_tally2.so
```

这里，deny参数被设置以限制重试次数为5并且unlock_time已经被设置为900秒来保持账户在被解锁前锁定900秒。请配置这些参数以反映用户的安全性策略需求。被锁定的账户可以用pam_tally2工具手工解锁：

```
/sbin/pam_tally2 --user {username} -reset
```

用户可以使用PAM限制重用最近用过的口令。可以设置pam_unix模块的remember选项来记住最近的口令并且阻止重用它们。要做到这一点，可在/etc/pam.d/system-auth中编辑适当的行来包括remember选项。

例如：

```
password sufficient pam_unix.so [ ... existing_options ... ]  
remember=5
```

用户可以设置要记住的历史口令的数量以正确地反映其安全性策略需求。

```
cd /etc  
chown root:root passwd shadow group gshadow  
chmod 644 passwd group  
chmod 400 shadow gshadow
```

Parent topic: [安全性配置指南](#)

Greenplum数据库® 6.0文档

□ Greenplum数据库最佳实践

最佳实践概要

系统配置

模式设计

采用资源组管理内存和资源

采用资源队列管理内存和资源

□ 系统监控和维护

装载数据

安全性

加密数据和数据库连接

SQL查询调优

高可用性

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

本章包含Greenplum数据库的最佳实践概要。

数据模型

Greenplum数据库是一种shared nothing的分析型MPP数据库。这种模型与高度规范化的/事务型的SMP数据库有显著区别。正因为如此，推荐以下几项最佳实践。

- Greenplum数据库使用非规范化的模式设计会工作得最好，非规范化的模式适合于MPP分析型处理，例如带有大型事实表和较小维度表的星形模式或者雪花模式。
- 表之间用于连接(join)的列采用相同的数据类型。

详见[模式设计](#)。

堆存储 vs. 追加优化存储

- 经常进行反复的批量或单一UPDATE, DELETE和INSERT操作的表或分区使用堆存储。
- 经常进行并发UPDATE, DELETE和INSERT的表或分区使用堆存储。
- 对于在初始装载后很少更新并且只会在大型批处理操作中进行后续插入的表和分区，使用追加优化存储。
- 绝不在追加优化表上执行单个INSERT, UPDATE或DELETE操作。
- 绝不在追加优化表上执行并发的批量UPDATE或DELETE操作。可以执行并发的批量INSERT操作。

详见[堆存储还是追加优化存储](#)。

行存 vs. 列存

- 如果负载中有要求更新并且频繁执行插入的迭代事务，则对这种负载使用行存。
- 在对宽表选择时使用行存。
- 为一般目的或混合负载使用行存。



- 选择面很窄（很少的列）和在少量列上计算数据聚集时使用列存。
- 如果表中有单个列定期被更新而不修改行中的其他列，则对这种表使用列存。

详见[行存还是列存](#)。

压缩

- 在大型追加优化和分区表上使用压缩以改进系统范围的I/O。
- 在数据最终的存储位置表设置列压缩。
- 平衡压缩解压缩时间和CPU执行周期，选择性能最高的压缩级别。

详见[压缩](#)。

数据分布

- 为所有的表显式定义一个列分布或者随机分布。不要使用默认值。
- 使用能将数据在所有segment上均匀分布的单个列作为分布键。
- 不要采用查询的WHERE条件中使用的列作为分布键。
- 不要采用日期或时间戳作为分布键。
- 不要将同一列同时用于数据分布和分区。
- 在经常做连接(join)操作的大表上采用相同的分布键，这样可以通过本地连接(join)来显著提高性能。
- 在初始装载数据以及增量装载数据之后验证数据没有明显倾斜。

详见[分布](#)。

资源队列内存管理

- 设置vm.overcommit_memory为2。
- 不要配置OS使用大页。
- 使用gp_vmem_protect_limit设置实例可以为每个Segment数据库中执行的所有工作分配的最大内存。
- 可以通过以下计算方法来设置gp_vmem_protect_limit:
 - gp_vmem – Greenplum数据库可用总内存

```
gp_vmem = ((SWAP + RAM) - (7.5GB + 0.05 * RAM)) / 1.7
```

其中SWAP是该主机的交换空间，单位GB，RAM是该主机的内存，单位GB

- max_acting_primary_segments – 当发生主机或主segment故障导致镜像segment切换时，单台主机上可以运行的主segment上限。
- gp_vmem_protect_limit

```
gp_vmem_protect_limit = gp_vmem /  
acting_primary_segments
```

转换成MB来设置配置参数的值。

- 在有大量工作文件被生成的场景下用下面的公式计算将工作文件考虑在内的gp_vmem因子：

```
gp_vmem = ((SWAP + RAM) - (7.5GB + 0.05 * RAM - (300KB *  
total_#_workfiles))) / 1.7
```

- 绝不将gp_vmem_protect_limit设置得过高或者比系统上的物理RAM大。
- 使用计算出的gp_vmem值来计算操作系统参数vm.overcommit_ratio 的设置：

```
vm.overcommit_ratio = (RAM - 0.026 * gp_vmem) / RAM
```

- 使用statement_mem来分配每个segment数据库中用于一个查询的内存。
- 使用资源队列设置活动查询的数目 (ACTIVE_STATEMENTS) 以及队列中查询所能利用的内存量 (MEMORY_LIMIT) 。
- 把所有的用户都与一个资源队列关联。不要使用默认的队列。
- 设置PRIORITY以匹配用于负载以及实际情况的队列的实际需要，避免使用MAX权限。
- 确保资源队列的内存分配不会超过gp_vmem_protect_limit的设置。
- 动态更新资源队列设置以匹配日常操作流。

见[采用资源队列管理内存和资源](#).

分区

- 只对大型表分区。不要分区小表。
- 只有能基于查询条件实现分区消除（分区裁剪）时才使用分区。
- 选择范围分区而舍弃列表分区。
- 基于查询谓词对表分区。
- 不要在同一列上对表进行分布和分区。
- 不要使用默认分区。
- 不要使用多级分区，创建较少的分区让每个分区中有更多数据。
- 通过检查查询的EXPLAIN计划验证查询有选择地扫描分区表（分区被裁剪）。
- 不要用列存储创建太多分区，因为每个Segment上的物理文件总数：`physical files = segments x columns x partitions`

见[分区](#)。

索引

- 通常在Greenplum数据库中无需使用索引。
- 对高基数的表在列式表的单列上创建索引用于钻透目的要求查询具有较高的选择度。
- 不要索引被频繁更新的列。
- 总是在装载数据到表之前删除索引。在装载后，重新为该表创建索引。
- 创建具有选择性的B-树索引。
- 不要在被更新的列上创建位图索引。
- 不要为唯一列、基数非常高或者非常低的数据使用位图索引。位图索引在列值唯一性位于100-100,000之间时性能最好。
- 不要为事务性负载使用位图索引。
- 通常不要索引分区表。如果需要索引，索引列必须与分区列不同。

见[索引](#)。

资源队列

- 使用资源队列来管理集群上的负载。
- 将所有的角色都与一个用户定义的资源队列关联。
- 使用ACTIVE_STATEMENTS参数限制特定队列的成员能并发运行的活动查询数量。
- 使用MEMORY_LIMIT参数控制通过队列运行的查询所能利用的总内存量。
- 动态修改资源队列以匹配负载以及现状。

见[配置资源队列](#).

监控和维护

- 实现*Greenplum*数据库管理员指南中的"推荐的监控和维护任务"。
- 安装时运行gpcheckperf并且在之后定期运行该工具，保存其输出用来比较系统性能随时间的变化。
- 使用手头的所有工具来理解系统在不同负载下的表现。
- 检查任何异常事件以判断成因。
- 通过定期运行执行计划监控查询活动以确保查询被以最优的方式运行。
- 检查执行计划以判断索引是否被使用以及分区消除是否按照预期发生。
- 了解系统日志文件的位置和内容并且定期监控它们，而不是只在问题出现时才去检查日志。

见[系统监控和维护](#), [Query Profiling](#) and [监控Greenplum数据库日志文件](#).

ANALYZE

- 如果分析操作势在必行，那么请做出决定。如果p_autostats_mode设置为on_no_stats（默认）并且表没有被分区，分析操作并不是必须的。
- 处理大量的表时，优先使用analyzedb工具而不是ANALYZE命令，因为analyzedb工具不会分析整个数据库。analyzedb工具可以增量并发地更新统计数据。针对堆表，统计数据会一直被更新。相较而言ANALYZE则不像analyzedb一样去更新表的元数据（用来记录表统计信息是否是最新的）。

- 不要在整个数据库上运行ANALYZE。需要时，有选择地在表级别上运行ANALYZE。
- 在显著改变底层数据的INSERT、UPDATE以及DELETE操作之后总是运行ANALYZE。
- 在CREATE INDEX操作之后总是运行ANALYZE。
- 如果在非常大的表上运行ANALYZE需要太长时间，可以只在用于连接条件、WHERE子句、SORT子句、GROUP BY子句或者HAVING子句的列上运行ANALYZE。
- 当处理大批量表时，使用analyzedb工具而不是使用ANALYZE命令

见[用ANALYZE更新统计信息](#).

Vacuum

- 在大量UPDATE和DELETE操作后运行 VACUUM。
- 不要运行VACUUM FULL。而是运行一个CREATE TABLE...AS 操作，然后重命名并且删掉原始表。
- 频繁地在系统目录上运行VACUUM以避免目录膨胀以及在目录上运行 VACUUM FULL的需要。
- 绝不要杀掉系统目录表上的VACUUM操作。

见[管理数据库膨胀](#).

装载

- 随着segment数目增加最大化并行度。
- 在尽可能多的ETL节点上均匀散布数据。
 - 把非常大型的数据文件分割成相等的部分，并且把数据散布在尽可能多的文件系统上。
 - 每个文件系统运行两个gpfdist实例。
 - 在尽可能多的接口上运行gpfdist。
 - 使用gp_external_max_segs以控制每个gpfdist 服务的segment数量。
 - 总是保持gp_external_max_segs和gpfdist 进程的数量为偶因子。
- 在装载到现有表之前总是删除索引并且在装载之后重建索引。

- 总是在对表装载之后运行VACUUM。

见[装载数据](#)。

安全性

- 保护gpadmin用户ID并且只允许对它进行必需的系统管理员访问。
- 在执行特定的系统维护任务（例如升级或者扩容）时，管理员只应作为gpadmin 登入到Greenplum。
- 限制具有SUPERUSER角色属性的用户。成为超级用户的角色能绕过Greenplum 数据库中的所有访问特权检查以及资源队列。只有系统管理员应该被给予超级用户的权力。请见 *Greenplum*数据库管理员指南中的“修改角色属性”。
- 数据库用户绝不应该以gpadmin登录，且ETL或者生产负载也绝不应该以 gpadmin运行。
- 为每个登入的用户、应用和服务分派一个不同的角色。
- 对于应用或者Web服务，考虑为每个应用或服务创建一个不同的角色。
- 使用组管理访问特权。
- 保护root口令。
- 为操作系统口令强制一种强口令策略。
- 确保重要的操作系统文件受到保护。

见[安全性](#)。

加密

- 加密和解密数据需要性能作为代价，只加密需要加密的数据。
- 在生产系统中实现任何加密方案之前，先执行性能测试。
- 生产Greenplum数据库系统中的服务器证书应该由一个数字证书认证机构 (CA) 签发，这样客户端可以认证该服务器。如果客户端都是机构中的本地客户端，CA可以是本地的。
- 只要客户端到Greenplum数据库的连接会通过不安全的链接，就应该对其使用SSL加密。
- 对称加密方案（加密和解密使用同样的密钥）具有比非对称方案更好的性能，因此在密钥能被安全共享时应当使用 对称加密方案。

使用cryptographic函数来加密磁盘上的数据。数据在数据库进程中被加密和解密，因此有必要用SSL保护客户端连接以避免传输未加密数据。

- 在ETL数据被装载到数据库中或者从数据库中卸载时，使用gpfdists而不是gpfdist。

见[加密数据和数据库连接](#)

高可用性

Note: 以下指导规则在真实硬件部署环境中总结而出，但是并不能直接移植到公有云架构下，该架构下本身已经存在一些高可用手段。

- 使用带有8至24个磁盘的硬件RAID存储方案。
- 使用RAID 1、5或6，这样磁盘阵列能容忍一个失效的磁盘。
- 在磁盘阵列中配置一个热后备以允许在检测到磁盘失效时自动开始重建。
- 通过镜像RAID卷防止重建时整个磁盘阵列失效和退化。
- 定期监控磁盘使用并且在需要时增加额外的空间。
- 监控segment倾斜以确保数据被平均地分布并且在所有segment上存储被平均地消耗。
- 设置一个后备master以便在主master故障后接管。
- 规划当故障发生时，如何把客户端切换到新的master实例，例如，通过更新DNS中的master地址。
- 设置监控机制以便在主Master失效时在系统监控应用中或者通过email发出通知。
- 为所有的segment设置镜像。
- 将主segment和它们的镜像放置在不同的主机上以预防主机故障。
- 迅速地使用gprecoverseg工具恢复故障的segment，以便恢复冗余并且让系统回到最佳平衡。
- 考虑双集群配置以提供额外层次上的冗余以及额外的查询处理吞吐。
- 除非数据库可以很容易地从源端恢复，否则定期备份Greenplum数据库。
- 如果堆表相对较小并且两次备份之间只有很少的追加优化或列存分区被修改，使用增量备份
- 如果备份被保存到NFS挂载点，使用例如Dell EMC Isilon之类的横向扩展NFS方案以避免IO瓶颈。

考虑使用Greenplum集成将备份流式传送给Dell EMC Data Domain或者 Veritas NetBackup企业级备份平台。

见[高可用性](#).

Parent topic: [Greenplum数据库最佳实践](#)

Greenplum数据库® 6.0文档

□ Greenplum数据库最佳实践

最佳实践概要

系统配置

模式设计

采用资源组管理内存和资源

采用资源队列管理内存和资源

□ 系统监控和维护

装载数据

安全性

加密数据和数据库连接

SQL查询调优

高可用性

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

这一节的主题描述配置Greenplum数据库集群主机的要求和最佳实践。

通常使用root用户配置Greenplum数据库集群。

配置时区

Greenplum数据库会从存储在PostgreSQL内部的一个时区集合中选择一个时区使用。PostgreSQL中存储的可用时区全部取自于Internet Assigned Numbers Authority (IANA) 时区数据库，一旦PostgreSQL的IANA数据库发生改变，Greenplum数据库也会随之更新它的可用时区列表。

Greenplum通过将用户定义的时区与PostgreSQL的时区进行匹配来选择自身的时区，如果用户时区没配置，则会采用操作系统主机时区。例如，当选择默认时区时，Greenplum会基于主机操作系统时区文件并根据算法来选择PostgreSQL的时区。如果系统时区包含闰秒信息，Greenplum数据库便不能用PostgreSQL的时区匹配到系统时区。这种情形下，Greenplum数据库会基于主机系统的相关信息来计算一个最佳的PostgreSQL时区匹配值。

作为最佳实践，应该配置Greenplum数据库和主机系统采用已知的被支持的时区。采用当前系统时区和Greenplum数据库时区文件（该信息可能自上次重启后已经从IANA数据库更新）来匹配，这样做可以设置好Greenplum数据库master和segment实例的时区，防止Greenplum数据库每次重启后都重新计算这个最佳匹配值。使用gpconfig工具设置和显示Greenplum数据库时区。例如，以下命令显示Greenplum数据库时区并设置时区为US/Pacific。

```
# gpconfig -s TimeZone
# gpconfig -c TimeZone -v 'US/Pacific'
```

修改时区后必须重启Greenplum数据库。重启Greenplum数据库的命令为gpstop -ra。系统视图 pg_timezone_names 提供Greenplum数据库时区相关的信息。

文件系统

XFS是Greenplum数据库数据目录的最佳实践文件系统。XFS应该用下列选项挂载：

```
rw,nodev,noatime,nobarrier,inode64
```

端口配置

`ip_local_port_range`应该被设置为不与Greenplum数据库端口范围冲突。例如，`/etc/sysctl.conf`文件中的设置：

```
net.ipv4.ip_local_port_range = 10000 65535
```

客户可以设置Greenplum数据库基础端口号为下列值。

```
PORT_BASE = 6000
MIRROR_PORT_BASE = 7000
```

I/O配置

在含有数据目录的设备上，`blockdev`预读尺寸应该被设置为16384。下列命令设置`/dev/sdb`的预读值大小。

```
# /sbin/blockdev --setra 16384 /dev/sdb
```

下列命令显示`/dev/sdb`的预读值大小。

```
# /sbin/blockdev --getra /dev/sdb
16384
```

应该为所有数据目录设备设置deadline IO调度器。

```
# cat /sys/block/sdb/queue/scheduler
noop anticipatory [deadline] cfq
```

应该在`/etc/security/limits.conf`文件中增加OS文件和进程的最大数量。

```
* soft  nofile 65536
* hard  nofile 65536
* soft  nproc 131072
* hard  nproc 131072
```

让内核文件输出到一个已知的位置并且确保`limits.conf`允许输出内核文件。

```
kernel.core_pattern = /var/core/core.%h.%t
# grep core /etc/security/limits.conf
* soft  core unlimited
```

OS内存配置

Linux中sysctl的变

量`vm.overcommit_memory`和`vm.overcommit_ratio`影响操作系统管理内存分配的方式。这些变量应该按照下面的方式设置：

`vm.overcommit_memory`决定OS用于确定为进程可以分配多少内存的方法。这个变量应该总是被 设置为2，它是对数据库唯一安全的设置。

Note: 有关如何配置overcommit memory的信息，参见：

- https://en.wikipedia.org/wiki/Memory_overcommitment
- <https://www.kernel.org/doc/Documentation/vm/overcommit-accounting>

`vm.overcommit_ratio`是被用于应用进程的RAM的百分数。在Red Hat Enterprise Linux上默认是50。为这一变量计算最优值的公式可见[资源队列的segment内存配置](#)。

不要启用操作系统大页设置。

另见[采用资源队列管理内存和资源](#)。

共享内存设置

Greenplum数据库使用共享内存在`postgres`进程之间通信，这些进程是同一个`postgres`实例的组成部分。下面的共享内存设置应该在`sysctl`中设定并且很少会被修改。

```
kernel.shmmmax = 500000000
kernel.shmmni = 4096
kernel.shmall = 4000000000
```

验证操作系统

运行`gpcheck`来验证操作系统配置。更多信息请见Greenplum数据库工具指南中的`gpcheck`部分。

每台主机上的Segment数量

每台segment主机上运行的segment数量对总体系统性能有着巨大的影响。这些segment之间以及主机上的其他进程 共享该主机的CPU核心、内存和

网络接口。过高估计一台服务器能容纳的segment数量是导致非最优性能的常见原因。

在选择每台主机上运行多少Segment时必须要考虑的因素包括：

- 核心数量
- 安装在该服务器上的物理RAM容量
- NIC数量
- 附加到服务器的存储容量
- 主segment和镜像segment的混合
- 将在主机上运行的ETL进程
- 运行在主机上的非Greenplum进程

资源队列的segment内存配置

`gp_vmem_protect_limit`服务器配置参数指定单个segment的所有活动postgres进程在任何给定时刻能够消耗的内存量。查询一旦超过该值则会失败。可使用下面的计算方法为`gp_vmem_protect_limit`估计一个安全值。

1. 使用这个公式计算`gp_vmem` (Greenplum数据库可用的主机内存) :

```
gp_vmem = ((SWAP + RAM) - (7.5GB + 0.05 * RAM)) / 1.7
```

其中`SWAP`是主机的交换空间 (以GB为单位) 而`RAM`是主机上安装的内存 (以GB为单位) 。

2. 计算`max_acting_primary_segments`。当镜像segment由于集群中其他主机上的 segment或者主机故障而被激活时，这是能在一台主机上运行的主segment的最大数量。例如，对于布置在每台主机有8个主segment的四主机块中的镜像来说，单一segment主机失效将会在其所在块中剩余每台主机上激活2个或者3个镜像segment。这一配置的`max_acting_primary_segments`值是11 (8个主Segment外加3个故障时激活的镜像) 。
3. 通过将总的Greenplum数据库内存除以活动主segment的最大数量来计算`gp_vmem_protect_limit`:

```
gp_vmem_protect_limit = gp_vmem /
max_acting_primary_segments
```

转换成兆字节就是`gp_vmem_protect_limit`系统配置参数的设置。

对于有大量工作文件产生的场景，可调整`gp_vmem`的计算以增加工作文件条件：

```
gp_vmem = ((SWAP + RAM) - (7.5GB + 0.05 * RAM - (300KB * total_#_workfiles))) / 1.7
```

有关监控和管理工作文件使用的信息请见*Greenplum*数据库管理员指南。

用户可以根据gp_vmem的值计算操作系统参数
vm.overcommit_ratio的值：

```
vm.overcommit_ratio = (RAM - 0.026 * gp_vmem) / RAM
```

更多有关vm.overcommit_ratio的信息请见[OS内存配置](#)。

另见[采用资源队列管理内存和资源](#)。

资源队列语句内存配置

statement_mem服务器配置参数是分配给segment数据库中任何单个查询的内存量。如果一个语句要求额外的内存，它将溢出到磁盘。用下面的公式计算statement_mem的值：

```
(gp_vmem_protect_limit * .9) /  
max_expected_concurrent_queries
```

例如，如果gp_vmem_protect_limit被设置为8GB (8192MB)，对于40个并发查询，statement_mem的计算可以是：

```
(8192MB * .9) / 40 = 184MB
```

在每个查询被溢出到磁盘之前，它被允许使用184MB内存。

要安全地增加statement_mem的值，用户必须增加gp_vmem_protect_limit或者减少并发的查询数量。要增加gp_vmem_protect_limit，用户必须增加物理RAM或者交换空间，也可以减少每台主机上的segment数量。

注意在集群中增加segment主机无助于内存不足错误，除非用户使用额外的主机来减少每台主机上的segment数量。

当不能提供足够的内存来映射所有的输出时，才会创建溢出文件。通常发生在缓存空间占据达到80%以上时。

另外，使用资源队列管理查询内存的最佳实践可参考[资源管理](#)。

资源队列溢出文件配置

如果查询没有被分配足够的内存，Greenplum数据库会在磁盘上创建溢出文件（也被称为工作文件）。默认单个查询可以创建不超过100,000个溢出文件，这对大部分查询来说都是足够的。

用户可以用配置参数`gp_workfile_limit_files_per_query`控制每个查询和每个segment 创建的溢出文件最大数量。设置该参数为0将允许查询创建无限个溢出文件。限制允许的溢出文件数量可以防止失控的查询损坏系统。

如果一个查询没有被分配足够的内存或者被查询数据中存在数据倾斜，查询可能会生成大量溢出文件。如果查询创建超过指定数量的溢出文件，Greenplum数据库会返回这个错误：

```
ERROR: number of workfiles per query limit exceeded
```

在增加`gp_workfile_limit_files_per_query`的值之前，尝试通过更改查询、改变数据分布 或者更改内存配置来降低溢出文件的数量。

`gp_toolkit`模式包括一些视图可以允许用户查看所有正在使用溢出文件的查询的信息。这些信息 可以被用来排查故障以及查询调优：

- `gp_workfile_entries`视图中包含当前在某个segment上使用工作文件的操作。有关操作 的信息请见[如何阅读执行计划](#)。
- `gp_workfile_usage_per_query`视图包含当前在某个segment上使用工作文件的查询。
- `gp_workfile_usage_per_segment`视图为包含segment信息。每一行显示当前在该 segment上用于工作文件的磁盘空间总量。

这些视图中列的描述请见*Greenplum*数据库参考指南。

`gp_workfile_compression`配置参数指定是否压缩溢出文件。默认设置为`off`。启用压缩可以提高文件溢出时的性能。

Parent topic: [Greenplum数据库最佳实践](#)

Greenplum数据库® 6.0文档

□ Greenplum数据库最佳实践

最佳实践概要

系统配置

模式设计

采用资源组管理内存和资源

采用资源队列管理内存和资源

□ 系统监控和维护

装载数据

安全性

加密数据和数据库连接

SQL查询调优

高可用性

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

这一节包含设计Greenplum数据库模式的最佳实践。

Greenplum数据库是一种分析型的shared-nothing数据库，它和高度规范化的事务型SMP数据库有很大不同。 Greenplum数据库使用非规范化的模式设计会工作得最好，例如带有大型事实表和多个较小维度表的星形模式或者雪花模式。

Parent topic: [Greenplum数据库最佳实践](#)

数据类型

使用一致的数据类型

为表之间的连接列使用相同的数据类型。如果数据类型不同，Greenplum数据库必须动态地转换其中一列的数据类型，这样数据值才能被正确地比较。出于这种考虑，用户可能需要增加数据类型尺寸以便与其他常用对象进行连接操作。

选择使用最小空间的数据类型

通过选择最有效的数据类型存储数据，用户可以增加数据库容量并且改进查询执行能力。

使用TEXT或者VARCHAR而不是CHAR。在这些字符数据类型之间没有性能差异，但使用TEXT或VARCHAR能够降低使用的存储空间。

使用能容纳数据的最小数字数据类型。对适合于INT或SMALLINT的数据使用BIGINT会浪费存储空间。

存储模型

在创建表时，Greenplum数据库提供了一些存储选项。必要了解何时使用堆存储或追加优化（AO）存储，还有何时使用行存储或列存储。在堆和AO以及行和列之间做出正确的选择对于大型的事实表来说极其重要，但是对于小的维度表就不那么重要。

确定存储模型的最佳实践是：

1. 设计并且构建一种只插入的模型，在装载前截断每天的分区。
2. 对于大型的分区事实表，为不同的分区评估并且使用最优的存储选项。一种存储选项对于整个分区表并不总是正确的。例如，有些分区可以是行存的而其他分区是列存的。
3. 在使用面向列的存储时，每一列在每个Greenplum数据库Segment上都是一个单独的文件。对于具有大量列的表，对经常访问的数据（热数据）考虑列存，对不经常访问的数据（冷数据）考虑面向行的存储。
4. 存储选项应该在分区级别设置。
5. 如果需要，压缩大型表以提升I/O性能并且在集群中腾出空间。

堆存储还是追加优化存储

堆存储是默认存储模型，并且是PostgreSQL为所有数据库表使用的模型。为频繁进行UPDATE、DELETE以及单个INSERT操作的表和分区使用堆存储。为将收到并发 UPDATE、DELETE以及 INSERT操作的表和分区 使用堆存储。

为初始装载后就很少被更新并且后续只会以批操作执行插入的表和分区使用追加优化存储。绝不要在追加优化表上 执行单个INSERT、UPDATE或者DELETE操作。并发的批量INSERT操作可以被执行但是绝不执行并发的批量UPDATE 或者DELETE操作。

追加优化表中被更新和删除的行所占用的空间不会像堆表那样被有效地回收及重用，因此追加优化存储模型不适合于频繁更新的表。它的设计目标是用于一次装载、很少更新且频繁进行分析查询处理的大型表。

行存还是列存

按行存储数据是传统的存储数据库元组的方式。组成一行的列被连续地存储在磁盘上，因此整个行可以被以单次I/O的形式从磁盘上读出。

面向列的方式把列值在磁盘上存在一起。对每一列都会创建一个单独的文件。如果表被分区，则会对每个列 和分区的组合创建一个单独的文件。当一个查询在一个有很多列的列存表中访问少量列时，I/O代价

会比行存表要减少很多，因为不必从磁盘上检索没有被引用的列。

对于包含要求更新并且频繁执行插入的事务的交易型负载，推荐使用面向行的存储。当对表的选择很宽（即查询中需要单个行的很多列）时，应该使用面向行的存储。如果大部分列出现在查询的 SELECT 列表或者 WHERE 子句中，请使用行存储。对一般目的或者混合负载使用面向行的存储，因为它能提供灵活性和性能的最佳组合。

面向列的存储是为了读操作而优化，但它并未对写操作优化，一行的列值必须被写入到磁盘上的不同位置。对于有很多列的大型表，当查询中只访问列的一个小集合时，列存表可以提供最优查询性能。

列存的另一个好处是，同一种数据类型的值集合可以用比混合类型值集合更少的空间存储在一起，因此列存表比行存表使用的磁盘空间更少（进而导致需要更少的磁盘 I/O）。列存表的压缩效果也比行存表更好。

对于数据仓库的分析型负载，其中的选择很窄或者在少量列上计算数据聚集，请使用面向列的存储。对于定期更新单个列但不修改行中其他列的表，使用面向列的存储。在一个很宽的列存表中读取一个完整的行比在行存表中读取同样一行需要更多时间。有必要理解每个列都是 Greenplum 数据库中每个 segment 上一个单独的物理文件。

压缩

Greenplum 数据库提供了多种选项以压缩追加优化表和分区。压缩数据允许在每次磁盘读取操作中读取更多的数据，这样就能提高系统的 I/O。最佳实践是在数据所在的层次设置列的压缩设置。

注意被增加到分区表的新分区不会自动继承表级定义的压缩，在增加新分区时，用户必须明确地定义压缩。

Run-length encoding (RLE) 压缩提供了最好的压缩级别。较高的压缩级别通常会使数据在磁盘上更加紧凑的存储，但是写入时的数据压缩和读取时的数据解压会要求额外的时间和 CPU 周期。排序数据并且结合多种压缩选项可以实现最高的压缩级别。

绝不要对存储在压缩文件系统上的数据使用数据压缩。

测试不同的压缩类型和排序方法以确定对用户特定数据的最佳压缩方式。例如，客户可以从 zstd 8 级或 9 级压缩开始，然后调整参数达到最理想的结果。RLE 压缩在存储文件中包含大量重复数据时工作效果最好。

分布

能让数据被均匀分布的最优分布方式是Greenplum数据库使用过程中的一个重要因素。在一个MPP无共享环境中，一个查询的总体响应时间由所有segment的完成时间度量。整个系统的响应速度和最慢的segment正相关。如果数据发生倾斜，拥有更多数据的segment将需要更多时间完成，因此每一个segment必须有大约相同数据量的行并且执行大概相同量级的处理。如果一个Segment比其他segment有明显更多的数据要处理，将会导致糟糕的性能和内存不足的情况。

在决定分布策略时，考虑下列最佳实践：

- 为所有的表明确定义一个分布列或者随机分布。不要使用默认分布。
- 理想情况下，使用单个将数据在所有Segment之间均匀分布的列作为分布列。
- 不要将查询的WHERE子句中将要使用的列作为分布列。
- 不要在日期或者时间戳上分布。
- 分布键列数据应该含有唯一值或者非常高的可辨别性。
- 如果单个列无法实现均匀分布，则使用多列分布键，但不要超过两列。额外的列值通常不会得到更均匀的分布，而且它们要求额外的哈希处理时间。
- 如果两个列的分布键无法实现数据的均匀分布，则使用随机分布。大部分情况中的多列分布键都要求移动操作来连接表，因此它们对于随机分布来说没有优势。

Greenplum数据库的随机分布不是循环的，因此无法保证每个segment上的记录数相等。随机分布通常会落在变化低于10个百分点的目标范围内。

在连接大型表时，最优分布非常关键。为了执行连接，匹配的行必须位于同一个segment上。如果数据没有按照同一个连接列分布，其中一个表中需要的行会被动态重新分布到其他segment上。在一些情况下会执行一次广播移动而不是执行重新分布移动，在这种情况下每个segment都会重新对数据进行哈希操作并根据哈希键将对应的行发送到合适的segment上。

本地（局内）连接

使用在所有segment之间均匀分布表行并且达到本地连接的哈希分布能够提供可观的性能受益。当被连接的行在同一个segment上时，很多

处理都可以在该segment实例内完成。这被称为本地或者局内连接。本地连接能最小化数据移动，每一个segment都独立于其他segment操作，不需要segment之间的网络流量或通信。

为了在常被连接在一起的大型表上实现本地连接，请在相同的列上分布这些表。本地连接要求连接的两边都被按照相同的列（以及相同的顺序）分布并且连接表时使用分布子句中的所有列。分布列还必须是同样的数据类型——虽然一些具有不同数据类型的值看起来有相同的表现形式，但它们的存储方式不同并且会被哈希为不同的值，因此它们会被存放在不同的segment上。

数据倾斜

数据倾斜通常是糟糕查询和内存不足的根源。倾斜的数据会影响扫描（读取）性能，但它还影响所有其他的执行操作，例如操作执行的连接和分组。

有必要验证分布以确保数据在初始装载之后被均匀地分布。在增量装载之后继续验证分布也同等重要。

下列查询显示每个segment的行数以及与最大、最小行数之间的差异：

```
SELECT 'Example Table' AS "Table Name",
       max(c) AS "Max Seg Rows", min(c) AS "Min Seg Rows",
       (max(c)-min(c))*100.0/max(c) AS "Percentage Difference
Between Max & Min"
FROM (SELECT count(*) c, gp_segment_id FROM facts GROUP BY
2) AS a;
```

gp_toolkit模式有两个可以用来检查倾斜的视图。

- `gp_toolkit.gp_skew_coefficients` 视图通过计算存储在每个segment上的数据的变异系数 (CV) 来显示数据分布倾斜。`skccoeff` 列显示变异系数 (CV)，它由标准偏差除以均值算出。它同时考虑均值和围绕一个数据序列的均值的变化性。值越低，情况就越好。值越高表明数据倾斜越严重。
- `gp_toolkit.gp_skew_idle_fractions` 视图通过计算一次表扫描期间系统空闲的百分数来显示数据分布倾斜，这种数据是计算性倾斜的指示器。`siffraction` 列显示在一次表扫描期间系统处于空闲的百分数。这是一种非均匀数据分布或者查询处理倾斜的指示器。例如，值为 0.1 表示 10% 的倾斜，值为 0.5 表示 50% 的倾斜等等。如果表的倾斜超过 10%，就应该评估其分布策略。

处理倾斜

当不成比例的数据量流入一个或者少数segment并被它们处理时，处理倾斜就会发生。它常常就是Greenplum数据库性能和稳定性问题背后的罪人。它可能随着连接、排序、聚集和多种OLAP操作而发生。查询倾斜在查询执行时才会发生，因此并不如数据倾斜那么容易检测，数据倾斜由于错误的分布键选择导致的非均匀数据分布而产生。数据倾斜存在于表级别，因此它可以很容易地检测到并且通过选择最优的分布键来避免。

如果单个segment故障（也就是说并非主机上所有segment失效），可能就会是一个处理倾斜问题。当前确定处理倾斜还是一种手工处理。首先查看溢出文件。如果有倾斜但还不足以导致溢出，这将不会成为一种性能问题。如果使用者确定倾斜存在，接着查找对该倾斜负责的查询。下面是这个处理过程要使用的步骤和命令（请相应地更改传递给gpssh 的主机文件名之类的东西）：

1. 查找要在其中监控倾斜处理的数据库的OID：

```
SELECT oid, datname FROM pg_database;
```

其输出的例子：

oid	datname
17088	gpadmin
10899	postgres
1	template1
10898	template0
38817	pws
39682	gpperfmon
(6 rows)	

2. 运行一个gpssh命令以在系统中所有的Segment节点间检查文件尺寸。把<OID>用前一个命令中得到的数据库OID替换：

```
[gpadmin@mdw kend]$ gpssh -f ~/hosts -e \
"du -b /data[1-
2]/primary/gpseg*/base/<OID>/pgsql_tmp/*" | \
grep -v "du -b" | sort | awk -F" " '{ arr[$1] =
arr[$1] + $2 ; tot = tot + $2 }; END \
{ for ( i in arr ) print "Segment node" i, arr[i],
"bytes (" arr[i]/(1024**3)" GB)"; \
print "Total", tot, "bytes (" tot/(1024**3)" GB)" }'
```

其输出的例子：

```

Segment node[sdw1] 2443370457 bytes (2.27557 GB)
Segment node[sdw2] 1766575328 bytes (1.64525 GB)
Segment node[sdw3] 1761686551 bytes (1.6407 GB)
Segment node[sdw4] 1780301617 bytes (1.65804 GB)
Segment node[sdw5] 1742543599 bytes (1.62287 GB)
Segment node[sdw6] 1830073754 bytes (1.70439 GB)
Segment node[sdw7] 1767310099 bytes (1.64594 GB)
Segment node[sdw8] 1765105802 bytes (1.64388 GB)
Total 14856967207 bytes (13.8366 GB)

```

如果在磁盘使用上有显著且持续的差别，那么应该研究正在被执行的查询看看有没有倾斜（上面的输出例子并未表明明显的倾斜）。在被监控的系统中，总是会有一点倾斜，但通常它们是短暂的并且将会持续很短的时间。

3. 如果显著且持久的倾斜出现，下一个任务就是确定导致问题的查询。

前一步的命令已经摘要了整个节点。这一次，要找到实际的segment目录。使用者可以从master或者通过登入前一步确定的特定节点来做这些。下面是一个从master运行的例子。

这个例子专门地查找排序文件。并非所有的溢出文件或者倾斜情况都由排序文件导致，因此使用者将需要自定义这个命令：

```

$ gpssh -f ~/hosts -e
  "ls -l /data[1-
2]/primary/gpseg*/base/19979/pgsql_tmp/*"
  | grep -i sort | awk '{sub(/base.*tmp\//, "...",
$10); print $1,$6,$10}' | sort -k2 -n

```

下面是来自这个命令的输出：

```

[sdw1] 288718848

/data1/primary/gpseg2/.../pgsql_tmp_slice0_sort_17758_0001
291176448

/data2/primary/gpseg5/.../pgsql_tmp_slice0_sort_17764_0001
924581888

/data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_00
980582400

/data1/primary/gpseg18/.../pgsql_tmp_slice10_sort_29425_00
986447872

/data2/primary/gpseg35/.../pgsql_tmp_slice10_sort_29602_00
[sdw5] 999620608

/data1/primary/gpseg26/.../pgsql_tmp_slice10_sort_28637_00
999751680

/data2/primary/gpseg9/.../pgsql_tmp_slice10_sort_3969_0001

```

```
1000112128
```

```
/data1/primary/gpseg13/.../pgsql_tmp_slice10_sort_24723_00
1000898560

/data2/primary/gpseg28/.../pgsql_tmp_slice10_sort_28641_00
[sdw8] 1008009216

/data1/primary/gpseg44/.../pgsql_tmp_slice10_sort_15671_00
1008566272

/data1/primary/gpseg24/.../pgsql_tmp_slice10_sort_28633_00
1009451008

/data1/primary/gpseg19/.../pgsql_tmp_slice10_sort_29427_00
1011187712

/data1/primary/gpseg37/.../pgsql_tmp_slice10_sort_18526_00
1573741824

/data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_00
1573741824

/data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_00
1573741824

/data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_00
1573741824

/data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_00
1573741824

/data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_00
1573741824
```

```
/data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_00
```

扫描这一输出将会揭示出主机sdw8上的名为gpseg45的segment是罪魁祸首。

- 用ssh登入导致问题的节点并且成为root。使用lsof命令查找拥有排序文件的进程的PID：

```
[root@sdw8 ~]# lsof
/data2/primary/gpseg45/base/19979/pgsql_tmp/pgsql_tmp_slic
```

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE
NODE				NAME		
postgres	15673	gpadmin	1lu	REG	8,48	1073741824
	64424546751			/data2/primary/gpseg45/base/19979/pgsql_tmp/pgsql_tmp_slice		

The PID 15673也是文件名的一部分，但并不总是这样。

5. 用该PID作为参数运行ps命令以确定数据库和连接信息：

```
[root@sdw8 ~]# ps -eaf | grep 15673
gpadmin 15673 27471 28 12:05 ? 00:12:59
postgres: port 40003, sbaskin bdw
172.28.12.250(21813) con699238 seg45 cmd32
slice10 MPPEXEC SELECT
root 29622 29566 0 12:50 pts/16 00:00:00 grep
15673
```

6. 在master上，在pg_log日志文件中查找上一个命令中的用户（sbaskin）、连接（con699238）以及命令编号（cmd32）。日志文件中含有这三个值的行应该是包含该查询的行，但偶尔命令编号可能会略有不同。例如，ps输出可能显示cmd32，但在日志文件中是cmd34。如果该查询仍在运行，该用户和连接的最后一个查询就是导致该问题的原因。

对于处理倾斜的纠正几乎都是重写该查询。创建临时表可以消除倾斜。临时表可以被随机地分布以强制一种两阶段的聚集。

分区

一种好的分区策略可以通过只读取满足查询所需的分区来降低被扫描的数据量。

每个分区在每一个segment上都是一个单独的物理文件或文件集合（这种情况出现在列寸表上）。就像在宽列存表中读取一整行比从堆表读取同一行需要更多时间一样，在分区表中读取所有分区比从非分区表中读取相同的数据要求更多的时间。

下面是分区的最佳实践：

- 只分区大型表，不要分区小型表。
- 仅当可以基于查询条件实现分区消除（分区裁剪）并且可以基于查询谓词对表分区来完成分区消除时才在大型表上使用分区。无论何时，优先使用范围分区而不是列表分区。
- 只有当查询中where选择条件包含表的分区列使用不可变操作符

(例如`=`、`<`、`<=`、`>`、`>=`以及`<>`) 时，查询规划器才能有选择地扫描分区表。

- 选择性扫描会识别`STABLE`和`IMMUTABLE`函数，但是不识别查询中的`VOLATILE`函数。例如，

```
date > CURRENT_DATE
```

之类的`WHERE` 子句会导致查询规划器选择性地扫描分区表，但

```
time > TIMEofday
```

之类的`WHERE` 子句却不行。有必要通过`EXPLAIN`检查执行计划来验证查询是否选择性地扫描分区表（分区被裁剪）。

- 不要使用默认分区。默认分区总是会被扫描，但是更重要的是，在很多情况下它们会被填得太满导致糟糕的性能。
- 绝不在相同的列上对表分区和分布。
- 不要使用多级分区。虽然支持子分区，但并不推荐使用这种特性，因为通常子分区包含很少的数据或者不包含数据。分区或者子分区数量增加时性能也增加简直就是天方夜谭。维护很多分区和子分区的管理工作将会压过得到的性能收益。为了性能、可扩展性以及可管理性，请在分区扫描性能和总体分区数量之间做出平衡。
- 谨防对列式存储使用太多分区。
- 考虑负载并发性以及为所有并发查询打开并且扫描的平均分区数。

分区和列存文件的数量

Greenplum数据库支持的文件数的唯一硬限制是操作系统的打开文件限制。但是，有必要考虑集群中文件的总数、每个`segment`上的文件数以及一台主机上的文件总数。在一个MPP无共享环境中，每一个节点都独立于其他节点操作。每个节点受到其磁盘、CPU和内存的约束。CPU和I/O约束对Greenplum数据库并不常见，但内存常常是一种限制因素，因为查询执行模型会在内存中优化查询性能。

每个`segment`上的最优文件数也基于该节点上的`segment`数量、集群的大小、SQL访问、并发、负载和倾斜等因素而变化。通常在每台主机上有六到八个`segment`，大的集群中每台主机可能有很少的`segment`。当使用分区和列存时，更重要的是考虑每个`segment`的文件数和节点上的文件总数。

例子 DCA V2 每节点64GB内存

- 节点数：16
- 每节点的segment数：8
- 每个segment的平均文件数：10,000

每节点上的文件总数是 $8 * 10,000 = 80,000$ ，而该集群的文件总数是 $8 * 16 * 10,000 = 1,280,000$ 。随着分区数和列数的增加，文件数会快速增加。

作为最佳实践最推荐的设置，请把每节点的文件总数限制为低于100,000。如上一个例子所示，每个 segment的最优文件数和每节点的文件总数取决于节点的硬件配置（主要是内存）、集群的大小、SQL 访问、并发性、负载以及倾斜。

索引

在Greenplum数据库中通常不需要索引。大部分分析型查询会在大体量数据上操作，而索引是用于从多行数据中定位某一行或某几行。

在Greenplum数据库中，顺序扫描是一种读取数据的有效方法，因为每个segment都含有数据同等大小的一部分并且所有的segment都并行工作以读取数据。

如果增加索引不能获得性能提升，马上删掉它。验证您创建的每个索引都被优化器使用到。

对于具有高选择性的查询，索引可能会提升查询性能。对于选择性查询所要求的高基数表，在一个列式表的单列上 创建用于钻透目的的索引。

不要在频繁更新的列上创建索引。在频繁被更新的列上创建索引会增加更新时所需的写次数。

只有当表达式被频繁地使用在查询中时，才应该在表达式上建立索引。

带有谓词的索引会创建一个部分索引，它可以被用来从大型表中选择少量行。

避免重叠的索引。具有相同前导列的索引是冗余的。

对于返回一个定向行集合的查询来说，索引能够提高在压缩追加优化表上的性能。对于压缩数据，采用索引访问 方法意味着只有必要的页面会被解压缩。

创建有选择性的B-树索引。索引选择度是一列中的唯一值数量除以表

中的行数。例如，如果一个表有1000行并且有一列中有800个唯一值，那么该索引的选择度就是0.8，这被认为是中不错的索引使用情形。

总是在向表中装载数据前删除索引。这样装载的运行速度将会比在带有索引的表中装载数据快一个数量级。在装载之后，重新创建索引。

位图索引适合于查询但不适合于更新。当列具有较低的基数（100到100,000个唯一值）时位图索引表现得最好。不要为唯一列、基数非常高或者非常低的数据使用位图索引。不要为事务性负载使用位图索引。

通常，不要索引分区表。如果需要索引，索引列必须不同于分区列。索引分区表的一个好处是因为当B-树尺寸增长时其性能呈指数下降，在分区表上创建索引可以得到很多较小的B-树，其性能比未分区表上的B-树更好。

列顺序与字节对齐

为了最优性能，请布置表列以实现数据类型的字节对齐。以下面的顺序布置堆表中的列：

1. 分布列和分区列
2. 固定的数字类型
3. 可变的数据类型

从大到小布置数据类型，这样BIGINT和TIMESTAMP会在INT和DATE的前面，而所有这些类型都在TEXT、VARCHAR或者NUMERIC(x,y)之前。例如，首先是8字节类型(BIGINT、TIMESTAMP)，接着是4字节类型(INT、DATE)，再后面是2字节类型(SMALLINT)，而可变数据类型在最后(VARCHAR)。

不要以这种顺序定义列：

```
Int, Bigint, Timestamp, Bigint, Timestamp, Int (分布键) , Date (分区键) , Bigint, Smallint
```

以这种顺序定义列：

```
Int (分布键) , Date (分区键) , Bigint, Bigint, Timestamp, Bigint, Timestamp, Int, Smallint
```


Greenplum数据库® 6.0文档

□ Greenplum数据库最佳实践

最佳实践概要

系统配置

模式设计

采用资源组管理内存和资源

采用资源队列管理内存和资源

□ 系统监控和维护

装载数据

安全性

加密数据和数据库连接

SQL查询调优

高可用性

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

采用资源组管理Greenplum数据库资源。

在Greenplum数据库集群中，内存、CPU和并发事务管理对性能影响非常巨大。资源组是Greenplum数据库提供的用来强制限制内存、CPU和并发事务的全新资源管理模式。

- [配置Greenplum数据库内存](#)
- [配置资源组](#)
- [低内存查询](#)
- [管理工具和admin_group并行](#)

配置Greenplum数据库内存

一直增加系统内存是不可能的，客户可以通过配置资源组来管理可预期的工作负载，这样能够避免内存溢出的情况发生。

以下操作系统和Greenplum数据库内存设置对采用资源组来管理日常工作是非常有用的：

- **`vm.overcommit_memory`**

该Linux内核参数在`/etc/sysctl.conf`文件中设置，用来指定操作系统分配给系统进程使用多少内存的方法。`vm.overcommit_memory`在Greenplum数据库所在的机器上必须设置为2。

- **`vm.overcommit_ratio`**

该Linux内核参数在`/etc/sysctl.conf`文件中设置，用来执行应用进程可以使用的内存百分比；剩余的内存留给操作系统。操作系统默认值（Red Hat上默认是50）对于部署Greenplum数据库集群基于资源组的管理方式是一个不错的初始值。如果感觉内存利用率太低，便可以提高该值；如果内存或交换分区使用太高，就减少该设置。

- **`gp_resource_group_memory_limit`**

系统分配给Greenplum数据库的内存百分比。默认值为.7(70%)。

- **`gp_workfile_limit_files_per_query`**

设置`gp_workfile_limit_files_per_query`以限制每个查询允许使用的临时溢出文件（工作文件）的最大数量。当查询要求的内存比它能分配的更多时，它将创建溢出文件。当上述限制被超过时，查询会被中止。默认值为零，允许无限多的溢出文件并且可能

会填满文件系统。

- **gp_workfile_compression**

如果有很多溢出文件，则设置`gp_workfile_compression`来压缩这些溢出文件。压缩溢出文件可能有助于避免IO操作导致磁盘子系统过载。

其他考虑因素：

- 不要启用操作系统大页配置。
- 当您配置资源组内存时，提前考虑出现segment实例或segment主机宕机时，镜像segment变成主segment对系统内存的占用。

配置资源组

Greenplum数据库资源组能提供管理集群负载的强有力手段。当您在系统中配置资源组时，考虑以下常规指导方法：

- 任何具有SUPERUSER权限的用户提交的事务都在默认资源组`admin_group`下运行。在调度和运行任何Greenplum管理工具时都要牢记这一点。
- 确保为每一个非管理员用户分配一个用户组。如果不给用户分配资源组，那么该用户提交的查询会被默认资源组`default_group`处理。
- 采用资源组参数`CONCURRENCY`来限制某个资源组可以并发运行的活动查询的数量。
- 采用`MEMORY_LIMIT`和`MEMORY_SHARED_QUOTA`参数 控制运行在资源组中的查询可以申请的最大内存数量。
- Greenplum数据库会将无保留内存（100-（所有资源组`MEMORY_LIMIT`总和））全部分配给全局共享内存池。该内存本着一视同仁的原则，先到先得。
- 基于实时需求和负载的变化来动态调整资源组满足业务要求。
- 采用`gptoolkit`视图检查资源组使用情况，来监控资源组工作良好。

低内存查询

`memory_spill_ratio`设置为较低值时（例如，在0-2%之间）能够提升低内存要求查询的性能。我们可以在每个查询之前让`memory_spill_ratio`生效来覆盖系统默认设置。例如：

```
SET memory_spill_ratio=0;
```

管理工具和admin_group并行

Greenplum数据库用户SUPERUSERs的默认资源组是admin_group。admin_group资源组的默认CONCURRENCY值为10。

某些Greenplum数据库管理工具可能会同时使用多个CONCURRENCY槽，例如 使用gpbackup带有--jobs选项时。如果客户运行的工具 要求的并发事务数比admin_group的多，可以考虑临时增加该资源组的 MEMORY_LIMIT和CONCURRENCY，以满足工具的要求，但一定要记得在工具执行完后及时将这些设置恢复原样。

Note: 通过修改ALTER RESOURCE GROUP达到的内存改变并不能立刻影响到正在运行的 查询。所以尽量选择在维护窗口时间修改资源组参数。

Parent topic: [Greenplum数据库最佳实践](#)

源

Greenplum数据库® 6.0文档

□ Greenplum数据库最佳实践

最佳实践概要

系统配置

模式设计

采用资源组管理内存和资源

采用资源队列管理内存和资源

□ 系统监控和维护

装载数据

安全性

加密数据和数据库连接

SQL查询调优

高可用性

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

[如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息](#)

避免内存错误，管理Greenplum数据库资源。

Note: 资源组是Greenplum数据库提供的用来强制限制内存、CPU和并发事务的全新资源管理模式。 [管理资源组](#)这个话题提供了对资源队列和资源组管理模式的比较。有关资源管理模式的配置和使用方法请见[使用资源组](#)。

Greenplum数据库集群中，内存管理对性能有显著的影响。默认设置适合于大部分环境。不要更改默认设置，除非理解系统上的内存特点和使用方法。

- [解决内存不足错误](#)
- [低内存查询](#)
- [为Greenplum数据库配置内存](#)
- [配置资源队列](#)

解决内存不足错误

内存不足错误消息表明Greenplum的segment、主机和进程遇到了内存不足错误。例如：

```
Out of memory (seg27 host.example.com pid=47093)
VM Protect failed to allocate 4096 bytes, 0 MB available
```

Greenplum数据库中一些常见的导致内存不足的情况有：

- 集群上可用的系统内存（RAM）不足
- 内存参数配置不当
- segment级别有数据倾斜
- 查询级别有操作性倾斜

下面是内存不足情况的可用解决方案：

- 调优查询以要求较少的内存
- 使用资源队列降低查询并发
- 在数据库级别上验证gp_vmem_protect_limit配置参数。最大安全设置的计算请见[为Greenplum数据库配置内存](#)。



- 在资源队列上设置内存限额以限制资源队列中执行的查询所使用的内存
- 使用会话设置来降低特定查询使用的statement_mem
- 在数据库级别降低statement_mem
- 降低Greenplum集群中每台主机上的Segment数量。该操作要求重新出实话集群并重新加载数据。
- 如果可能，增加主机上的内存。（要求增加额外的硬件。）

向集群中增加segment主机本身不会缓解内存不足问题。每个查询使用的内存由statement_mem参数决定，并且在查询被调用时会设置它。不过，如果增加更多的主机允许每台主机上的segment数量降低，那么gp_vmem_protect_limit中分配的内存量就可以上升。

低内存查询

较低的statement_mem设置（例如，在1-3MB之间）可以提升低内存查询的性能。可以在语句级别设置statement_mem配置参数来覆盖系统默认值。例如：

```
SET statement_mem='2MB';
```

为Greenplum数据库配置内存

如果合理地管理内存，大部分内存不足的情况是可以避免的。

一直增加系统内存是不可能的，客户可以通过配置资源组来管理可预期的工作负载，这样能够避免内存溢出的情况发生。

当您配置资源组内存时，提前考虑出现segment实例或segment主机宕机时，镜像segment变成主segment对系统内存的占用。

下面是推荐的操作系统以及Greenplum数据库内存设置：

- 不要启用操作系统大页配置
- **vm.overcommit_memory**

该Linux内核参数在/etc/sysctl.conf文件中设置，用来指定操作系统分配给系统进程使用多少内存的方法。vm.overcommit_memory在Greenplum数据库所在的机器上必须设置为2。

- **vm.overcommit_ratio**

该Linux内核参数在`/etc/sysctl.conf`文件中设置，用来执行应用进程可以使用的内存百分比；剩余的内存留给操作系统。操作系统默认值（Red Hat上默认是50）。

把`vm.overcommit_ratio`设置得太高可能会导致没有为操作系统保留足够的内存，进而导致segment主机故障或者数据库故障。将这个值设置得太低会降低并发量和通过降低Greenplum数据库可用内存量能运行的查询复杂度。当增加这一设置时，有必要记住总是为操作系统活动保留一些内存。

有关计算`vm.overcommit_ratio`值的步骤，请见[资源队列的segment内存配置](#)。

- **gp_vmem_protect_limit**

使用`gp_vmem_protect_limit`设置实例能够为在每个segment数据库中完成的所有工作分配的最大内存。不要把这个值设置得高于系统上的物理RAM。如果`gp_vmem_protect_limit`太高，有可能耗尽系统上的内存并且正常的操作可能会失败，导致segment故障。如果`gp_vmem_protect_limit`被设置为一个安全的较低值，系统上真正的内存耗尽就能避免。查询可能会因为达到限制而失败，但是系统崩溃和segment故障可以避免，这也是我们想要的行为。

有关计算`gp_vmem_protect_limit`安全值的步骤，请见[资源队列的segment内存配置](#)。

- **runaway_detector_activation_percent**

失控查询终止在Greenplum数据库4.3.4中被引入，它能防止内存不足的问题。`runaway_detector_activation_percent`系统参数控制触发查询终止的`gp_vmem_protect_limit`内存利用率。

默认它被设置为90%。如果一个segment利用的`gp_vmem_protect_limit memory`内存的百分比超过指定的值，Greenplum数据库会基于内存使用终止查询，从消耗内存量最大的查询开始。查询会被挨个终止直至`gp_vmem_protect_limit`的利用率重新低于指定的百分比。

- **statement_mem**

使用`statement_mem`分配每个segment数据库中一个查询所使用的内存。如果要求额外的内存，将会溢出到磁盘。按照下面的方式为`statement_mem`设置最优点：

```
(vmpredict * .9) / max_expected_concurrent_queries
```

`statement_mem`的默认值是125MB。例如，一个使用默认`statement_mem`值运行在Dell EMC DCA V2系统上的查询将在每台segment服务器上使用1GB内存（8个Segment × 125MB）。为要求额外内存完成的特定查询在会话级别上设置`statement_mem`。在并发性低的集群上这种设置可以很好地管

理查询内存。对于高并发的集群还是要使用资源队列来控制在系统上运行什么以及运行多少。

- **gp_workfile_limit_files_per_query**

设置`gp_workfile_limit_files_per_query`以限制每个查询允许使用的临时溢出文件（工作文件）的最大数量。当查询要求的内存比它能分配的更多时，它将创建溢出文件。当上述限制被超过时，查询会被中止。默认值为零，允许无限多的溢出文件并且可能会填满文件系统。

- **gp_workfile_compression**

如果有很多溢出文件，则设置`gp_workfile_compression`来压缩这些溢出文件。压缩溢出文件可能有助于避免IO操作导致磁盘子系统过载。

配置资源队列

Greenplum数据库的资源队列为管理集群负载提供了一种强有力的机制。队列可以被用来限制活动查询的数量以及队列中查询可使用的内存量。当查询被提交给Greenplum数据库时，它会被加入一个资源队列，资源队列会决定该查询是否应该被接受并且何时有资源可用来执行它。

- 为所有用户分配预定义好的资源队列。

每个登录用户（角色）都被关联到单个资源队列，任何该用户提交的查询都由关联的资源队列处理。如果没有为用户的查询明确地分配一个队列，则会由默认队列`pg_default`处理。

- 不要用`gpadmin`角色或者其他超级用户角色运行查询。

超级用户会被从资源队列限制中排除，因此超级用户的查询运行不会考虑在其所属队列上设置的限制。

- 使用`ACTIVE_STATEMENTS`资源队列参数来限制特定队列成员能够并发运行的活动查询数量。

• 使用`MEMORY_LIMIT`参数控制通过队列运行的查询可以利用的内存总量。通过组合`ACTIVE_STATEMENTS`和`MEMORY_LIMIT`属性，管理员可以完全控制从一个给定资源队列发出的活动。

分配按如下方式进行：假定资源队列`sample_queue`的`ACTIVE_STATEMENTS`被设置为10，而`MEMORY_LIMIT`被设置为2000MB。这限制该队列在每个segment上使用大约2GB内存。对于每台服务器有8个segment的集群，`sample_queue`在每台服务器上的总用量是16GB（ $2\text{GB} * 8\text{ Segment/服务器}$ ）。如果segment服务器有64GB内存，系统中可以有不超过四个这种资源队列，再多就会内存不足（ $4\text{队列} * 16\text{GB/队列}$ ）。

注意通过使用STATEMENT_MEM，运行在队列中的个体查询能够分配超过其内存“份额”的内存，从而降低队列中其他查询可用的内存。

- 资源队列优先级可以被用来排列具有预期结果的负载。带有MAX优先权的队列会扼杀所有其他队列中的活动，直至MAX队列完成所有查询的运行。
- 根据负载和现状动态修改资源队列以匹配队列的实际需求。用户可以把这些更改写成脚本并且增加 crontab项来执行这些脚本。
- 使用gptoolkit查看资源队列使用以及理解队列如何工作。

Parent topic: [Greenplum数据库最佳实践](#)

Greenplum数据库® 6.0文档

□ Greenplum数据库最佳实践

最佳实践概要

系统配置

模式设计

采用资源组管理内存和资源

采用资源队列管理内存和资源

□ 系统监控和维护

装载数据

安全性

加密数据和数据库连接

SQL查询调优

高可用性

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

本节介绍日常运维相关的最佳实践，关注这些内容可以确保Greenplum数据库日常高可用性和性能保持最佳状态。

- [用ANALYZE更新统计信息](#)
- [管理数据库膨胀](#)
- [监控Greenplum数据库日志文件](#)

Parent topic: [Greenplum数据库最佳实践](#)

监控

Greenplum数据库提供了一些对监控系统非常有用的工具。

`gp_toolkit`模式包含多个可以用SQL命令访问的视图，通过它们可以查询系统目录、日志文件以及操作环境来获得系统状态信息。

`gp_stats_missing`视图展示没有统计信息且要求运行ANALYZE的表。

更多`gpstate`和`gpcheckperf`工具的信息请参考*Greenplum数据库工具指南*。有关`gp_toolkit`模式的信息，请见*Greenplum数据库参考指南*。

gpstate

`gpstate`工具显示Greenplum系统的状态，包括哪些segment宕掉了、saster和segment的配置信息（hosts、数据目录等）、系统使用的端口以及主segment与它们对应的镜像segment之间的映射。

运行`gpstate -Q`可以得到一个segment的列表，这个列表列出了那些在master的系统目录中被标记为"down"的segment。

要得到Greenplum系统的详细状态信息，可运行`gpstate -s`。

gpcheckperf

`gpcheckperf`工具能用来测试主机硬件的基线性能。其结果可以帮助您识别可能影响系统性能的因素。



助发现硬件问题。它会执行下列检查：

- 磁盘I/O测试 – 通过使用操作系统命令dd读写一个大型文件来测量I/O性能。它报告以兆字节每秒为单位的读写速率。
- 内存带宽测试 – 使用STREAM基准测量以兆字节每秒为单位的可持续的内存带宽。
- 网络性能测试 – 运行gpnetbench网络基准程序（也可以选netperf）来测试网络性能。这种测试可以运行在三种模式中：并行结对测试（-r N）、串行结对测试（-r n）或者全矩阵测试（-r M）。报告的最小、最大、平均和中值传输率将以兆字节每秒为单位。

为了从gpcheckperf获得有效的数字，数据库系统必须被停止。即使系统仍在运行且没有运行查询活动，从gpcheckperf得到的数字可能也不准确。

gpcheckperf要求在参与性能测试的主机之间提前进行访问互信设置。因为工具会调用gpssh以及gpscp，因此这些工具必须也位于用户的PATH中。可以个别指定要检查的主机（-h host1 -h host2 ...）或者使用-f hosts_file，其中hosts_file是一个包含要检查的主机列表的文本文件。如果用户有多个子网，为每个子网都创建一个单独的主机文件，这样用户可以单独测试子网。

gpcheckperf默认会运行磁盘I/O测试、内存测试和串行结对网络性能测试。对于磁盘I/O测试，用户必须使用-d选项指定要测试的文件系统。下面的命令在subnet_1_hosts文件中列出的主机上测试磁盘I/O和内存带宽：

```
$ gpcheckperf -f subnet_1_hosts -d /data1 -d /data2 -r ds
```

-r选项选择要运行的测试：磁盘I/O（d）、内存带宽（s）、网络并行结对（n）、网络串行结对测试（n）、网络全矩阵测试（m）。每次执行只能选择一种网络模式。详细的gpcheckperf参考信息请见Greenplum数据库参考指南。

用操作系统工具监控

下面的Linux/UNIX工具可以被用来评估主机性能：

- iostat 允许用户监控segment主机上的磁盘活动。
- top 显示操作系统进程的动态视图。
- vmstat 显示内存使用统计信息。

用户可以使用gpssh在多台主机上运行相关工具。

最佳实践

- 实现*Greenplum*数据库管理员指南中的“推荐的监控和维护任务”。
- 在安装时运行`gpcheckperf`，并在安装之后定期运行它，将其输出保存起来以对比不同时刻的系统性能。
- 使用所有能支配的工具来理解系统在不同负载下的行为。
- 检查异常事件以确定原因。
- 通过定期运行执行计划监控系统上的查询活动，以确保查询以最优的方式运行。
- 检查执行计划以确定索引是否被使用以及分区裁剪是否按照预期发生。

附加信息

- `gpcheckperf`详细信息请见*Greenplum*数据库工具指南。
- "推荐的监控和日常运维任务"部分详情请柬*Greenplum*数据库管理员指南。
- [Sustainable Memory Bandwidth in Current High Performance Computers](#) . John D. McCalpin. Oct 12, 1995.
- 请参考www.netperf.org 来使用netperf, netperf必须被安装在所有测试机器上。更多有关`gpcheckperf`的信息请移步命令详细出处。

Greenplum数据库® 6.0文档

□ Greenplum数据库最佳实践

最佳实践概要

系统配置

模式设计

采用资源组管理内存和资源

采用资源队列管理内存和资源

□ 系统监控和维护

装载数据

安全性

加密数据和数据库连接

SQL查询调优

高可用性

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

[如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息](#)

本节描述了将数据装载到Greenplum数据库的不同方式。

Parent topic: [Greenplum数据库最佳实践](#)

带列值的INSERT语句

带有值的单个INSERT语句会向表中加入一行。这个行会流过master并且被分布到一个segment上。这是最慢的方法并且不适合装载大量数据。

COPY语句

PostgreSQL的COPY语句从外部文件拷贝数据到数据表中。它比INSERT语句插入多行的效率更高，但是行仍需流过master。所有数据都在一个命令中被拷贝，它并不是一种并行处理。

COPY命令的数据输入来自于一个文件或者标准输入。例如：

```
COPY table FROM '/data/mydata.csv' WITH CSV HEADER;
```

使用COPY适合于增加相对较小的数据集合（例如多达上万行的维度表）或者一次性数据装载。

在编写脚本处理装载少于1万行的少量数据时使用COPY。

因为COPY是一个单一命令，在使用这种方法填充表时没有必要禁用自动提交。

使用者可以运行多个并发的COPY命令以提高性能。

外部表

外部表提供了对Greenplum数据库之外的数据来源的访问。可以用SELECT语句访问它们，外部表通常被用于抽取、转换、装载（ETL）模式，这是一种抽取、转换、装载（ETL）模式的变种，这种模式可以利用Greenplum数据库的快速并行数据装载能力。

通过ETL，数据被从其来源抽取，在数据库外部使用外部转换工具（Informatica或者Datastage）转换，然后被装载到数据库中。

通过ELT，Greenplum外部表提供对外部来源中数据的访问，外部来源可以是只读文件（例如文本、CSV或者XML文件）、Web服务器、Hadoop文件系统、可执行的OS程序或者Greenplum gpfdist文件服务器，这些在下一节中描述。外部表支持选择、排序和连接这样的SQL操作，这样数据可以被同时装载和转换，或者被装载到一个装载表并且在数据库内被转换成目标表。

外部表使用CREATE EXTERNAL TABLE语句定义，该语句有一个LOCATION子句定义数据的位置以及一个FORMAT子句定义源数据的格式，这样系统才能够解析输入数据。文件使用file://协议，并且文件必须位于一台segment主机上由Greenplum超级用户可访问的位置。数据可以被分散在segment主机上，并且每台主机上的每个主segment有不超过一个文件。LOCATION子句中列出的文件的数量是将并行读取该外部表的segment的数量。

使用Gpf dist外部表

装载大型事实表的最快方式是使用基于gpdist的外部表。gpdist是一个使用HTTP协议的文件服务器程序，它以并行的方式向Greenplum数据库的segment供应外部数据文件。一个gpdist实例每秒能供应200MB并且很多gpdist进程可以同时运行，每一个供应要被装载的数据的一部分。当使用者用INSERT INTO <table> SELECT * FROM <external_table>这样的语句开始装载时，INSERT语句会被master解析并且分布给主segment。segment连接到gpdist服务器并且并行检索数据，解析并验证数据，从分布键数据计算一个哈希值并且基于哈希键把行发送给它的目标segment。每个gpdist实例默认将接受最多64个来自segment的连接。通过让更多的segment和gpdist服务器参与到装载处理中，可以以非常高的速度进行数据装载。

在使用gpdist数量达到配置参数gp_external_max_segments的最大值时，主segment会并行访问外部文件。在优化gpdist的性能时，随着segment的数量增加会最大化并行性。在尽可能多的ETL节点上均匀地散布数据。将非常大型的数据文件分解成相等的部分，并且把数据分散在尽可能多的文件系统上。

在每个文件系统上运行两个gpdist实例。在装载数据时，gpdist在segment节点上容易变成CPU密集型的操作。举个例子，如果有八个机架的segment节点，在segment上就有大量可用的CPU来驱动更多的gpdist进程。在尽可能多的接口上运行gpdist。要注意绑定

网卡并且确保启动足够的gpfdist实例配合它们一起工作。

有必要在所有这些资源上保持工作处于均衡状态。装载的速度与最慢的节点相同。装载文件布局上的倾斜将导致整体装载受制于资源瓶颈。

gp_external_max_segs配置参数控制每个gpfdist进程能服务的segment数量。默认值是64。使用者可以在saster上的postgresql.conf配置文件中设置一个不同的值。总是保持gp_external_max_segs和gpfdist进程的数量为一个偶因子，也就是说gp_external_max_segs值应该是gpfdist进程数的倍数。例如，如果有12个segment和4个gpfdist进程，规划器会按照下面的方式循环分配segment连接：

```
Segment 1 - gpfdist 1
Segment 2 - gpfdist 2
Segment 3 - gpfdist 3
Segment 4 - gpfdist 4
Segment 5 - gpfdist 1
Segment 6 - gpfdist 2
Segment 7 - gpfdist 3
Segment 8 - gpfdist 4
Segment 9 - gpfdist 1
Segment 10 - gpfdist 2
Segment 11 - gpfdist 3
Segment 12 - gpfdist 4
```

在装载到已有表之前删除索引，并且在装载之后重建索引。在装载完数据后重新创建索引比装载每行时增量更新索引更快。

装载后在表上运行ANALYZE。在装载期间通过设置gp_autostats_mode为NONE来禁用自动统计信息收集。在装载出错后运行VACUUM来回收空间。

对重度分区的列存表执行少量高频的数据装载可能会对系统有很大影响，因为在每个时间间隔内被访问的物理文件会很多。

Gload

gload是一种数据装载工具，它扮演着Greenplum外部表并行装载特性的接口的角色。

要当心对gload的使用，因为它会创建并且删除外部表，从而可能会导致系统目录膨胀。可转而使用gpfdist，因为它能提供最好的性能。

`gpload` 使用定义在一个 YAML 格式的控制文件中的规范来执行一次装载。它会执行下列操作：

- 调用 `gpfdist` 进程
- 基于定义的源数据创建一个临时的外部表定义
- 执行 `INSERT`、`UPDATE` 或者 `MERGE` 操作 将源数据载入数据库中的目标表
- 删除临时外部表
- 清除 `gpfdist` 进程

装载会在单个事务中完成。

最佳实践

- 在装载数据之前删掉现有表上的任何索引，并且在装载之后重建那些索引。新创建索引比装载每行时 增量更新索引更快。
- 在装载期间通过将 `gp_autostats_mode` 配置参数设置为 `NONE` 禁用自动统计信息收集。
- 外部表并非为频繁访问或者 `ad hoc` 访问而设计。
- 外部表没有统计信息来告知优化器。可以用下面这样的语句在 `pg_class` 系统目录中为 外部表设置粗略的行数和磁盘页数估计：

```
UPDATE pg_class SET reltuples=400000, relpages=400  
WHERE relname='myexttable';
```

- 在使用 `gpfdist` 时，通过为 ETL 服务器上的每一块 NIC 运行一个 `gpfdist` 实例以最大化网络带宽。在 `gpfdist` 实例之间均匀地划分源数据。
- 在使用 `gpload` 时，在资源允许的情况下同时运行尽可能多的 `gpload` 实例。利用可用的 CPU、内存和网络资源以增加能从 ETL 服务器传输到 Greenplum 数据库的数据量。
- 使用 `COPY` 语句的 `SEGMENT REJECT LIMIT` 子句设置在 `COPY` `FROM` 命令被中止之前可以出现错误的行的百分数限制。这个拒绝限制是针对每个 segment 的，当任意一个 segment 超过该限制时，命令将被中止且不会有行被增加。使用 `LOG ERRORS` 子句可以保存错误行。如果有一行在格式上有错误—例如缺少值或者有多余的值，或者数据类型不对—Greenplum 数据库会在内部存储错误信息和行。使用内建 SQL 函数 `gp_read_error_log()` 可以访问这种存储下来的信息。

- 如果装载出现错误，在该表上运行VACUUM以恢复空间。
- 在用户装载数据到表中后，在堆表（包括系统目录）上运行VACUUM，并且在所有的表上运行ANALYZE。没有必要在追加优化表上运行VACUUM。如果表已经被分区，用户可以只清理和分析受数据装载影响的分区。这些步骤会清除来自于被中止的装载、删除或者更新中的行并且为表更新统计信息。
- 在装载大量数据之后重新检查表中的segment倾斜。用户可以使用下面这样的查询来检查倾斜：

```
SELECT gp_segment_id, count(*)
FROM schema.table
GROUP BY gp_segment_id ORDER BY 2;
```

- gpfdist默认假定最大记录尺寸为32K。要装载大于32K的数据记录，用户必须通过在gpfdist命令行上指定-m <bytes>选项来增加最大行尺寸参数。如果用户使用的是gupload，在gupload控制文件中设置MAX_LINE_LENGTH参数。
Note: 与Informatica Power Exchange的集成当前被限制为默认的32K记录长度。

额外信息

使用gpfdist和gupload装载数据的详细指导步骤请见 *Greenplum数据库参考指南*。

Greenplum数据库® 6.0文档

□ Greenplum数据库最佳实践

最佳实践概要

系统配置

模式设计

采用资源组管理内存和资源

采用资源队列管理内存和资源

□ 系统监控和维护

装载数据

安全性

加密数据和数据库连接

SQL查询调优

高可用性

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

最佳实践可以确保最高级别的系统安全性。

基础安全最佳实践

- 保护好`gpadmin`系统用户。Greenplum要求一个UNIX用户ID来安装和初始化Greenplum 数据库系统。这个系统用户在Greenplum文档中被称作`gpadmin`。`gpadmin` 用户是Greenplum数据库中的默认数据库超级用户，也是Greenplum安装及其底层数据文件的文件系统拥有者。默认的管理员账户是Greenplum数据库设计的根本。没有它系统无法运行，并且也没有办法限制`gpadmin` 用户ID的访问。这个`gpadmin`用户可以绕过Greenplum数据库的所有安全性特性。任何人通过这一用户ID登入到Greenplum主机，就可以读取、修改或者删除任何数据，包括系统目录数据和数据库访问权限。因此，非常有必要保护好`gpadmin`用户ID并且只允许必要的系统管理员可以接触到它。只有在执行特定系统维护任务（例如升级或扩展）时，管理员才应该作为`gpadmin`登入到Greenplum。数据库用户绝不应作为`gpadmin`登录，并且也绝不应以`gpadmin`运行ETL或者生产负载。
- 为每个登入的用户分配一个不同的角色。为了日志和审计目的，每个被允许登入Greenplum数据库的用户应该被给定其自己的数据库角色。对于应用或者Web服务，考虑为每种应用或者服务创建一个不同的角色。详情请见*Greenplum*数据库管理员指南中的“创建新角色（用户）”部分。
- 使用组来管理访问特权。详情请见*Greenplum*数据库管理员指南中的“创建组（角色的成员关系）”部分。
- 限制拥有SUPERUSER角色属性的用户。作为超级用户的角色会绕过Greenplum数据库中的所有访问特权检查，也会绕过资源队列。只有系统管理员才应该被给予超级用户权利。详情请见*Greenplum*数据库管理员指南中的“修改角色属性”部分。

口令强度指导

为了保护网络不受侵入，系统管理员应该验证组织中使用的口令是强口令。下面的建议可以增强口令：

- 最小口令强度推荐：至少9个字符。MD5口令应该为至少15个字符。
- 混合大小写字母。

- 混合字母和数字。
- 包括非字母数字字符。
- 选择一个用户可以记住的口令。

下面推荐了一些可以用来确定口令强度的口令破解软件。

- John The Ripper。一种快速灵活的口令破解程序。它允许使用多个单词列表并且可以进行蛮力口令破解。该程序可以从<http://www.openwall.com/john/> 得到。
- Crack。可能是最著名的口令破解软件，Crack也非常快，但是可能不如John The Ripper那么易用。该软件可以在<http://www.crypticide.com/alecm/security/crack/c50-faq.html> 得到。

整个系统的安全性依赖于root口令的强度。该口令应该至少长达12个字符并且包括大写字母、小写字母、特殊字符 和数字的组合。它不能基于任何词典中的词。

应该配置口令过期参数。

确保下面的行存在于文件/etc/libuser.conf的[import]小节中。

```
login_defs = /etc/login.defs
```

确保在[userdefaults]小节中没有以下列文本开头的行，因为这些词会覆盖来自 /etc/login.defs的设置：

- LU_SHADOWMAX
- LU_SHADOWMIN
- LU_SHADOWWARNING

确保下面的命令不会产生输出。通过这一命令列出的任何账号都应该被锁定。

```
grep "^\+:" /etc/passwd /etc/shadow /etc/group
```

注意：我们强烈推荐客户在初始设置后更改他们的口令。

```
cd /etc
chown root:root passwd shadow group gshadow
chmod 644 passwd group
chmod 400 shadow gshadow
```

找出所有全域可写的文件以及没有设置其粘滞位的文件。

```
find / -xdev -type d \(\ -perm -0002 -a ! -perm -1000 \) -print
```

为前一个命令结果中的所有目录设置其粘滞位(# chmod +t {dir})。

找出所有全域可写的文件并且修正每一个被列出的文件。

```
find / -xdev -type f -perm -0002 -print
```

为前述命令给出的所有文件设置正确的权限(# chmod o-w {file})。

找出所有不属于有效用户或组的文件，然后为它们分配一个拥有者或者移除文件。

```
find / -xdev \(\ -nouser -o -nogroup \) -print
```

找出所有全域可写的目录并且确认它们属于root或者某个系统账户（假定只有系统账户的用户ID低于500）。如果该命令生成输出，验证其分配是否正确或者将它们重新分配给root。

```
find / -xdev -type d -perm -0002 -uid +500 -print
```

口令质量、口令过期策略、口令重用、口令重试尝试以及更多认证设置可以通过可插拔认证模块 (PAM) 框架配置。PAM在目录/etc/pam.d中查找应用相关的配置信息。运行authconfig 或者system-config-authentication将重写PAM配置文件，这会毁掉任何手工更改并且将它们替换为系统默认配置。

默认的PAM模块pam_cracklib提供了口令的强度检查。要配置pam_cracklib以要求至少一个大写字符、小写字符、数字和特殊字符 (U.S.国防部指导方针推荐)，可编辑文件/etc/pam.d/system-auth 并且在对应于口令前置条件pam_cracklib.so try_first_pass的行中包括下列参数。

```
retry=3:  
dcredit=-1. Require at least one digit  
ucredit=-1. Require at least one upper case character  
ocredit=-1. Require at least one special character  
lcredit=-1. Require at least one lower case character  
minlen=14. Require a minimum password length of 14.
```

例如：

```
password required pam_cracklib.so try_first_pass
retry=3\minlen=14 dcredit=-1 ucredit=-1 ocredit=-1
lcredit=-1
```

可以设置这些参数来反映用户的安全性策略需求。注意口令限制不适用于root口令。

PAM模块pam_tally2提供了在指定失次数的败登录尝试之后锁住用户账户的功能。要实施口令封锁，可编辑文件/etc/pam.d/system-auth来包括下列行：

- 第一个认证行应该包括：

```
auth required pam_tally2.so deny=5 onerr=fail
unlock_time=900
```

- 第一个账户行应该包括：

```
account required pam_tally2.so
```

这里，deny参数被设置以限制重试次数为5并且unlock_time已经被设置为900秒来保持账户在被解锁前锁定900秒。请配置这些参数以反映用户的安全性策略需求。被锁定的账户可以用pam_tally2工具手工解锁：

```
/sbin/pam_tally2 --user {username} -reset
```

用户可以使用PAM限制重用最近用过的口令。可以设置pam_unix模块的remember选项来记住最近的口令并且阻止重用它们。要做到这一点，可在/etc/pam.d/system-auth中编辑适当的行来包括remember选项。

例如：

```
password sufficient pam_unix.so [ ... existing_options ... ]
remember=5
```

用户可以设置要记住的历史口令的数量以正确地反映其安全性策略需求。

```
cd /etc
chown root:root passwd shadow group gshadow
chmod 644 passwd group
chmod 400 shadow gshadow
```

Parent topic: [Greenplum数据库最佳实践](#)

Greenplum数据库® 6.0文档

□ Greenplum数据库最佳实践

最佳实践概要

系统配置

模式设计

采用资源组管理内存和资源

采用资源队列管理内存和资源

□ 系统监控和维护

装载数据

安全性

加密数据和数据库连接

SQL查询调优

高可用性

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

本节描述了有关实现加密和管理密钥的最佳实践。

在Greenplum数据库系统中可以用下面的方式使用加密来保护数据：

- 客户端和Master数据库之间的连接可以用SSL加密。这种方式可以通过设置ssl服务器配置参数为on并且配置好pg_hba.conf文件来启用。有关在Greenplum数据库中启用SSL的信息请见Greenplum数据库管理员指南中的“加密客户端/服务器连接”部分。
- Greenplum数据库4.2.1及以上的版本允许在Greenplum的并行文件分发服务器、gpfdist 和segment主机之间传输SSL加密数据。详见[加密gpfdist连接](#)。
- Greenplum数据库集群中主机之间的网络通信可以使用IPsec加密。集群中的每一对主机之间会建立一个认证过的加密的VPN。对IPsec的支持请检查操作系统文档，或者考虑[Zettaset](#) 等组织提供的第三方解决方案。
- pgcrypto包中的加密/解密函数保护停留在数据库中的数据。列级的加密可以保护敏感信息，例如口令、社会保险号码或者信用卡号码。例子可以在[使用PGP加密表中的数据](#)中找到。

最佳实践

- 加密确保数据只能被拥有解密数据所需密钥的用户看见。
- 加密和解密数据有性能代价，只加密需要加密的数据。
- 在生产系统中实现任何加密解决方案之前先做性能测试。
- 用于生产的Greenplum数据库系统中的服务器证书应该由一个数字证书认证机构 (CA) 签发，这样客户端可以认证服务器。如果所有的客户端都在该组织本地，这个CA可以是本地的。
- 只要到Greenplum数据库的客户端连接会通过一个不安全的链路，就应该使用SSL加密。
- 对称加密模式（加密和解密使用相同的密钥）比非对称模式具有更好的性能，并且应该在可以安全共享密钥时使用对称模式。
- 使用来自pgcrypto包的函数来加密磁盘上的数据。数据在数据库进程中被加密和解密，因此有必要用SSL来保护客户端以避免传输未加密的数据。
- 在ETL数据被载入数据库或者被从数据库中卸载时，使用gpfdists协议来保护ETL数据。详情请见[加密gpfdist连接](#)。

密钥管理

只要使用对称（单私钥）或者非对称（公钥和私钥）加密，就有必要安全地存储主密钥或者私钥。存储加密密钥有很多选项，例如在文件系统上保存、密钥保管库、加密的USB、可信平台模块（TPM）或者硬件安全模块（HSM）。

在规划密钥管理时考虑下列问题：

- 密钥将被存在哪里？
- 密钥何时过期？
- 如何保护密钥？
- 如何访问密钥？
- 如何恢复和收回密钥？

开放Web应用安全性项目（OWASP）提供了一套非常全面的[保护加密密钥指南](#)。

用pgcrypto加密静止数据

Greenplum数据库的pgcrypto包提供了加密数据库中静止数据的函数。管理员可以加密具有敏感信息（例如社会保险号码或信用卡号）的列以提供一个额外的保护层。没有加密密钥的用户无法读取以加密形式存储的数据库数据，并且这些数据也无法从磁盘直接读取。

pgcrypto在Greenplum数据库安装时默认已经安装。客户只需要在想要使用的数据库中启用该组件即可。

pgcrypto允许使用对称和非对称加密的PGP加密。对称加密使用同样的密钥加密和解密数据，并且比非对称加密更快。在交换密钥不成问题的环境中它是首选方法。在非对称加密中，公钥被用来加密数据而私钥被用来解密数据。这种模式比对称加密慢一些并且要求更强的密钥。

使用pgcrypto总是会带来性能和可维护性的代价。有必要只对需要加密的数据使用加密。还有，要记住用户无法通过对数据增加索引来搜索加密数据。

在用户实现数据库内加密之前，考虑下列PGP限制。

- 不支持签名。这还意味着不会检查加密子密钥是否属于主密钥。

- 不支持将加密密钥作为主密钥。这种做法通常是不被鼓励的，因此这一限制应该不是问题。
- 不支持多个子密钥。这可能看起来像一个问题，因为这是一种常见的做法。在另一方面，用户不应将其常规GPG/PGP密钥用于pgcrypto，而是要创建新的密钥，因为使用场景不同。

Greenplum数据库默认用zlib编译，这允许PGP加密函数在加密数据之前先压缩数据。在编译有OpenSSL时，将会有更多算法可用。

因为pgcrypto函数运行在数据库服务器内部，数据和口令是以明文形式在pgcrypto和客户端应用之间移动。为了最好的安全性，用户应该使用本地连接或者使用SSL连接，并且用户应该信任系统管理员和数据库管理员。

pgcrypto会根据主PostgreSQL配置脚本配置自身。

当编译有zlib时，pgcrypto加密函数能在加密前压缩数据。

pgcrypto拥有从基本内建函数到高级内建函数的多个加密级别。下面的表格展示了支持的加密算法。

Table 1. Pgcrypto支持的加密函数

功能值	内建	带有OpenSSL
MD5	yes	yes
SHA1	yes	yes
SHA224/256/384/512	yes	yes ¹ 。
其他摘要算法	no	yes ²
Blowfish	yes	yes
AES	yes	yes ³
DES/3DES/CAST5	no	yes
Raw Encryption	yes	yes
PGP Symmetric-Key	yes	yes
PGP Public Key	yes	yes

创建PGP密钥

要在Greenplum数据库中使用PGP非对称加密，用户必须首先创建公私钥并且安装它们。

这一节假定用户正在Linux机器上用Gnu Privacy Guard (gpg) 命令行工具 安装Greenplum数据库。Pivotal推荐使用最新版本的GPG来创建密钥。可以从 <https://www.gnupg.org/download/> 为用户的操作系统下载并安装Gnu Privacy Guard (GPG)。在GnuPG网站上，用户将找到用于常见Linux发行的安装器以及Windows和Mac OS X安装器的链接。

1. 作为root，执行下列命令并且从菜单选择选项1：

```
# gpg --gen-key
gpg (GnuPG) 2.0.14; Copyright (C) 2009 Free Software
Foundation, Inc.
This is free software: you are free to change and
redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: directory `/root/.gnupg' created
gpg: new configuration file `/root/.gnupg/gpg.conf'
created
gpg: WARNING: options in `/root/.gnupg/gpg.conf' are not
yet active during this run
gpg: keyring `/root/.gnupg/secring.gpg' created
gpg: keyring `/root/.gnupg/pubring.gpg' created
Please select what kind of key you want:
(1) RSA and RSA (default)
(2) DSA and Elgamal
(3) DSA (sign only)
(4) RSA (sign only)
Your selection? 1
```

2. 如下例所示，回应提示并且遵循其指导：

```
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) Press enter to accept
default key size
Requested keysize is 2048 bits
Please specify how long the key should be valid.
 0 = key does not expire
<n> = key expires in n days
<n>w = key expires in n weeks
<n>m = key expires in n months
<n>y = key expires in n years
Key is valid for? (0) 365
Key expires at Wed 13 Jan 2016 10:35:39 AM PST
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: John Doe
Email address: jdoe@email.com
Comment:
You selected this USER-ID:
"John Doe <jdoe@email.com>"
```

```

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
You need a Passphrase to protect your secret key.
(该演示例子的密码为空。)
can't connect to `/root/.gnupg/S.gpg-agent': No such
file or directory
You don't want a passphrase - this is probably a *bad*
idea!
I will do it anyway. You can change your passphrase at
any time,
using this program with the option "--edit-key".

We need to generate a lot of random bytes. It is a good
idea to perform
some other action (type on the keyboard, move the mouse,
utilize the
disks) during the prime generation; this gives the
random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good
idea to perform
some other action (type on the keyboard, move the mouse,
utilize the
disks) during the prime generation; this gives the
random number
generator a better chance to gain enough entropy.
gpg: /root/.gnupg/trustdb.gpg: trustdb created
gpg: key 2027CC30 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdbgpg:
      3 marginal(s) needed, 1 complete(s) needed, PGP
trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q,
0n, 0m, 0f, 1u
gpg: next trustdb check due at 2016-01-13
pub   2048R/2027CC30 2015-01-13 [expires: 2016-01-13]
      Key fingerprint = 7EDA 6AD0 F5E0 400F 4D45  3259
      077D 725E 2027 CC30
uid            John Doe <jdoe@email.com>
sub   2048R/4FD2EFBB 2015-01-13 [expires: 2016-01-13]

```

3. 通过输入下列命令来列出PGP密钥：

```

gpg --list-secret-keys
/root/.gnupg/secring.gpg
-----
sec   2048R/2027CC30 2015-01-13 [expires: 2016-01-13]
uid            John Doe <jdoe@email.com>
ssb   2048R/4FD2EFBB 2015-01-13

```

2027CC30是公钥并且将被用来加密数据库中的数据。4FD2EFBB是私钥并且将被用来解密数据。

4. 使用下列命令导出密钥:

```
# gpg -a --export 4FD2EFBB > public.key
# gpg -a --export-secret-keys 2027CC30 > secret.key
```

更多有关PGP加密函数的信息请见[pgcrypto](#)。

使用PGP加密表中的数据

这个小节展示如何使用用户产生的PGP密钥加密插入到一列中的数据。

1. 转储public.key文件的内容，然后把它拷贝的剪贴板：

```
# cat public.key
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.14 (GNU/Linux)

mQENBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jt
2HislojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgK1GSgd9texg2nnSL9Adm
R5syVKG+qcdWuvyZg9oO0meyjhc3n+kkbRTEMuM3f1bMs8shOwzMvstCUV
.
.
.
WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8Hkk
HMUC55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNlGWE
/UUZB+dYqCwtvX0nnBu1KNcmk2AkEcFK3YoliCxomdOxhFOv9AKjjojDyC
Pv2MikPS2fKOAg1R3LpMa8zDETl4w3vckPQNrQNnYuUtfj6ZoCxv
=XZ8J
-----END PGP PUBLIC KEY BLOCK-----
```

2. 创建一个名为userssn的表并且插入一些敏感数据，这个例子中是Bob和Alice的社会保险号码。在"dearmor("之后粘贴public.key的内容。

```
CREATE TABLE userssn( ssn_id SERIAL PRIMARY KEY,
    username varchar(100), ssn bytea);

INSERT INTO userssn(username, ssn)
SELECT robotccs.username, pgp_pub_encrypt(robotccs.ssn,
keys.pubkey) AS ssn
FROM (
    VALUES ('Alice', '123-45-6788'), ('Bob', '123-
```

45-6799'))
AS robotccs(username, ssn)
CROSS JOIN (SELECT dearmor('-----BEGIN PGP PUBLIC KEY
BLOCK-----
Version: GnuPG v2.0.14 (GNU/Linux)

mQENBFS1zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jt

2His1ojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgK1GSgd9texg2nnSL9Adm

R5syVKG+qcdWuvyZg9oO0meyjhc3n+kkbRTEMuM3f1bMs8shOwzMvstCUV

.
.
.
WH+N21asoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8Hkk

HMUC55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNlGWE

/UUZB+dYqCwtvX0nnBu1KNcmk2AkEcFK3YoLiCxomdOxhFOv9AKjjojDyC

Pv2MikPS2fKOAg1R3LpMa8zDETl4w3vckPQNrnQNyUUtfj6ZoCvx
=XZ8J
-----END PGP PUBLIC KEY BLOCK-----' AS pubkey) AS keys;

3. 验证ssn列被加密。

```
test_db=# select * from userssn;
ssn_id      | 1
username    | Alice
ssn         |
\301\300L\003\235M%_O\322\357\273\001\010\000\272\227\010\
[\227\034\313:c\354d<\337\006Q\351( '\2330\0311X\263Qf\341\
7u\270*\304\361\355\220\021\330"\200%\264\274}R\213\377\36
f
\015\004\242\231\263\225%\032\271a\001\035\277\021\375X\23
37-
\251\336\303\340\377_\011\275\301/MY\334\343\245\244\372y\
017fi\226Q\307\012\326\3646\000\326\005:E\364W\252=zz\010(
315G^\027:K_9\254\362\354\215<\001\304\357\331\355\323,\30
(\\\373
4\254\230\331\356\006B\257\333\326H\022\013\353\216F?
\023\220\370\035vH5/\227\344b\322\227\026\362=
42\033\322<\001}\243\224; )\030zqX\214\340\221\035\275U\345
011\214\307\227\237\270\026`R\205\205a~1\263\236[\037C\260
```

ssn_id	2
username	Bob
ssn	\301\300L\003\235M%_O\322\357\273\001\007\377t>\345\343,\2
L[v\262k\244\2435\264\232B\357\370d9\375\011\002\327\235<\7`\012c\353]\355d7\360T\335\314\367\370;x\371\350*\231\212	\000\370\370\366\013\022\357\005i\202~\005\\z\301o\012\23
\213\032\226\$\2751\256XR\346k\266\030\234\267\201vUh\004\2	20\316\306 \203+\010\261;\232\254tp\255\243\261\373Rq;\316
\322\347ea\220\0151\212g\337\264\336b\263\004\311\210.4\34	12\342y^>\202\262 A7\202t\240\333p\345G\373\253\243oCO\011\
\347\240\005\213\0078\036\210\307\$\\317\322\311\222\035\354	3270\013c\327\272\212%\363\033\252\322\337\354\276\225\232

4. 从数据库提取public.key ID

```
SELECT pgp_key_id(dearmor('-----BEGIN PGP PUBLIC KEY
BLOCK-----
Version: GnuPG v2.0.14 (GNU/Linux)

mQENBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jt
2HislojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgK1GSgd9texg2nnSL9Adm
R5syVKG+qcdWuvyZg9oO0meyjhc3n+kkbRTEMuM3flbMs8sh0wzMvstCUV
.
.
.
WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8Hkk
HMUC55H0g2qQAY0BpnJHg00045Q6pk3G2/7Dbek5WJ6K1wUrFy51sN1GWE
```

```
/UUZB+dYqCwtvX0nnBu1KNMmk2AkEcFK3YoliCxomdOxhFOv9AKjjojDyC
Pv2MikPS2fKOAg1R3LpMa8zDETl4w3vcKPNrQNnYuUtfj6ZoCxv
=XZ8J
-----END PGP PUBLIC KEY BLOCK-----' ));
```

pgp_key_id | 9D4D255F4FD2EFBB

这会显示用来加密ssn列使用的PGP键ID是9D4D255F4FD2EFBB。只要有新密钥被创建，都推荐执行这一步，然后将该ID保存下来用于跟踪。
用户可以使用这个键来查看哪一个密钥对被用来加密数据：

```
SELECT username, pgp_key_id(ssn) AS key_used
FROM userssn;
username | Bob
key_used | 9D4D255F4FD2EFBB
-----+-----
username | Alice
key_used | 9D4D255F4FD2EFBB
```

Note: 不同的密钥可能具有相同的ID。这很少见，但属于正常现象。客户端应用应该尝试用每一个来解密看看哪一个合适—就像处理ANYKEY一样。详情请见pgcrypto文档中的[pgp_key_id\(\)](#) 部分。

5. 使用私钥解密数据。

```
SELECT username, pgp_pub_decrypt(ssn, keys.privkey)
      AS decrypted_ssn FROM userssn
  CROSS JOIN
  (SELECT dearmon('-----BEGIN PGP PRIVATE
KEY BLOCK-----'
Version: GnuPG v2.0.14 (GNU/Linux)

1QOYBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jt
2HislojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgK1GSgd9texg2nnSL9Adm
R5syVKG+qcdWuvyZg9oO0meyjhC3n+kkbRTEMuM3f1bMs8sh0wzMvstCUV
vG5rJAe8PuYDSJCJ74I6w7SOH3RiRIC7IfL6xYddV4213ctd44bl8/i71h
/Hbsji2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFfnFnE6
ic8MD0lK5/GAlR6Hc0ZIHf8KEcavruQlikjnABEBAAEAB/wNfjjvP1brRf
XwUNm+sI4v2Ur7qZC94VTukPGf671vqcYZJuqXxvZrZ8bl6mv165xEUiZY
fe0PaM4Wy+Xr94Cz2bPbWgawnRNN3GAQy4rlBTrvqQWy+kmpbd87iTjwZi
02iSzraqq41Rt0Zx21Jh4rkpF67ftmzOH0vlrS0bW0vHUeMY7tCwmdPe9Hb
```

n9C1lUqBn4/acTtCClWAjREZn0zXAsNixtTIPC1V+9nO9YmecMkVwNfIPk
OPFnuZ/Dz1rCRHjNHb5j6ZyUM5zDqUVnnezktxqrOENSxm0gfMGcpxHQog
6UyBBADSCXHPfo/VPVtMm5p1yGrNOR2jR2rUj9+poZZDgjkt5G/xIKRlk
emu27wr9dVEX7ms0nvDq58iutbQ4d0JIDlcHMeSRQZluErb1B75Vj3HtIm
4Jx6SWRXPUJPGXGI87u0UoBH0Lwij7M2PW711ao+MLEA9jaJQwQA+sr9BK
r5nE72gsbCCLowkC0rdldf1RGtobwYDMpmYZhOaRKjkOTMG6rCXJxrf6Lq
/gNziTmch35MCq/MZzA/bN4VMPyeIlwzxVZkJLsQ7yyqX/A7ac7B7DH0Kf
MSOAJhMmk1W1Q1RRNw3cnYi8w3q7X40EAL/w54FVvvPqp3+sCd86SAAapM
tIsuNVemMWdgNXwvK8AJsz7VrevU5yZ4B8hvCuQj1C7geaN/LXhiT8foRs
Bf+iHC/VNEv4k4uDb4lOgnHJYYyifB1wC+nn/EnXCZYQINMiala4M6Vqc/
nwkZt/89LsAiR/20HHR1c3Qga2V5IDx0ZXN0a2V5QGVtYWlsLmNvbT6JAT
ACgFAls1Zf0CGwMFCQHhM4AGCwkIBwMCBhUIAgkKCwQWAghMBAh4BAheAAA
c14gJ8wwbfwH/3VyVsPkQ11owRJNxvXGt1bY7BfrvU52yk+PPZYoes9Upd
8gAM9bx5Sk08q2UXSZLC6ffOpEW4uWgmGYf8JRoc3ooezTkmCBW8I1bU0q
opdXLuPGCE7hVWQe9HcSntiTLxGov1mJAo7TAoccXLbyuZh9Rf5vLoQdK
h5IqXaQOT100TeFeEpb9TIiwcntg3WCSU5P0DGoUAoanjDZ3KE8Qp7V74f
zHb8Fa jR62CXSHFKqpBgiNxntOk45NbXADn4eTUXPSnwPi46qoAp9UQogs
DOTB2UOqhutAMECaM7VtpEpV79i0Z/NfnBedA5gEVLV1/QEIAnabFdQ+8Q
pM1bF/JrQt3zUoc4BTqICaxdyzAfz0tUSf/7Zro2us99GlARqLWd8EqJcl
iZyUam6ZAzzFXCgnH5Y1sdtMTJZdLp5We0jwgCWG/ZLu4wzxOFFzDkiPv9
MNLTJrSp4hS5o2apKdbO4Ex83O4mJYnav/rEiDDCWU4T0lhv3hSKCpke6L
liozp+aNmP0Ypwfi4hR3UUMP70+V1beFqW2JbVLz31LLouHRgpCzla+Pzz
jq77vG9kqZTCIzXoWaLljuitrlfJkO3vQ9hOv/8yAnkcAmowZrIBlyFg2K
mN2YvkUAEQEAAQAH/A7r4hDrnmzX3QU6FAzePlRB7niJtE2IEN8AufF05Q
c1S72WjtqMAIAgYasDkOhfhcxanTneGuFVYggKT3eSDm1RFKpRjX22m0zK
Mu95V20klul6OCm8d06+2fmkGxGqc4ZsKy+jQxtxK3HG9YxMC0dvA2v2C5
Utc7zh//k6IbmaLd7F1d7DXt7Hn2Qsmo8I1rtgPE8grDToomTnRUodToye
ORwsp8n8g2CSFaXSrEyU6HbFYXSxZealhQJGYLFozdR0MzVtZQCn/7n+IH
Nd2a8DVx3yQS3dAmvLzhFacZdjXi31wvj0moFOkEAOCz1E63SKNNksniQ1

```

gaov6Ux/zGLMstwTzNouI+Kr8/db0G1SAy1Z3UoAB4tFQXEAp0X9A4AJ2K
cZVULenfDZaxrbb9Lid7ZnTDXKVyGTWDF7ZHavHJ4981mCW17lU11zHBB9
dhFvb0gdy0jSLaFMFr/JBAD0fz3RrhP7e6X112zdBqGthjC5S/IoKwwBgw
LoxqBr2p19PotJJ/JUMPhD/LxuTcOZtYjy8PKgm5jhNBdq3Ss0kNKAY1f5
6I4iAX/NekqSyF+OgBfC9aCgS5RG8hYoOCbp8na5R3bgiuS8IzmVmm5OhZ
nQP7BzmR0p5BahpZ8r3Ada7FcK+0ZLLRdLmOYF/yUrZ53SoYCZRzU/GmtQ
Gjqied9Bs1MHdNUolq7GaexcjZmOWHEf6w9+9M4+vxtQq1nkIWqtaphewE
EP3sIY0EAE3mmiLmHLqBju+UJKMNwFNeyMTqgchg50ISH8J9FRIkBJQQYAQ
VLVl/QibDAUJAeEzgAAKCRAHFxJeICfMMOHYCACFhInZA9uAM3TC44l+Mr
W9izrO48WrdTsxR8WkSNbIxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRr
QNPSvz62WH+N21asoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQ
kk8qb5x/HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy
WE8pvgeEx/UUZB+dYqCwtvX0nnBu1KNcmk2AkEcFK3YoliCxomdOxhFOv9A
yC65KJciPv2MikPS2fKOAg1R3LpMa8zDETl4w3vckPQNrQNnYuUtfj6ZoC
=fa+6
-----END PGP PRIVATE KEY BLOCK-----') AS privkey) AS
keys;

username | decrypted_ssn
-----
Alice    | 123-45-6788
Bob      | 123-45-6799
(2 rows)

```

如果用户使用口令创建一个密钥，用户可能必须在此处输入口令。不过对于这个演示例子，口令为空。

加密gpfdist连接

`gpfdists`协议是`gpfdist`协议的一个安全版本，它能安全地标识文件服务器和Greenplum数据库并且加密它们之间的通信。使用`gpfdists`可以防止窃听和中间人攻击。

`gpfdists`协议利用下列值得关注的特性实现客户端/服务器的SSL安

全性：

- 要求客户端证书。
- 不支持多语言证书。
- 不支持证书撤销列表（CRL）。
- TLSv1协议被用于TLS_RSA_WITH_AES_128_CBC_SHA加密算法。这些SSL参数不能被更改。
- 不支持SSL再协商。
- SSL忽略主机失配参数被设置为false。
- gpfdist文件服务器（server.key）或Greenplum数据库（client.key）不支持含有口令的私钥。
- 为使用的操作系统颁发合适的证书是用户的责任。通常，支持将证书转换成所要求的格式，例如可使用 <https://www.ssllshopper.com/ssl-converter.html> 的SSL转换器。

用--ssl选项启动的gpfdist服务器只能用gpfdists协议通信。没有用--ssl选项启动的gpfdist服务器只能用gpfdist协议通信。更多有关gpfdist的细节请参考*Greenplum*数据库管理员指南。

有两种方式启用gpfdists协议：

- 用--ssl选项运行gpfdist，然后在CREATE EXTERNAL TABLE语句的LOCATION子句中使用gpfdists协议。
- 在YAML控制文件中将SSL选项设置为true，然后用它来运行gupload。运行的gupload会用--ssl选项启动gpfdist服务器，然后使用gpfdists协议。

在使用gpfdists时，下列客户端证书必须位于每个segment的\$PGDATA/gpfdists目录中：

- 客户端证书文件，client.crt
- 客户端私钥文件，client.key
- 受信证书发布机构，root.crt

Important: 重要：不要用口令保护私钥。服务器不会为私钥提示要求口令，并且数据装载会在要求口令时失败报错。

在使用带SSL的gupload时，用户要在YAML控制文件中指定服务器证书的位置。在使用带SSL的gpfdist时，用户用--ssl选项指定服务器证书的位置。

下面的例子展示了如何安全地装载数据到外部表中。这个例子从所有带txt扩展名的文件使用gpfdists协议创建一个可读外部

表ext_expenses。这些文件被格式化为用一个竖线 (|) 作为列定界符，并且用空格表示空。

1. 在segment主机上用--ssl选项运行gpfdist。

2. 登入数据库并执行下列命令：

```
=# 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 '' ) ;
```

Parent topic: [Greenplum数据库最佳实践](#)

[1](#) SHA2算法在OpenSSL 0.9.8版本中增加。在老版本中，pgcrypto会采用其内建代码。

[2](#) 任何OpenSSL支持的算法都可以使用，那些需要显式提供密码的除外。

[3](#) AES算法在OpenSSL 0.9.7版本中增加。在老版本中，pgcrypto会采用其内建代码。

Greenplum数据库® 6.0文档

□ Greenplum数据库最佳实践

最佳实践概要

系统配置

模式设计

采用资源组管理内存和资源

采用资源队列管理内存和资源

□ 系统监控和维护

装载数据

安全性

加密数据和数据库连接

SQL查询调优

高可用性

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

Greenplum数据库基于代价的优化器会权衡所有执行查询的策略并选择代价最小的策略去执行。

和其他RDBMS的优化器类似，在计算可选执行计划的代价时，Greenplum的优化器会考虑诸如要连接的表中的行数、索引的可用性以及列数据的基数等因素。优化器还会考虑数据的位置、倾向于在segment上做尽可能多的工作以及最小化完成查询必须在segment之间传输的数据量。

当查询运行得比预期慢时，用户可以查看优化器选择的计划以及它为计划的每一步计算出的代价。这将帮助用户确定哪些步骤消耗了最多的资源，然后修改查询或者模式来为优化器提供更加有效的执行方法。用户可以使用SQL语句EXPLAIN来查看查询的执行计划。

优化器基于为表生成的统计信息产生计划。精确的统计信息对于产生最好的执行计划非常重要。有关更新统计信息的方法请见[用ANALYZE更新统计信息](#)。

Parent topic: [Greenplum数据库最佳实践](#)

如何生成执行计划

EXPLAIN和EXPLAIN ANALYZE语句是查询原因并改进查询性能的有用工具。EXPLAIN会为查询显示其查询计划和估算的代价，但是不执行该查询。EXPLAIN ANALYZE除了显示查询的查询计划之外，还会执行该查询。EXPLAIN ANALYZE会丢掉任何来自SELECT语句的输出，但是其他语句中的操作会被执行（例如INSERT, UPDATE或DELETE）。要在DML语句上使用EXPLAIN ANALYZE却不让该命令影响数据，可以明确地把EXPLAIN ANALYZE用在一个事务中（BEGIN; EXPLAIN ANALYZE ...; ROLLBACK;）。

EXPLAIN ANALYZE语句运行后除了显示执行计划外，还有以下额外信息：

- 运行该查询消耗的总时间（以毫秒计）
- 计划节点操作中涉及的worker（segments）数量
- 操作中产生最多行的segment返回的最大行数（及其segment ID）
- 操作所使用的内存

- 从产生最多行的segment中检索到第一行所需的时间（以毫秒计），以及从该segment中检索所有行花费的总时间。

如何阅读执行计划

执行计划是一份报告，它详细描述了Greenplum数据库优化器确定的执行查询要遵循的步骤。查询计划是一棵由节点构成的树，应该从下向上阅读，每一个节点都会将其结果传递给其上一级节点。每个节点表示执行计划中的一个步骤，每个节点对应的那一行标识了在该步骤中执行的操作—例如一个扫描、一个连接、一个聚集或者排序操作。节点还标识了用于执行该操作的方法。例如，扫描操作的方法可能是顺序扫描或者索引扫描。而连接操作可以执行哈希连接或者嵌套循环连接。

下面是一个简单查询的执行计划。该查询在存储于每一segment中的分布表中查找行数。

```
gpadmin=# EXPLAIN SELECT gp_segment_id, count(*)
               FROM contributions
              GROUP BY gp_segment_id;
          QUERY PLAN
-----
-----
      Gather Motion 2:1  (slice2; segments: 2)
(cost=0.00..431.00 rows=2 width=12)
      ->  GroupAggregate  (cost=0.00..431.00 rows=1 width=12)
          Group By: gp_segment_id
          ->  Sort  (cost=0.00..431.00 rows=1 width=12)
              Sort Key: gp_segment_id
              ->  Redistribute Motion 2:2  (slice1;
segments: 2)  (cost=0.00..431.00 rows=1 width=12)
                  Hash Key: gp_segment_id
                  ->  Result  (cost=0.00..431.00 rows=1
width=12)
                      ->  GroupAggregate
(cost=0.00..431.00 rows=1 width=12)
                      Group By: gp_segment_id
                      ->  Sort
(cost=0.00..431.00 rows=7 width=4)
                      Sort Key:
gp_segment_id
                      ->  Seq Scan on
table1  (cost=0.00..431.00 rows=7 width=4)
Optimizer status: Pivotal Optimizer (GPORCA) version
2.56.0
(14 rows)
```

这个执行计划有八个节点 – Seq

Scan、Sort、GroupAggregate、Result、Redistribute Motion、Sort、GroupAggregate和最后的Gather Motion。每一个节点包含三个代价评估：代价评估（以顺序页面读取的方式）- cost、行数 - rows、以及行宽度 - width。

代价评估由两部分构成。1.0的代价等于一次顺序磁盘页面读取。代价评估的第一部分是启动代价，它是获取第一行的代价。第二部分是总代价，它是得到所有行的代价。

行数评估是由计划节点输出的行数。这个数字可能会小于执行计划节点实际处理或者扫描的行数，它反映了WHERE子句条件的选择度评估。总代价代表假设所有的行将被检索出来的代价评估，但并非总是这样（例如，如果用户使用LIMIT子句，情况可能不一样）。

宽度评估是计划节点输出的所有列的以字节计的总宽度。

节点中的代价评估包括了其所有子节点的代价总和，因此执行计划中最顶层节点（通常是一个Gather Motion）具有对计划总体执行代价的评估。这就是查询规划器想要最小化的那个数字。

扫描操作符扫描表中的行以寻找一个行的集合。对于不同种类的存储有不同的扫描操作符。它们包括：

- 表上的Seq Scan — 扫描表中的所有行。
- Index Scan — 遍历一个索引以从表中取得行。
- Bitmap Heap Scan — 从索引中收集表中行的指针并且按照磁盘上的位置进行排序。（无论是否是AO表，该操作都会调用一个Bitmap Heap Scan）
- Dynamic Seq Scan — 使用一个分区选择函数来选择分区。

Join操作符包括以下这些：

- Hash Join – 从较小的表构建一个哈希表，用连接列作为哈希键。然后扫描较大的表，为连接列计算哈希键 并且探索哈希表寻找具有相同哈希键的行。哈希连接通常是Greenplum数据库中最快的连接方式。执行计划中的Hash Cond 显示列出要被连接的列。
- Nested Loop – 在较大数据集的行上迭代，在每次迭代时从较小的数据集中扫描行。嵌套循环连接要求广播 其中的一个表，这样一个表中的所有行才能与其他表中的所有行进行比较。它在较小的表或者通过使用索引约束的表上 性能表现的更好。它还被用于笛卡尔积和范围连接。在使用Nested Loop连接大型表时会有性能影响。对于包含 Nested Loop连接操作符的执行计划节点，应该验证SQL并且确保结果是想要的结果。设置服务器配置参数enable_nestloop 为OFF（默认）能够让优化器更倾向于使用Hash Join。

Merge Join – 排序两个数据集并且将它们合并起来。归并连接对预排序好的数据很快，但是在现实世界中很少见。为了更倾向于使用Merge Join而不是Hash Join，可以把系统配置参数enable_mergejoin设置为ON。

一些查询计划节点指定数据移动操作。在处理查询操作时，数据移动操作在Segment之间进行。该节点标识执行移动操作 使用的方法。Motion操作符包括以下这些：

- Broadcast motion – 每一个segment将自己的行发送给所有其他segment，这样每一个segment实例都有表的一份完整的本地拷贝。Broadcast motion可能不如Redistribute motion那么好，因此优化器通常只在小表上选择 Broadcast motion。对大表来说，Broadcast motion是不可接受的。在数据没有按照连接键分布的情况下，将把一个表中所需的行动态重分布到另一个segment。
- Redistribute motion – 每一个segment重新哈希数据并且把行发送到对应于哈希键的合适的segment上。
- Gather motion – 来自所有segment的结果数据被组装成一个单一的流。对大部分执行计划来说这是最后的操作。

查询计划中出现的其他操作符包括：

- Materialize – 规划器将一个子查询物化一次，这样就不用为顶层行重复该工作。
- InitPlan – 一个预查询，被用在动态分区裁剪中，当执行时还不知道规划器需要用来标识要扫描分区的值时，会执行这个预查询。
- Sort – 为另一个要求排序数据的操作（例如Aggregation或者Merge Join）准备排序数据。
- Group By – 通过一个或者更多列分组行。
- Group/Hash Aggregate – 使用哈希聚集行。
- Append – 串联数据集，例如在整合从分区表中各分区扫描的行时会用到。
- Filter – 使用来自于一个WHERE子句的条件选择行。
- Limit – 限制返回的行数。

优化Greenplum查询

这个主题描述可以用来在某些情况下提高系统性能的Greenplum数据库特性和编程实践。

为了分析执行计划，首先找出评估代价非常高的计划节点。判断估计

的行数和代价是不是和该操作执行的行数相关。

如果使用分区，验证是否实现了分区裁剪。要实现分区裁剪，查询谓词（WHERE子句）必须与分区条件相同。还有，WHERE子句不能包含显式值且不能含有子查询。

审查查询计划树的执行顺序。审查估计的行数。用户想要执行顺序构建在较小的表或者哈希连接结果上并且用较大的表来探查。最优情况下，最大的表被用于最后的连接或者探查以减少传递到树最顶层计划节点的行数。如果分析结果显示构建或探查的执行顺序不是最优的，应确保数据库统计信息为最新。运行ANALYZE将能更新数据库统计信息，进而产生一个最优的查询计划。

查找计算性倾斜的迹象。当Hash Aggregate和Hash Join之类的操作符的执行导致segment上的不平均执行时，查询执行中会发生计算性倾斜。在一些segment上会使用比其他segment更多的CPU和内存，导致非最优化执行。原因可能是在具有低基数或者非一致分布的列上使用连接、排序或者聚集操作。用户可以在查询的EXPLAIN ANALYZE语句中检测计算性倾斜。每个节点包括任一segment所处理的最大行数以及所有segment处理的平均行数。如果最大行数远大于平均数，那么至少有一个segment执行了比其他segment更多的工作，因此应该怀疑该操作符出现了计算性倾斜。

确定执行Sort或者Aggregate操作的执行计划节点。Aggregate操作下隐藏的就是一个Sort。如果Sort或者Aggregate操作涉及到大量行，这就是改进查询性能的机会。在需要排序大量行时，HashAggregate操作是比Sort和Aggregate操作更好的操作。通常优化器会因为SQL结构（也就是编写SQL的方式）而选择Sort操作。在重写查询时，大部分的Sort操作可以用HashAggregate替换。要更倾向于使用HashAggregate操作而不是Sort和Aggregate，请确保服务器配置参数enable_groupagg被设置为ON。

当执行计划显示带有大量行的广播移动时，用户应该尝试消除广播移动。一种方法是使用服务器配置参数gp_segments_for_planner来增加这种移动的代价评估，这样优化器会偏向其他可替代的方案。gp_segments_for_planner变量告诉查询规划器在其计算中使用多少主segment。默认值是零，这会告诉规划器在估算中使用实际的主segment数量。增加主segment的数量会增加移动的代价，因此会更加偏向重新分布移动而不是广播。例如，设置gp_segments_for_planner = 100000会告诉规划器有100,000个segment。反过来，要影响规划器广播表而不是重新分布它，可以把gp_segments_for_planner设置为一个较低的值，例如2。

Greenplum分组扩展

Greenplum数据库对GROUP BY子句的聚集扩展可以让一些常见计算在数据库中执行得比在应用或者存储过程代码中更加高效：

- GROUP BY ROLLUP(*col1, col2, col3*)
- GROUP BY CUBE(*col1, col2, col3*)
- GROUP BY GROUPING SETS((*col1, col2*), (*col1, col3*))

ROLLUP分组创建从最详细层次上滚到总计的聚集小计，后面跟着分组（或者表达式）列表。ROLLUP接收分组列的一个有序列表，计算GROUP BY子句中指定的标准聚集值，然后根据该列表从右至左渐进地创建更高层的小计。最后创建总计。

CUBE分组创建给定分组（或者表达式）列表所有可能组合的小计。在多维分析术语中，CUBE产生一个数据立方体在指定维度可以被计算的所有小计。

用户可以用GROUPING SETS表达式选择性地指定想要创建的分组集。这允许在多个维度间进行精确的说明而无需计算整个ROLLUP或者CUBE。

这些子句的细节请参考*Greenplum数据库参考指南*。

窗口函数

窗口函数在结果集的划分上应用聚集或者排名函数一例如，`sum(population) over (partition by city)`。窗口函数很强大，因为它们的所有工作都在数据库内完成，它们比通过从数据库中检索细节行并且预处理它们来产生类似结果的前端工具具有性能优势。

- `row_number()` 窗口函数为一个划分中的行产生行号，例如 `row_number() over (order by id)`。
- 当查询计划表明一个表被多个操作扫描时，用户可以使用窗口函数来降低扫描次数。
- 经常可以通过使用窗口函数消除自连接。

Greenplum数据库® 6.0文档

□ Greenplum数据库最佳实践

最佳实践概要

系统配置

模式设计

采用资源组管理内存和资源

采用资源队列管理内存和资源

□ 系统监控和维护

装载数据

安全性

加密数据和数据库连接

SQL查询调优

高可用性

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

[如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息](#)

当用户启用并且正确地配置Greenplum高可用特性时，Greenplum数据库支持高度可用的、容错的数据库服务。要保证达到要求的服务等级，每个组件都必须有一个备用组件以保证在它失效时及时顶上。

Parent topic: [Greenplum数据库最佳实践](#)

磁盘存储

源于Greenplum数据库的非共享MPP架构，每台master和segment主机都有它们自己独立的内存和磁盘存储空间，每个master和segment实例都有它们自己独立的数据目录。为了兼顾可靠性和高性能，Pivotal推荐采用8到24块磁盘的硬件RAID存储解决方案。当采用RAID5或RAID6时，大量的磁盘会提升I/O吞吐量，因为条带会增加并行的磁盘I/O。有一个失效磁盘时，RAID控制器能够继续工作，因为它在每个磁盘上都保存了校验数据，这样它能够重构阵列中任何失效磁盘上的数据。如果配置了热备盘（或者配置了能够用新磁盘替代故障磁盘的操作器），控制器能够自动重建故障磁盘。

在RAID1模式下，实际上就是镜像一组磁盘，因此如果出现某块磁盘故障，替代磁盘立即可用，并且性能与出现磁盘故障之前无二。

在RAID5模式下，故障磁盘上的每一个数据I/O都必须从剩余活动磁盘上重建出来，直到故障磁盘重建完成，因此会出现一段时间的性能下降。如果磁盘数据重建期间，Greenplum数据库master和segment配置了镜像实例，您可以将任何受到影响的Greenplum数据库实例切换为它们的镜像以保证性能最优。

RAID磁盘阵列仍然可能会出现单点故障，例如整个RAID卷故障。在硬件级别上，可以通过RAID控制器提供的镜像功能或操作系统提供的镜像功能来防范磁盘阵列故障。

定期监控每台主机的可用磁盘空间是很重要的。可以通过查询gptoolkit模式下的外部表 `gp_disk_free` 来查看segment节点的磁盘可用空间。该视图会运行Linux命令`df`。在执行占用大量磁盘空间的操作（例如copy大表）前要确保检查可用磁盘空间。

详见Greenplum数据库参考指南中的`gp_toolkit.gp_disk_free`部分。



最佳实践

- 使用带有8到24个磁盘的硬件RAID存储方案。
- 使用RAID 1、5或6，这样磁盘阵列能容忍一个故障磁盘。
- 在磁盘阵列中配置一个热备盘以允许在检测到磁盘失效时自动开始重建。
- 通过镜像RAID磁盘组防止整个磁盘阵列的故障和重建期间的性能衰退。
- 定期监控磁盘利用率并且在需要时增加额外的磁盘空间。
- 监控segment数据倾斜以确保所有segment节点的数据均匀分布、空间合理利用。

Master镜像

Greenplum数据库的master实例是客户端访问系统的唯一入口。master实例存储全局系统目录，也就是存储有关数据库实例的元数据的一系列系统表，但是它不存储用户数据。如果未配置镜像的master实例故障或不可访问，会直接导致Greenplum数据库离线，不能正常提供服务。因此，必须配置备用master实例以防止主master故障。

Master镜像使用两个进程使镜像实例与master同步，一个sender位于活动master主机上，一个receiver位于镜像主机上。随着客户端操作的变化数据被应用到master系统目录上，活动master会将预写日志（WAL）以流复制的方式应用到备用master节点，以保证每一个应用到master实例的事务也同时应用到了备用master上。

镜像是一个温备。如果主master失效，要切换到备用master需要管理员用户在备用主机上运行 `gpactivatestandby` 工具，这样它才会开始接受客户端连接。客户端此时必须重新连接到新 master，该客户端在主master故障时未提交的事务都会丢失。

更多信息请见 *Greenplum* 数据库管理员指南中的“启用高可用特性”。

最佳实践

- 设置一个备用master实例—镜像—在主master故障时接手工作。
- 备用实例可以位于主实例同一或者不同主机上，但是最佳实践是把它放在不同于主master的不同主机上，这样可以防止主机故障。
- 规划好当故障发生时如何把客户端切换到新的master实例，例如通过更新DNS中的master地址。

- 设置监控，当主Master故障时在系统监控应用中发送通知或者通过邮件通知。

Segment镜像

Greenplum数据库的每一个segment实例都在master实例的协调下存储和管理数据库数据的一部分。如果任何未配置 镜像的segment故障，数据库可能不得不被关闭然后恢复，并且在最近备份之后发生的事务可能会丢失。因此，镜像 segment是高可用方案的一个不可或缺的元素

Segment镜像是主segment的热备。Greenplum数据库会检测到segment何时不可用并且自动激活其镜像。在正常操作期间，当主segment实例活动时，数据以两种方式从主segment复制到镜像segment：

- 在事务被提交之前，事务提交日志被从主segment复制到镜像segment。这会确保当镜像被激活时，主segment 上最后一个成功的事务所作的更改会出现在镜像上。当镜像被激活时，日志中的事务会被应用到镜像中的表上。
- 第二种，segment镜像使用物理文件复制来更新堆表。Greenplum服务器在磁盘上以打包了元组的固定尺寸块 的形式存储表数据。为了优化磁盘I/O，块被缓冲在内存中，直到缓存被填满并且一些块必须被挤出去为新更新的块腾出空间。当块被从缓存中挤出时，它会被写入到磁盘并且通过网络复制到镜像。因为缓冲机制，镜像上的表更新可能落后于主segment。不过，由于事务日志也被复制，镜像会与主segment保持一致。如果镜像被激活，激活过程会用事务提交日志中未应用的更改更新表。

当活动的主segment不能访问其镜像时，复制会停止并且主segment的状态会改为“Change Tracking”。主segment 会把没有被复制到镜像的更改保存在一个系统表中，等到镜像重新在线时这些更改会被复制到镜像。

Master会自动检测segment故障并且激活镜像。故障时正在进行的事务会使用新的主segment重新开始。根据镜像被部署在主机上的方式，数据库系统可能会不平衡直到原始的主segment被恢复。例如，如果每台segment主机有四个主 segment和四个镜像segment，并且在一台主机上有一个镜像segment被激活，那台主机将有五个活动的主segment。查询直到最后一个Segment完成其工作才算结束，因此性能可能会退化直至原始的主segment被恢复使得系统恢复平衡。

当Greenplum数据库运行时，管理员通过运行gprecoverseg工具执行恢复。这个工具会定位 故障的segment、验证它们是否有效并且与当前活动的segment对比事务状态以确定segment离线期间发生的更

改。 `gprecoverseg`会与活动segment同步更改过的数据库文件并且将该segment重新拉回到在线状态。

在故障期间，有必要在segment主机上保留足够的内存和CPU资源，以允许承担了主segment觉得的镜像实例能提供 对应的活动负载。[为Greenplum数据库配置内存](#)中提供的配置 segment主机内存的公式包括一个因子，它代表故障期间任一主机上的最大主segment数量。segment主机上的镜像 布置会影响这一因子以及系统在故障时将如何应对。Segment镜像选项的讨论请见[Segment镜像配置](#)。

最佳实践

- 为所有segment设置镜像。
- 将主segment及其镜像放置在不同主机上以防止主机失效。
- 镜像可以放在一组单独的主机上或者主segment所在的主机上。
- 设置监控，当主segment故障时在系统监控应用中发送通知或者通过邮件通知。
- 即使恢复失效的segment，使用`gprecoverseg`工具来恢复冗余并且让系统回到最佳的 平衡状态。

双集群

对于一些用例，可以通过维护两个存储同样数据的Greenplum数据库集群提供额外层次的冗余。实现双集群的决定 应该考虑到业务需求。

在双集群配置中为了保持数据同步，有两种推荐方法。第一种方法被称作双ETL。ETL（抽取、转换和装载）是常见的清理、转换、验证并且将数据装载到数据仓库中的常见数据仓库处理。通过双ETL，ETL处理会被以并行的方式执行两次，每个集群上一次，并且每一次都会做验证。双ETL提供了一个存有相同数据的备用集群。它还提供了在两个集群 上查询数据的能力，并使得处理吞吐量翻倍。应用可以根据需要利用两个集群，还要确保ETL在两边都成功并且被验证。

维护双集群的第二种机制是备份和恢复。数据在主◆◆

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

□ 客户端工具参考

□ 附加程序

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

本章描述了Greenplum数据库提供的命令行管理工具相关内容

Greenplum数据库使用标准的PostgreSQL客户端和服务器程序，并且额外提供了用于管理分布式Greenplum数据库的管理工具。Greenplum数据库管理工具位于\$GPHOME/bin目录下。

Note: 在gpfdist的URL中引用IPv6地址或者在任何管理工具中使用数字形式的IP地址而不是主机名时，请把IP地址放在方括号中。在命令提示符使用时，最好的方式是将方括号转义或者把它们放在引号内。例如：\

[2620:0:170:610::11\]或'[2620:0:170:610::11]'。

下列是Greenplum数据库的管理工具。

[analyzedb](#)[gpactivatestandby](#)[gpaddmirrors](#)[gpbackup](#)[gpcheck](#)[gpcheckcat](#)[gpchecknet \(已弃用\)](#)[gpcheckos \(已弃用\)](#)[gpcheckperf](#)[gpconfig](#)[gpdeletesystem](#)[gpdetective \(已弃用\)](#)[gpexpand](#)[gpfdist](#)[gpinitstandby](#)[gpinitsystem](#)[gplogfilter](#)[gmapreduce](#)[gpperfmon_install](#)[gppkg](#)[gprebuildsystem \(已弃用\)](#)[gprecoverseg](#)[gpreload](#)[gprestore](#)[gpsizecalc \(已弃用\)](#)[gpscp](#)[gpskew \(已弃用\)](#)[gpseginstall](#)[gpssh](#)[gpssh-exkeys](#)[gpstart](#)[gpstate](#)

gupload	gpstop
	gpsys1
	pgbouncer
	pgbouncer.ini
	pgbouncer-admin
	pxf
	pxf cluster

后端服务器程序

Greenplum数据库提供了下列标准的PostgreSQL数据库管理程序，它们位于\$GPHOME/bin中。为了处理Greenplum数据库系统的并行性和分布性，这些程序都经过了不同程度的修改。用户只能通过Greenplum数据库管理工具来访问这些程序。

Table 1. Greenplum数据库后端服务程序

程序名	描述	替代程序
initdb	在初始化一个Greenplum数据库集群时，这个程序会被gpinitstandby调用。它被用来创建集群内部每一个segment实例和master实例。	gpinitsystem
ipcclean	Greenplum数据库中没有使用	N/A
pg_basebackup	该程序用来对某个单独的数据库实例创建一个二进制拷贝。Greenplum数据库会在创建备用master实例或创建一个segment镜像的完整拷贝时使用。不要使用该工具备份Greenplum 数据	gpinitstandby , gprecoverseg

	库segment实例，因为它达不到MPP一致性备份的要求。	
pg_controldata	Greenplum数据库中没有使用	gpstate
pg_ctl	在启动或者停止Greenplum数据库阵列时，这个程序会被gpstart 和gpstop调用。在集群内部，它被用来以正确的选项并行地停止和启动segment实例和master实例。	gpstart , gpstop
pg_resetxlog	不要使用 警告：这个程序可能会导致数据丢失或者导致数据变得不可用。如果使用了这个程序，集群必须被重新初始化然后回复数据。	N/A
postgres	postgres可执行程序是实际处理查询的服务器进程。	主postgres进程 (postmaster) 按照需要创建其他的postgres子进程以及postgres会话来处理客户端连接。
postmaster	postmaster开启接受客户端连接的postgres 数据库服务器监听进程。在Greenplum数据库中，Greenplum的master实例和每一个segment实例上都会运行一个postgres数据库监听进程。	在Greenplum数据库中，用户可以使用 gpstart 和 gpstop 在系统中以正确的顺序和正确的选项一次性开启和关闭所有的postmaster进程 (postgres进程) 。

Greenplum数据库® 6.0文档

 工具指南 管理工具参考 客户端工具参考 附加程序

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

描述Greenplum数据库随附的命令行客户端工具。

Greenplum数据库使用标准的PostgreSQL客户端程序，并提供其他客户端工具来管理分布式Greenplum数据库DBMS。Greenplum数据库客户端工具位于\$GPHOME/bin.中。

以下是Greenplum数据库客户端工具。

- [clusterdb](#)
- [createdb](#)
- [createlang](#)
- [createuser](#)
- [dropdb](#)
- [droplang](#)
- [dropuser](#)
- [pg_config](#)
- [pg_dump](#)
- [pg_dumpall](#)
- [pg_restore](#)
- [psql](#)
- [reindexdb](#)
- [vacuumdb](#)
- [客户端工具摘要](#)

Parent topic: [Greenplum数据库工具指南](#)



Greenplum数据库® 6.0文档

[工具指南](#)[管理工具参考](#)[客户端工具参考](#)[附加程序](#)

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

本节介绍Greenplum数据库安装后的可用附加程序

以下PostgreSQL contrib服务器工具程序被安装：

- [pg_upgrade](#) - 用来升级Greenplum数据库服务器实例的服务端程序。
Note: pg_upgrade目前不会被Greenplum 6使用，但是在未来版本会被Greenplum升级工具使用。
- pg_upgrade_support - pg_upgrade支持库。
- [pg_xlogdump](#) - 显示Greenplum数据库 WAL日志为人类可读形式的服务器端程序。

Parent topic: [Greenplum数据库工具指南](#)



Greenplum数据库® 6.0文档

 参考指南 SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

 系统目录参考 gp_toolkit管理模式 gpperfmon 数据库 Greenplum 数据库数据类型

字符集支持

 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

The following SQL commands are available in Greenplum Database:

- [SQL Syntax Summary](#)
- [ABORT](#)
- [ALTER AGGREGATE](#)
- [ALTER COLLATION](#)
- [ALTER CONVERSION](#)
- [ALTER DATABASE](#)
- [ALTER DEFAULT PRIVILEGES](#)
- [ALTER DOMAIN](#)
- [ALTER EXTENSION](#)
- [ALTER EXTERNAL TABLE](#)
- [ALTER FOREIGN DATA WRAPPER](#)
- [ALTER FOREIGN TABLE](#)
- [ALTER FUNCTION](#)
- [ALTER GROUP](#)
- [ALTER INDEX](#)
- [ALTER LANGUAGE](#)
- [ALTER OPERATOR](#)
- [ALTER OPERATOR CLASS](#)
- [ALTER OPERATOR FAMILY](#)
- [ALTER PROTOCOL](#)
- [ALTER RESOURCE GROUP](#)
- [ALTER RESOURCE QUEUE](#)
- [ALTER ROLE](#)
- [ALTER SCHEMA](#)
- [ALTER SEQUENCE](#)
- [ALTER SERVER](#)
- [ALTER TABLE](#)
- [ALTER TABLESPACE](#)
- [ALTER TEXT SEARCH CONFIGURATION](#)



- [ALTER TEXT SEARCH DICTIONARY](#)
- [ALTER TEXT SEARCH PARSER](#)
- [ALTER TEXT SEARCH TEMPLATE](#)
- [ALTER TYPE](#)
- [ALTER USER](#)
- [ALTER USER MAPPING](#)
- [ALTER VIEW](#)
- [ANALYZE](#)
- [BEGIN](#)
- [CHECKPOINT](#)
- [CLOSE](#)
- [CLUSTER](#)
- [COMMENT](#)
- [COMMIT](#)
- [COPY](#)
- [CREATE AGGREGATE](#)
- [CREATE CAST](#)
- [CREATE COLLATION](#)
- [CREATE CONVERSION](#)
- [CREATE DATABASE](#)
- [CREATE DOMAIN](#)
- [CREATE EXTENSION](#)
- [CREATE EXTERNAL TABLE](#)
- [CREATE FOREIGN DATA WRAPPER](#)
- [CREATE FOREIGN TABLE](#)
- [CREATE FUNCTION](#)
- [CREATE GROUP](#)
- [CREATE INDEX](#)
- [CREATE LANGUAGE](#)
- [CREATE OPERATOR](#)
- [CREATE OPERATOR CLASS](#)
- [CREATE OPERATOR FAMILY](#)

- [CREATE PROTOCOL](#)
- [CREATE RESOURCE GROUP](#)
- [CREATE RESOURCE QUEUE](#)
- [CREATE ROLE](#)
- [CREATE RULE](#)
- [CREATE SCHEMA](#)
- [CREATE SEQUENCE](#)
- [CREATE SERVER](#)
- [CREATE TABLE](#)
- [CREATE TABLE AS](#)
- [CREATE TABLESPACE](#)
- [CREATE TEXT SEARCH CONFIGURATION](#)
- [CREATE TEXT SEARCH DICTIONARY](#)
- [CREATE TEXT SEARCH PARSER](#)
- [CREATE TEXT SEARCH TEMPLATE](#)
- [CREATE TYPE](#)
- [CREATE USER](#)
- [CREATE USER MAPPING](#)
- [CREATE VIEW](#)
- [DEALLOCATE](#)
- [DECLARE](#)
- [DELETE](#)
- [DISCARD](#)
- [DO](#)
- [DROP AGGREGATE](#)
- [DROP CAST](#)
- [DROP COLLATION](#)
- [DROP CONVERSION](#)
- [DROP DATABASE](#)
- [DROP DOMAIN](#)
- [DROP EXTENSION](#)
- [DROP EXTERNAL TABLE](#)

DROP FOREIGN DATA WRAPPER

- [DROP FOREIGN TABLE](#)
- [DROP FUNCTION](#)
- [DROP GROUP](#)
- [DROP INDEX](#)
- [DROP LANGUAGE](#)
- [DROP OPERATOR](#)
- [DROP OPERATOR CLASS](#)
- [DROP OPERATOR FAMILY](#)
- [DROP OWNED](#)
- [DROP PROTOCOL](#)
- [DROP RESOURCE GROUP](#)
- [DROP RESOURCE QUEUE](#)
- [DROP ROLE](#)
- [DROP RULE](#)
- [DROP SCHEMA](#)
- [DROP SEQUENCE](#)
- [DROP SERVER](#)
- [DROP TABLE](#)
- [DROP TABLESPACE](#)
- [DROP TEXT SEARCH CONFIGURATION](#)
- [DROP TEXT SEARCH DICTIONARY](#)
- [DROP TEXT SEARCH PARSER](#)
- [DROP TEXT SEARCH TEMPLATE](#)
- [DROP TYPE](#)
- [DROP USER](#)
- [DROP USER MAPPING](#)
- [DROP VIEW](#)
- [END](#)
- [EXECUTE](#)
- [EXPLAIN](#)
- [FETCH](#)

GRANT

- [INSERT](#)
- [LOAD](#)
- [LOCK](#)
- [MOVE](#)
- [PREPARE](#)
- [REASSIGN OWNED](#)
- [REINDEX](#)
- [RELEASE SAVEPOINT](#)
- [RESET](#)
- [REVOKE](#)
- [ROLLBACK](#)
- [ROLLBACK TO SAVEPOINT](#)
- [SAVEPOINT](#)
- [SELECT](#)
- [SELECT INTO](#)
- [SET](#)
- [SET CONSTRAINTS](#)
- [SET ROLE](#)
- [SET SESSION AUTHORIZATION](#)
- [SET TRANSACTION](#)
- [SHOW](#)
- [START TRANSACTION](#)
- [TRUNCATE](#)
- [UPDATE](#)
- [VACUUM](#)
- [VALUES](#)

Parent topic: [Greenplum数据库参考指南](#)

Greenplum数据库® 6.0文档

 参考指南 SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

 系统目录参考 gp_toolkit管理模式 gpperfmon 数据库 Greenplum 数据库数据类型

字符集支持

 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

Greenplum的PL/Perl语言扩展

用于分析的Greenplum MADlib扩展

 附加提供的模块

Greenplum Partner Connector API

下表列出了2008 SQL标准中描述的特性。Greenplum数据库支持的特性在“支持”列中标记为“是”，未实现的特性标记为“否”。

有关Greenplum特性和SQL兼容性的信息，请参阅Greenplum数据库管理员指南。

Table 1. SQL 2008可选特性合规细节

ID	特性	支持	注解
B011	嵌入式Ada	NO	
B012	嵌入式C	NO	由于PostgreSQL ecpg的问题
B013	嵌入式COBOL	NO	
B014	嵌入式Fortran	NO	
B015	嵌入式MUMPS	NO	
B016	嵌入式Pascal	NO	
B017	嵌入式PL/I	NO	
B021	直接SQL	YES	
B031	基本动态SQL	NO	
B032	扩展动态SQL	NO	
B033	无类型的SQL调用函数参数	NO	
B034	游标属性的动态规范	NO	
B035	非扩展描述符名称	NO	
B041	嵌入式SQL异常声明的扩展	NO	
B051	增强的执行权限	NO	
B111	模块语言Ada	NO	
B112	模块语言C	NO	
B113	模块语言COBOL	NO	
B114	模块语言Fortran	NO	
B115	模块语言MUMPS	NO	
B116	模块语言Pascal	NO	
B117	模块语言PL/I	NO	
B121	常规语言Ada	NO	
B122	常规语言C	NO	
B123	常规语言COBOL	NO	
B124	常规语言Fortran	NO	
B125	常规语言MUMPS	NO	
B126	常规语言Pascal	NO	
B127	常规语言PL/I	NO	
B128	常规语言SQL	NO	
E011	数值数据类型	YES	
E011-01	INTEGER和SMALLINT数据类型	YES	
E011-02	DOUBLE PRECISION和FLOAT数据类型	YES	
E011-03	DECIMAL和NUMERIC数据类型	YES	

E011-04	算术操作符	YES	
E011-05	数值比较	YES	
E011-06	数值数据类型之间的隐式转换	YES	
E021	字符数据类型	YES	
E021-01	CHARACTER数据类型	YES	
E021-02	CHARACTER VARYING数据类型	YES	
E021-03	字符文字	YES	
E021-04	CHARACTER_LENGTH函数	YES	在计数之前修剪CHARACTER值的尾随空格
E021-05	OCTET_LENGTH函数	YES	
E021-06	SUBSTRING函数	YES	
E021-07	字符连接	YES	
E021-08	UPPER和LOWER函数	YES	
E021-09	TRIM函数	YES	
E021-10	字符串类型之间隐式转换	YES	
E021-11	POSITION函数	YES	
E021-12	字符比较	YES	
E031	标识符	YES	
E031-01	分隔标识符	YES	
E031-02	小写标识符	YES	
E031-03	尾随下划线	YES	
E051	基本查询规范	YES	
E051-01	SELECT DISTINCT	YES	
E051-02	GROUP BY子句	YES	
E051-03	GROUP BY可以包含不在SELECT列表中的列	YES	
E051-04	可以重命名SELECT列表项	YES	
E051-05	HAVING子句	YES	
E051-	SELECT列表中受限定的*	YES	

06			
E051-07	FROM子句中的关联名称	YES	
E051-08	重命名FROM子句中的列	YES	
E061	基本谓词和搜索条件	YES	
E061-01	比较谓词	YES	
E061-02	BETWEEN谓词	YES	
E061-03	带有值列表的IN谓词	YES	
E061-04	LIKE谓词	YES	
E061-05	LIKE谓词ESCAPE子句	YES	
E061-06	NULL谓词	YES	
E061-07	量化比较谓词	YES	
E061-08	EXISTS谓词	YES	并非所有用途都适用于Greenplum
E061-09	比较谓词中的子查询	YES	
E061-11	IN谓词中的子查询	YES	
E061-12	量化比较谓词中的子查询	YES	
E061-13	相关子查询	YES	
E061-14	搜索条件	YES	
E071	基本查询表达式	YES	
E071-01	UNION DISTINCT表操作符	YES	
E071-02	UNION ALL表操作符	YES	
E071-03	EXCEPT DISTINCT表操作符	YES	
E071-05	通过表操作符组合的列不需要具有完全相同的数据类型	YES	
E071-06	子查询中的表操作符	YES	
E081	基本权限	NO	支持部分子特性
E081-01	SELECT权限	YES	
E081-02	DELETE权限	YES	
E081-03	INSERT表级别的权限	YES	

E081-04	UPDATE表级别的权限	YES	
E081-05	UPDATE列级别的权限	YES	
E081-06	REFERENCES表级的权限	NO	
E081-07	REFERENCES列级别的权限	NO	
E081-08	WITH GRANT OPTION	YES	
E081-09	USAGE权限	YES	
E081-10	EXECUTE权限	YES	
E091	集合函数	YES	
E091-01	AVG	YES	
E091-02	COUNT	YES	
E091-03	MAX	YES	
E091-04	MIN	YES	
E091-05	SUM	YES	
E091-06	ALL量词	YES	
E091-07	DISTINCT量词	YES	
E101	基本数据操作	YES	
E101-01	INSERT语句	YES	
E101-03	搜索UPDATE语句	YES	
E101-04	搜索DELETE语句	YES	
E111	单行SELECT语句	YES	
E121	基本光标支持	YES	
E121-01	DECLARE CURSOR	YES	
E121-02	ORDER BY列不需要在选择列表中	YES	
E121-03	ORDER BY子句的值表达式	YES	
E121-04	OPEN语句	YES	
E121-06	Positioned UPDATE语句	NO	
E121-07	Positioned DELETE语句	NO	

E121-08	CLOSE语句	YES	
E121-10	FETCH语句隐含 NEXT	YES	
E121-17	WITH HOLD游标	YES	
E131	空值支持	YES	
E141	基本完整性约束	YES	
E141-01	NOT NULL约束	YES	
E141-02	UNIQUE NOT NULL 列约束	YES	必须与Greenplum分布键相同或是它的超集
E141-03	PRIMARY KEY约束	YES	必须与Greenplum分布键相同或是它的超集
E141-04	参考删除操作和参照更新操作的NO ACTION默认值的基本FOREIGN KEY约束	NO	
E141-06	CHECK约束	YES	
E141-07	列默认值	YES	
E141-08	NOT NULL推断PRIMARY KEY	YES	
E141-10	外键中的名称可以按任何顺序指定	YES	可以声明外键，但在Greenplum中不会强制执行
E151	事务支持	YES	
E151-01	COMMIT语句	YES	
E151-02	ROLLBACK语句	YES	
E152	基本SET TRANSACTION语句	YES	
E152-01	ISOLATION LEVEL SERIALIZABLE子句	NO	可以声明，但被视为REPEATABLE READ的同义词
E152-02	READ ONLY和READ WRITE 子句	YES	
E153	具有子查询的可更新查询	NO	
E161	使用前导双减去的SQL注释	YES	
E171	SQLSTATE支持	YES	
E182	模块语言	NO	
F021	基本信息模式	YES	
F021-01	COLUMNS视图	YES	
F021-02	TABLES视图	YES	
F021-03	VIEWS视图	YES	
F021-04	TABLE_CONSTRAINTS视图	YES	
F021-05	REFERENTIAL_CONSTRAINTS视图	YES	

F021-06	CHECK_CONSTRAINTS视图	YES	
F031	基本模式操作	YES	
F031-01	CREATE TABLE语句创建持久性基表	YES	
F031-02	CREATE VIEW语句	YES	
F031-03	GRANT语句	YES	
F031-04	ALTER TABLE语句: ADD COLUMN子句	YES	
F031-13	DROP TABLE语句: RESTRICT子句	YES	
F031-16	DROP VIEW语句: RESTRICT子句	YES	
F031-19	REVOKE语句: RESTRICT子句	YES	
F032	CASCADE删除行为	YES	
F033	ALTER TABLE语句: DROP COLUMN子句	YES	
F034	扩展的REVOKE语句	YES	
F034-01	REVOKE语句由schema对象的所有者以外的其他语句执行	YES	
F034-02	REVOKE语句: GRANT OPTION FOR子句	YES	
F034-03	REVOKE语句撤销授权人WITH GRANT OPTION的权限	YES	
F041	基本连接表	YES	
F041-01	内连接 (但不一定是INNER关键字)	YES	
F041-02	INNER关键字	YES	
F041-03	LEFT OUTER JOIN	YES	
F041-04	RIGHT OUTER JOIN	YES	
F041-05	外连接可以嵌套	YES	
F041-07	左或右外连接中的内表也可用于内连接	YES	
F041-08	支持所有比较操作符(而不仅是=)	YES	
F051	基本的日期和时间	YES	
F051-01	DATE数据类型 (包括支持DATE文字)	YES	
F051-02	TIME数据类型 (包括支持TIME文字) 分秒精确度至少为0	YES	
F051-03	TIMESTAMP数据类型 (包括支持TIMESTAMP文字) 分秒精确度至少为0和6	YES	
F051-	DATE, TIME和TIMESTAMP数据类型的	YES	

04	比较谓词		
F051-05	日期时间类型和字符串类型之间的显式CAST	YES	
F051-06	CURRENT_DATE	YES	
F051-07	LOCALTIME	YES	
F051-08	LOCALTIMESTAMP	YES	
F052	间隔和日期时间算术	YES	
F053	OVERLAPS谓词	YES	
F081	视图中的UNION和EXCEPT	YES	
F111	除SERIALIZABLE之外的隔离级别	YES	
F111-01	READ UNCOMMITTED 隔离级别	NO	可以声明但是被视为READ COMMITTED的同义词
F111-02	READ COMMITTED隔离级别	YES	
F111-03	REPEATABLE READ隔离级别	YES	
F121	基本诊断管理	NO	
F122	增强诊断管理	NO	
F123	所有诊断	NO	
F131-	分组操作	YES	
F131-01	在分组视图的查询中支持的WHERE, GROUP BY和 HAVING子句	YES	
F131-02	在具有分组视图的查询中支持多个表	YES	
F131-03	设置具有分组视图的查询支持的函数	YES	
F131-04	具有GROUP BY和HAVING子句以及分组视图的子查询	YES	
F131-05	具有GROUP BY和HAVING子句以及分组视图的单行SELECT	YES	
F171	每个用户的多个模式	YES	
F181	多模块支持	NO	
F191	引用删除操作	NO	
F200	TRUNCATE TABLE语句	YES	
F201	CAST函数	YES	
F202	TRUNCATE TABLE: 标识列重新启动选项	NO	
F221	显式默认值显式默认值	YES	
F222	INSERT语句: DEFAULT VALUES 子句	YES	
F231	特权表	YES	
F231-01	TABLE_PRIVILEGES视图	YES	
F231-02	COLUMN_PRIVILEGES视图	YES	
F231-	USAGE_PRIVILEGES视图	YES	

03			
F251	域支持		
F261	CASE表达式	YES	
F261-01	简单CASE	YES	
F261-02	搜索CASE	YES	
F261-03	NULLIF	YES	
F261-04	COALESCE	YES	
F262	扩展的CASE表达式	NO	
F263	简单CASE表达式中逗号分隔的谓词	NO	
F271	复合字符文字	YES	
F281	LIKE增强	YES	
F291	UNIQUE谓词	NO	
F301	查询表达式中的CORRESPONDING	NO	
F302	INTERSECT表操作符	YES	
F302-01	INTERSECT DISTINCT表操作符	YES	
F302-02	INTERSECT ALL表操作符	YES	
F304	EXCEPT ALL table operator		
F311	模式定义语句	YES	支持部分子特性
F311-01	CREATE SCHEMA	YES	
F311-02	CREATE TABLE用于持久性基表	YES	
F311-03	CREATE VIEW	YES	
F311-04	CREATE VIEW: WITH CHECK OPTION	NO	
F311-05	GRANT语句	YES	
F312	MERGE语句	NO	
F313	增强的MERGE语句	NO	
F321	用户授权	YES	
F341	用法表	NO	
F361	子程序支持	YES	
F381	扩展的模式操作	YES	
F381-01	ALTER TABLE语句: ALTER COLUMN子句		更改分布键列的一些限制
F381-02	ALTER TABLE语句: ADD CONSTRAINT子句		
F381-03	ALTER TABLE语句: DROP CONSTRAINT子句		
F382	更改列数据类型	YES	更改分布键列的一些限制
F391		YES	

	长标识符		
F392	Unicode转义为标识符	NO	
F393	Unicode以文字形式转义	NO	
F394	可选的正常形式规格	NO	
F401	扩展连接表	YES	
F401-01	NATURAL JOIN	YES	
F401-02	FULL OUTER JOIN	YES	
F401-04	CROSS JOIN	YES	
F402	命名列连接LOB, 数组和多重集	NO	
F403	分区连接表	NO	
F411	时区规格	YES	关于字面解释的差异
F421	国家特征	YES	
F431	只读可滚动光标	YES	仅向前滚动
01	有显式NEXT的FETCH	YES	
02	FETCH FIRST	NO	
03	FETCH LAST	YES	
04	FETCH PRIOR	NO	
05	FETCH ABSOLUTE	NO	
06	FETCH RELATIVE	NO	
F441	扩展集函数支持	YES	
F442	集合函数中的混合列引用	YES	
F451	字符集定义	NO	
F461	命名字符集	NO	
F471	标量子查询值	YES	
F481	扩展的NULL谓词	YES	
F491	约束管理	YES	
F501	特征和一致性视图	YES	
F501-01	SQL_FEATURES视图	YES	
F501-02	SQL_SIZING视图	YES	
F501-03	SQL_LANGUAGES视图	YES	
F502	增强的文档表	YES	
F502-01	SQL_SIZING_PROFILES视图	YES	
F502-02	SQL_IMPLEMENTATION_INFO视图	YES	
F502-03	SQL_PACKAGES视图	YES	
F521	断言	NO	
F531	临时表	YES	非标准表格
F555	增强秒精度	YES	

F561	全值表达式	YES	
F571	真值测试	YES	
F591	派生表	YES	
F611	指标数据类型	YES	
F641	行和表构造函数	NO	
F651	Catalog名称限定	YES	
F661	简单表	NO	
F671	CHECK中的子查询	NO	故意省略
F672	回溯检查限制	YES	
F690	排序规则支持	NO	
F692	增强排序规则支持	NO	
F693	SQL会话和客户端模块排序规则	NO	
F695	翻译支持	NO	
F696	附加翻译文件	NO	
F701	引用更新动作	NO	
F711	ALTER域	YES	
F721	可延迟的约束	NO	
F731	INSERT列权限	YES	
F741	引用MATCH类型	NO	没有部分匹配
F751	视图CHECK增强	NO	
F761	会话管理	YES	
F762	CURRENT_CATALOG	NO	
F763	CURRENT_SCHEMA	NO	
F771	连接管理	YES	
F781	自引用操作	YES	
F791	不敏感的游标	YES	
F801	全套函数	YES	
F812	基本标记	NO	
F813	扩展标记	NO	
F831	完整游标更新	NO	
F841	LIKE_REGEX谓词	NO	正则表达式的非标准语法
F842	OCCURENCES_REGEX函数	NO	
F843	POSITION_REGEX函数	NO	
F844	SUBSTRING_REGEX函数	NO	
F845	TRANSLATE_REGEX函数	NO	
F846	正则表达式操作符中的八位字节支持	NO	
F847	非常规正则表达式	NO	
F850	查询表达式中的顶级ORDER BY子句	YES	
F851	子查询中的顶级ORDER BY子句	NO	
F852	视图中的顶级ORDER BY子句	NO	
F855	查询表达式中的嵌套ORDER BY子句	NO	
F856	查询表达式中的嵌套FETCH FIRST 子句	NO	
F857	查询表达式中的顶级FETCH FIRST子句	NO	

F858	子查询中的FETCH FIRST 子句	NO	
F859	视图中的顶级FETCH FIRST 子句	NO	
F860	FETCH FIRST子句中的FETCH FIRST ROW数量	NO	
F861	查询表达式中的顶级RESULT OFFSET子句	NO	
F862	子查询中的RESULT OFFSET子句	NO	
F863	查询表达式中的嵌套RESULT OFFSET子句	NO	
F864	视图中的顶级RESULT OFFSET子句	NO	
F865	RESULT OFFSET子句中的OFFSET ROW数量	NO	
S011	不同的数据类型	NO	
S023	基本结构化类型	NO	
S024	增强的结构化类型	NO	
S025	最终结构化类型	NO	
S026	自引用结构化类型	NO	
S027	按特定方法名称创建方法	NO	
S028	可置换的UDT选项列表	NO	
S041	基本参考类型	NO	
S043	增强的参考类型	NO	
S051	创建类型表	NO	
S071	函数和类型名称解析中的SQL路径	YES	
S091	基本数组支持	NO	Greenplum有数组，但不完全符合标准
S091-01	内置数据类型的数组	NO	部分兼容
S091-02	不同类型的数组	NO	
S091-03	数组表达式	NO	
S092	用户定义类型的数组	NO	
S094	参考类型的数组	NO	
S095	通过查询的数组构造函数	NO	
S096	可选数组边界	NO	
S097	数组元素赋值	NO	
S098	ARRAY_AGG	部分支持	<p>支持：使用没有窗口规范的array_agg；例如</p> <pre>SELECT array_agg(x) FROM ...</pre> <p>SELECT array_agg (x order by y) FROM ...</p> <p>不支持：使用array_agg作为聚合派生窗口函数；例如</p> <pre>SELECT array_agg(x) over (ORDER BY y) FROM ...</pre>

			SELECT array_agg(x order by y) over (PARTITION BY z) FROM ... SELECT array_agg(x order by y) over (ORDER BY z) FROM ...
S111	ONLY在查询表达式中	YES	
S151	类型谓词	NO	
S161	子类型处理	NO	
S162	引用的子类型处理	NO	
S201	SQL上调用的数组例程	NO	函数可以传递Greenplum数组类型
S202	多集上的SQL调用例程	NO	
S211	用户定义的强制转换函数	YES	
S231	结构类型定位器	NO	
S232	数组定位器	NO	
S233	Multiset定位器	NO	
S241	转换函数	NO	
S242	改变转换语句	NO	
S251	用户定义的命令	NO	
S261	特定类型方法	NO	
S271	基本的多重集支持	NO	
S272	用户自定义类型的多重集	NO	
S274	引用类型的多重集	NO	
S275	高级多重集支持	NO	
S281	嵌套集合类型	NO	
S291	整个行的唯一约束	NO	
S301	增强的UNNEST	NO	
S401	基于数组类型的不同类型	NO	
S402	基于不同类型的不同类型	NO	
S403	MAX_CARDINALITY	NO	
S404	TRIM_ARRAY	NO	
T011	信息模式中的时间戳	NO	
T021	BINARY和VARBINARY 数据类型	NO	
T022	支持高级的BINARY和VARBINARY数据类型	NO	
T023	复合二进制文字	NO	
T024	二进制文字中的空格	NO	
T031	BOOLEAN数据类型	YES	
T041	支持基本的LOB数据类型	NO	
T042	支持扩展的LOB数据类型	NO	
T043	乘数T	NO	
T044	乘数P	NO	
T051	行类型	NO	
T052	用于行类型的MAX和MIN	NO	
T053	全域引用的显式别名	NO	
T061	UCS	NO	

	支持		
T071	BIGINT数据类型	YES	
T101	增强的可空性确定	NO	
T111	可更新的连接, 联合和列	NO	
T121	查询表达式中的WITH(除了RECURSIVE)	NO	
T122	子查询中的WITH(除了RECURSIVE)	NO	
T131	递归查询	NO	
T132	子查询中的递归查询	NO	
T141	SIMILAR谓词	YES	
T151	DISTINCT谓词	YES	
T152	DISTINCT否定谓词	NO	
T171	表定义中的LIKE子句	YES	
T172	在表中定义AS子查询子句	YES	
T173	在表定义中扩展LIKE子句	YES	
T174	身份列	NO	
T175	生成列	NO	
T176	支持序列生成器	NO	
T177	支持顺序发生器: 简单的重启选项	NO	
T178	身份列: 简单重启选项	NO	
T191	参照活动RESTRICT	NO	
T201	参照约束的可比较数据类型	NO	
T211	基本触发功能	NO	
T211-01	一个基本表的UPDATE, INSERT, 或 DELETE上激活触发器	NO	
T211-02	BEFORE触发器	NO	
T211-03	AFTER触发器	NO	
T211-04	FOR EACH ROW触发器	NO	
T211-05	能够在调用触发器之前指定一个必须为true的搜索条件	NO	
T211-06	支持触发器和约束的交互的运行时规则	NO	
T211-07	TRIGGER特权	YES	
T211-08	同一事件的多个触发器按其在catalog中创建的顺序执行	NO	故意省略
T212	增强的触发器功能	NO	
T213	INSTEAD OF触发器	NO	
T231	敏感游标	YES	
T241	START TRANSACTION语句	YES	
T251	SET TRANSACTION语句: LOCAL选项	NO	
T261	联锁事务	NO	
T271	保存点	YES	
T272	增强的保存点管理	NO	

T281	SELECT具有列粒度的特权	YES	
T285	增强派生列名称	NO	
T301	函数依赖	NO	
T312	OVERLAY函数	YES	
T321	基本的SQL-invoked例程	NO	部分支持
T321-01	不带重载的用户定义的函数	YES	
T321-02	不带重载的用户定义的存储过程	NO	
T321-03	函数调用	YES	
T321-04	CALL语句	NO	
T321-05	RETURN语句	NO	
T321-06	ROUTINES视图	YES	
T321-07	PARAMETERS视图	YES	
T322	SQL-invoked的函数和过程的重载	YES	
T323	外部程序的显式安全性	YES	
T324	SQL例程的显式安全性	NO	
T325	合格的SQL参数引用	NO	
T326	表函数	NO	
T331	基本角色	NO	
T332	扩展角色	NO	
T351	括号SQL注释 /*...*/ 注释)	YES	
T431	扩展分组容量	NO	
T432	嵌套和级联GROUPING SETS	NO	
T433	多参数GROUPING函数	NO	
T434	GROUP BY DISTINCT	NO	
T441	ABS和MOD函数	YES	
T461	对称的BETWEEN谓词	YES	
T471	结果集返回值	NO	
T491	LATERAL派生表	NO	
T501	增强的EXISTS谓词	NO	
T511	事务总数	NO	
T541	可更新表引用	NO	
T561	可定位器	NO	
T571	Array-returning外部SQL-invoked函数	NO	
T572	Multiset-returning外部SQL-invoked函数	NO	
T581	正则表达式子串函数	YES	
T591	UNIQUE可能为空列的约束	YES	
T601	本地游标引用	NO	
T611	初级OLAP操作	YES	

T612	高级OLAP操作	NO	部分支持
T613	采样	NO	
T614	NTILE函数	YES	
T615	LEAD和LAG函数	YES	
T616	LEAD和LAG函数的空值处理选项	NO	
T617	FIRST_VALUE和LAST_VALUE函数	YES	
T618	NTH_VALUE	NO	函数存在于Greenplum中，但不支持所有选项
T621	增强数字函数	YES	
T631	N谓词与一个列表元素	NO	
T641	多列分配	NO	支持一些语法变体
T651	SQL例程中的SQL-schema语句	NO	
T652	SQL例程中的SQL动态语句	NO	
T653	外部例程中的SQL-schema语句	NO	
T654	外部例程中的SQL动态语句	NO	
T655	循环依赖例程	NO	
M001	数据链路	NO	
M002	数据链路通过SQL/CLI	NO	
M003	数据链路通过嵌入式SQL	NO	
M004	外部数据支持	NO	
M005	外部模式支持	NO	
M006	GetSQLString例程	NO	
M007	TransmitRequest	NO	
M009	GetOpts和GetStatistics例程	NO	
M010	外部数据包装支持	NO	
M011	通过Ada的数据链接	NO	
M012	数据链通过C	NO	
M013	数据链通过COBOL	NO	
M014	数据链路通过Fortran	NO	
M015	数据链路通过M	NO	
M016	数据链路通过Pascal	NO	
M017	数据链路通过PL/I	NO	
M018	Ada中的外部数据包装器接口例程	NO	
M019	C中的外部数据包装器接口例程	NO	
M020	COBOL中的外部数据包装器接口例程	NO	
M021	Fortran中的外部数据包装器接口例程	NO	
M022	MUMPS中的外部数据包装器接口例程	NO	
M023	Pascal中的外部数据包装器接口例程	NO	
M024	PL/I中的外部数据包装器接口例程	NO	
M030	SQL-server外部数据支持	NO	
M031	外部数据包装程序一般例程	NO	
X010	XML类型	YES	
X011	XML类型的数组	YES	
X012	XML类型的多重集	NO	

X013	XML类型的不同类型	NO	
X014	XML类型的属性	NO	
X015	XML类型的字段	NO	
X016	持久的XML值	YES	
X020	XMLConcat	YES	支持xmlconcat2()
X025	XMLCast	NO	
X030	XMLDocument	NO	
X031	XMLElement	YES	
X032	XMLForest	YES	
X034	XMLAgg	YES	
X035	XMLAgg: ORDER BY option	YES	
X036	XMLComment	YES	
X037	XMLPI	YES	
X038	XMLText	NO	
X040	基本表映射	NO	
X041	基本表映射: 空值不存在	NO	
X042	基本表映射: null as nil	NO	
X043	基本表映射: 表作为森林	NO	
X044	基本表映射: 表作为元素	NO	
X045	基本表映射: 具有目标命名空间	NO	
X046	基本表映射: 数据映射	NO	
X047	基本表映射: 元数据映射	NO	
X048	基本表映射: 二进制字符串的base64编码	NO	
X049	基本表映射: 二进制字符串的十六进制编码	NO	
X051	高级表映射: 空值不存在	NO	
X052	高级表映射: null as nil	NO	
X053	高级表映射: 表作为森林	NO	
X054	高级表映射: 表作为元素	NO	
X055	高级表映射: 目标命名空间	NO	
X056	高级表映射: 数据映射	NO	
X057	高级表映射: 元数据映射	NO	
X058	高级表映射: 二进制字符串的base64编码	NO	
X059	高级表映射: 二进制字符串的十六进制编码	NO	
X060	XMLParse: 字符串输入和CONTENT选项	YES	
X061	XMLParse: 字符串输入和DOCUMENT选项	YES	
X065	XMLParse: BLOB输入和CONTENT选项	NO	
X066	XMLParse: BLOB输入和DOCUMENT选项	NO	
X068	XMLSerialize: BOM	NO	
X069	XMLSerialize: INDENT	NO	
X070	XMLSerialize: 字符串序列化和CONTENT选项	YES	
X071	XMLSerialize: 字符串序列化	YES	

	和DOCUMENT选项		
X072	XMLSerialize: 字符串序列化	YES	
X073	XMLSerialize: BLOB序列化 和CONTENT选项	NO	
X074	XMLSerialize: BLOB序列化 和DOCUMENT选项	NO	
X075	XMLSerialize: BLOB序列化	NO	
X076	XMLSerialize: VERSION	NO	
X077	XMLSerialize: 显式ENCODING选项	NO	
X078	XMLSerialize: 显式XML声明	NO	
X080	XML发布中的命名空间	NO	
X081	查询级XML命名空间声明	NO	
X082	DML中的XML命名空间声明	NO	
X083	DDL中的XML命名空间声明	NO	
X084	复合语句中的XML命名空间声明	NO	
X085	预定义的命名空间前缀	NO	
X086	XMLTable中的XML命名空间声明	NO	
X090	XML文档谓词	NO	支持 xml_is_well_formed_document()
X091	XML内容谓词	NO	支持xml_is_well_formed_content()
X096	XMLExists	NO	支持xmlexists()
X100	主机语言支持XML: CONTENT选项	NO	
X101	主机语言支持XML: DOCUMENT选项	NO	
X110	主机语言支持XML: VARCHAR映射	NO	
X111	主机语言支持XML: CLOB映射	NO	
X112	主机语言支持XML: BLOB映射	NO	
X113	主机语言支持XML: STRIP WHITESPACE选项	YES	
X114	主机语言支持XML: PRESERVE WHITESPACE选项	YES	
X120	SQL例程中的XML参数	YES	
X121	外部例程中的XML参数	YES	
X131	查询级XMLBINARY子句	NO	
X132	DML中的XMLBINARY子句	NO	
X133	DDL中的XMLBINARY子句	NO	
X134	复合语句中的XMLBINARY子句	NO	
X135	子查询中的XMLBINARY子句	NO	
X141	IS VALID谓词: 数据驱动案例	NO	
X142	IS VALID谓词: ACCORDING TO子句	NO	
X143	IS VALID谓词: ELEMENT子句	NO	
X144	IS VALID谓词: 模式位置	NO	
X145	IS VALID谓词外部检查约束	NO	
X151	带有DOCUMENT选项的IS VALID谓词	NO	
X152	带有CONTENT选项的IS VALID谓词	NO	
X153	带有SEQUENCE选项的IS VALID谓词	NO	

X155	IS VALID谓词: 不带ELEMENT子句的NAMESPACE	NO	
X157	IS VALID谓词: NO NAMESPACE与ELEMENT子句	NO	
X160	注册XML模式的基本信息模式	NO	
X161	注册XML模式的高级信息模式	NO	
X170	XML null处理选项	NO	
X171	NIL ON NO CONTENT选项	NO	
X181	XML(DOCUMENT (UNTYPED))类型	NO	
X182	XML(DOCUMENT (ANY))类型	NO	
X190	XML(SEQUENCE) type	NO	
X191	XML(DOCUMENT (XMLSCHEMA))类 型	NO	
X192	XML(CONTENT (XMLSCHEMA))类型	NO	
X200	XMLQuery	NO	
X201	XMLQuery: RETURNING CONTENT	NO	
X202	XMLQuery: RETURNING SEQUENCE	NO	
X203	XMLQuery: 传递上下文条目	NO	
X204	XMLQuery: 初始化XQuery变量	NO	
X205	XMLQuery: EMPTY ON EMPTY选项	NO	
X206	XMLQuery: NULL ON EMPTY选项	NO	
X211	支持XML 1.1	NO	
X221	XML传递机制BY VALUE	NO	
X222	XML传递机制BY REF	NO	
X231	XML (CONTENT (UNTYPED)) 类型	NO	
X232	XML (CONTENT (ANY)) 类型	NO	
X241	在XML发布中返回内容	NO	
X242	在XML发布中返回序列	NO	
X251	XML (DOCUMENT (UNTYPED)) 类 型的持久XML值	NO	
X252	XML (DOCUMENT (ANY)) 类型的持 久XML值	NO	
X253	XML (CONTENT (UNTYPED)) 类型 的持久XML值	NO	
X254	XML (CONTENT (ANY)) 类型的持 久XML值	NO	
X255	XML (SEQUENCE) 类型的持久XML值	NO	
X256	XML (DOCUMENT (XMLSCHEMA)) 类型的持久XML值	NO	
X257	XML的持久XML值 (CONTENT (XMLSCHEMA))	NO	
X260	XML类型: ELEMENT子句	NO	
X261	XML类型: 不带ELEMENT子句 的NAMESPACE	NO	
X263	XML类型: NO NAMESPACE与ELEMENT子句	NO	
X264	XML type: schema位置	NO	

X271	XMLValidate: 数据驱动的案例	NO	
X272	XMLValidate: ACCORDING TO子句	NO	
X273	XMLValidate: ELEMENT子句	NO	
X274	XMLValidate: schema位置	NO	
X281	XMLValidate: 具有DOCUMENT选项	NO	
X282	带有CONTENT选项的XMLValidat	NO	
X283	带有SEQUENCE选项的XMLValidat	NO	
X284	不带ELEMENT子句的XMLValidate NAMESPACE	NO	
X286	XMLValidate: 带ELEMENT子句的NO NAMESPACE	NO	
X300	XMLTable	NO	
X301	XMLTable: 派生列列表选项	NO	
X302	XMLTable: ordinality列选项	NO	
X303	XMLTable: 列默认选项	NO	
X304	XMLTable: 传递上下文条目	NO	
X305	XMLTable: 初始化XQuery变量	NO	
X400	名称和标识符映射	NO	

Parent topic: [Greenplum数据库参考指南](#)

Greenplum数据库® 6.0文档

- 参考指南

- SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

- 系统目录参考

- gp_toolkit管理模式

- gpperfmon 数据库

- Greenplum 数据库数据类型

字符集支持

- 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

该参考列出并描述了Greenplum中可设置的环境变量。如果想为所有用户设置这些环境变量，需要在用户的启动shell配置文件（如`~/.bashrc`或`~/.bash_profile`或者在`/etc/profile`中）进行设置这些环境变量。

- 必需的环境变量
- 可选的环境变量

Parent topic: [Greenplum数据库参考指南](#)

必需的环境变量

Note: `GPHOME`, `PATH`和`LD_LIBRARY_PATH`能够通过引入Greenplum安装目录下的`greenplum_path.sh`文件来设置。

GPHOME

该参数指明了Greenplum安装位置。例如：

```
GPHOME=/usr/local/greenplum-db-<version>
export GPHOME
```

PATH

PATH环境变量指出Greenplum的bin目录所在位置。例如：

```
PATH=$GPHOME/bin:$PATH
export PATH
```

LD_LIBRARY_PATH

LD_LIBRARY_PATH环境变量指出了Greenplum/PostgreSQL库文件所在位置。例如：

[Greenplum的PL/Perl语言](#)

```
LD_LIBRARY_PATH=$GPHOME/lib  
export LD_LIBRARY_PATH
```

MASTER_DATA_DIRECTORY

该变量指出在master数据目录中通过gpinitcluster创建的目录。例如：

```
MASTER_DATA_DIRECTORY=/data/master/gpseg-1  
export MASTER_DATA_DIRECTORY
```

可选的环境变量

以下是标准的PostgreSQL环境变量，它们也在Greenplum数据库中被识别。为方便起见，您可能希望将与连接相关的环境变量添加到配置文件中，因此您无需在命令行上为客户端连接键入这么多选项。请注意，这些环境变量应仅在Greenplum数据库master主机上设置。

PGAPPNAME

应用的名称，通常当一个应用连接到服务器时设置的。该名称在活动视图和日志条目中有显示。PGAPPNAME环境变量和application_name连接参数有相同的行为。application_name的默认值是pgsql。该名称不能超过63个字符。

PGDATABASE

默认连接到的数据库名称。

PGHOST

Greenplum数据库的Master主机名称。

PGHOSTADDR

Master主机的数字IP地址。 设置该变量可以用来代替或者在PGHOST之上从而避免DNS查询的过多开销。

PGPASSWORD

服务器要求密码验证时使用的密码。出于安全原因，不建议使用此环境变量（某些操作系统允许非root用户通过ps查看进程环境变量）。而是考虑使用~/.pgpass文件。

PGPASSFILE

用于查找的密码文件的名称。如果未设置，则默认为~/.pgpass。有关详细信息，请参阅PostgreSQL文档中有关[密码文件](#)的主题。

PGOPTIONS

为Greenplum数据库master服务器设置其他配置参数。

PGPORT

master主机上Greenplum数据库服务器的端口号。默认端口为5432。

PGUSER

用于连接的Greenplum数据库用户名。

PGDATESTYLE

设置会话的日期/时间表示的默认样式。（相当于SET datestyle TO...）

PGTZ

设置会话的默认时区。 (相当于SET timezone TO...)

PGCLIENTENCODING

设置会话的默认客户端字符集编码。 (相当于SET client_encoding TO...)

Greenplum数据库® 6.0文档

- 参考指南

- SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

- 保留标识符和SQL关键字**

- 系统目录参考

- gp_toolkit管理模式

- gpperfmon 数据库

- Greenplum 数据库数据类型

字符集支持

- 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

本主题描述了Greenplum数据库保留标识符和对象名称，以及Greenplum数据库和PostgreSQL命令解析器识别的SQL关键字。

保留标识符

在Greenplum数据库系统中，以gp_和pg_开头的名称是保留的，不应作用于用户创建的对象的名称，例如表，视图和函数。

资源组名称admin_group, default_group和none都是保留的。资源队列名称pg_default是保留的。

表空间名称pg_default和pg_global是保留的。

角色名称gpadmin和gpmon是保留的。gpadmin是默认的Greenplum数据库超级用户角色。gpmon角色拥有gpperfmon数据库。

在数据文件中，分隔字段（列）和行的字符具有特殊含义。如果它们出现在数据中，则必须将它们转义，以便Greenplum数据库将它们视为数据而不是分隔符。反斜杠字符（\）是默认的转义字符。有关详细信息，请参见[转义](#)。

有关SQL标识符，常量，运算符和表达式的更多信息，请参阅PostgreSQL文档中的[SQL语法](#)。

SQL关键词

[Table 1](#) 列出了所有token，这些token是Greenplum数据库6和PostgreSQL 9.4中的关键词。

ANSI SQL区分保留和未保留的关键词。根据标准，保留关键词是唯一真正的关键词；永远不允许他们作为标识符。未保留的关键词在特定上下文中仅具有特殊含义，并且可以在其他上下文中用作标识符。大多数未保留的关键词实际上是SQL指定的内置表和函数的名称。无保留关键词的概念基本上只存在于声明在某些上下文中某些预定义含义附加到单词上。

在Greenplum数据库和PostgreSQL解析器中，有几种不同类别的token，从永远不能用作标识符的token到与普通标识符相比在解析

Greenplum的PL/Perl语言

器中完全没有特殊状态的token。（后者通常是SQL指定的函数的情况。）即使保留的关键字也没有完全保留，但可以用作列标签（例如，`SELECT 55 AS CHECK`，即使`CHECK`是保留关键字）。

Table 1 将那些解析器明确知道但允许作为列或表名称的关键字归类为“unreserved”。一些未经保留的关键字不能用作函数或数据类型名称，并相应标记。（这些单词中的大部分代表具有特殊语法的内置函数或数据类型。函数或类型仍然可用，但用户无法重新定义。）标记为“reserved”的关键字不允许作为列或表名。一些保留的关键字可以作为函数或数据类型的名称；这也显示在表中。如果没有这么标记，则保留的关键字仅允许作为“AS”列标签名称。

如果您为包含任何列出的关键字作为标识符的命令获得虚假解析器错误，您应该尝试引用标识符以查看问题是否消失。

在研究表之前，请注意关键字未保留的事实并不意味着未实现与该字相关的特性。相反，关键词的存在并不表示特性存在。

Table 1. SQL Key Words

关键字	Greenplum数据库	PostgreSQL 9.4
ABORT	unreserved	unreserved
ABSOLUTE	unreserved	unreserved
ACCESS	unreserved	unreserved
ACTION	unreserved	unreserved
ACTIVE	unreserved	
ADD	unreserved	unreserved
ADMIN	unreserved	unreserved
AFTER	unreserved	unreserved
AGGREGATE	unreserved	unreserved
ALL	reserved	

[Greenplum数据库® 6.0文档](#)[参考指南](#)[Greenplum database 5管理员指南](#)[Greenplum database 4管理员指南](#)[FAQ](#)

如果您发现翻译不妥之处，欢迎点击“[反馈](#)”按钮提交详细信息

该参考文献描述了Greenplum数据库系统目录表和视图。以gp_为前缀的系统表与Greenplum数据库的并行特性有关。以pg_为前缀的表是Greenplum数据库中支持的标准PostgreSQL系统目录表，或与Greenplum数据库性能有关，以提升PostgreSQL的数据仓库工作负载。请注意，Greenplum数据库的全局系统目录位于主实例上。

Warning: 不支持对Greenplum数据库系统目录表或视图的更改。如果更改目录表或视图，则必须重新初始化并还原集群。

- [系统表](#)
- [系统视图](#)
- [系统目录定义](#)

Parent topic: [Greenplum数据库参考指南](#)



Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

Greenplum数据库提供了一个名为gp_toolkit的管理模式，可用于查询系统catalog，日志文件和操作环境中的系统状态信息。gp_toolkit模式包含许多视图，您可以使用SQL命令访问这些视图。尽管某些对象可能需要超级用户权限，但所有数据库用户都可以访问gp_toolkit模式。为了方便起见，您可能需要将gp_toolkit模式添加到模式搜索路径。例如：

```
=> ALTER ROLE myrole SET search_path TO myschema, gp_toolkit;
```

本文档介绍了gp_toolkit中最有用的视图。您可能会注意到gp_toolkit模式中的其他对象（视图，函数和外部表）在本文档中未描述（这些是本节中描述的视图的支持对象）。

Warning: 不要在gp_toolkit模式中更改数据库对象。不要在模式中创建数据库对象。对模式对象的更改可能会影响模式对象返回的管理信息的准确性。备份数据库，然后使用gpbackup和gprestore实用程序还原时，在gp_toolkit模式中所做的任何更改都将丢失。

- [检查需要例行维护的表](#)
- [检查锁](#)
- [检查追加优化表](#)
- [查看Greenplum数据库服务器日志文件](#)
- [检查服务器配置文件](#)
- [检查失败的segments](#)
- [检查资源组活动和状态](#)
- [检查资源队列活动和状态](#)
- [检查查询磁盘溢出空间使用情况](#)
- [查看用户和组（角色）](#)
- [检查数据库对象大小和磁盘空间](#)
- [检查数据分布不均](#)

Parent topic: [Greenplum数据库参考指南](#)

检查需要例行维护的表

以下视图可以帮助您识别需要日常维护的表（VACUUM和/或ANALYZE）。

- [gp_bloat_diag](#)
- [gp_stats_missing](#)

VACUUM或VACUUM FULL命令可回收已删除或过时的行所占用的磁盘空间。由于Greenplum数据库中使用了MVCC事务并发模型，即使任何新事务都不可见，被删除或更新的数据行仍占据磁盘上的物理空间。过期的行会增加磁盘上的表大小，并最终减慢表的扫描速度。

`ANALYZE`命令收集查询优化器所需的列级统计信息。Greenplum数据库使用依赖数据库统计信息的基于成本的查询优化器。准确的统计信息使查询优化器可以更好地估计选择性和查询操作检索到的行数，从而选择最有效的查询计划。

Parent topic: [gp_toolkit管理模式](#)

gp_bloat_diag

此视图显示膨胀的常规堆存储表（给定表统计信息，磁盘上的实际页面数超过了预期的页面数）。膨胀的表需要VACUUM或VACUUM FULL，才能回收已删除或过时的行所占用的磁盘空间。所有用户都可以访问该视图，但是非超级用户将只能看到他们有权访问的表。

Note: 有关返回追加优化表信息的诊断函数，请参阅[检查追加优化表](#)。

Table 1. gp_bloat_diag视图

列	描述
bdirelid	表OID。
bdinspname	模式名称。
bdirelname	表名。
bdirelpages	磁盘上实际页的数量。
bdiexppages	给定表数据的预期页数。
bdidiag	膨胀诊断消息。

gp_stats_missing

该视图显示的表没有统计信息，因此可能需要对该表运行ANALYZE。

Table 2. gp_stats_missing view

列	描述
smischema	模式名称。
smitable	表名。
smisize	该表是否有统计信息？如果该表没有在系统catalog中记录行数和行大小统计信息，则为False，这可能表明该表需要进行分析。如果表不包含任何行，这也为false。例如，分区表的父表始终为空，并且始终返回false。
smicols	表中的列数。
smirecs	表中的行数。

检查锁

当事务访问关系（例如表）时，它获取一个锁。根据获取的锁的类型，后续事务可能必须等待才能访问相同的关系。有关锁类型的更多信息，请参阅Greenplum数据库管理员指南中的“管理数据”。Greenplum数据库资源队列（用于资源管理）还使用锁来控制查询进入系统的权限。

`gp_locks_*`系列视图可以帮助诊断由于锁定而正在等待访问对象的查询和会话。

- [gp_locks_on_relation](#)
- [gp_locks_on_resqueue](#)

Parent topic: [gp_toolkit管理模式](#)

gp_locks_on_relation

此视图显示当前在某个关系上持有的所有锁，以及有关与该锁关联的查询的关联会话信息。有关锁类型的更多信息，请参阅Greenplum数据库管理员指南中的“管理数据”。所有用户都可以访问此视图，但是非超级用户将只能看到他们有权访问的关系的锁。

Table 3. `gp_locks_on_relation`视图

列	描述
lorlocktype	可锁定对象的类型：relation, extend, page, tuple, transactionid, object, userlock, resource queue, 或 advisory
lordatabase	对象所在的数据库的对象ID，如果该对象是共享对象，则为零。
lorrelname	关系的名称。
lorrelation	关系的OID。
lortransaction	受锁影响的事务ID。
lorpid	持有或正在等待此锁定的服务器进程的进程ID。如果锁由准备好的事务持有，则为NULL。
lormode	此进程持有或需要的锁定模式的名称。
lorgranted	显示是授予锁（true）还是不授予锁（false）。
lorcurrentquery	会话中的当前查询。

gp_locks_on_resqueue

Note: 仅当基于资源队列的资源管理处于活动状态时，`gp_locks_on_resqueue`视图才有效。

该视图显示了资源队列中当前持有的所有锁，以及与该锁关联的查询的关联会话信息。所有用户都可以访问此视图，但是非超级用户将只能看到与其自己的会话相关联的锁。

Table 4. gp_locks_on_resqueue视图

列	描述
lorusename	执行会话的用户名。
lorrsqname	资源队列名称。
lorlocktype	可锁定对象的类型： 资源队列
lorobjid	锁定事务的ID。
lortransaction	受锁影响的事务的ID。
lorpid	受锁影响的事务的进程ID。
lormode	此进程持有或需要的锁定模式的名称。
lorgranted	显示是授予锁 (true) 还是不授予锁 (false) 。
lorwaiting	显示会话是否正在等待。

检查追加优化表

gp_toolkit模式包含一组诊断功能，可用于调查追加优化表的状态。

创建追加优化表（或面向列的追加优化表）时，将隐式创建另一个表，其中包含有关该表当前状态的元数据。元数据包括诸如表的每个segment中的记录数之类的信息。

追加优化表可能包含不可见的行，这些行已被更新或删除，但是保留在存储中，直到使用VACUUM压缩表为止。使用辅助可见性映射表或visimap跟踪隐藏的行。

通过以下函数，您可以访问针对追加优化和面向列的表的元数据，并查看不可见的行。其中一些函数具有两个版本：一个采用表的oid，而另一个采用表的名称。后一种版本在函数名称后附加了“_name”。

Parent topic: [gp_toolkit管理模式](#)

__gp_aovisimap_compaction_info(oid)

此函数显示追加优化表的压缩信息。该信息适用于Greenplum数据库segment中存储表数据的磁盘上数据文件。您可以使用该信息来确定将通过VACUUM操作对追加优化表进行压缩的数据文件。

Note: 在VACUUM操作从数据文件中删除该行之前，已删除或已更新的数据行将占据磁盘上的物理空间，即使它们已被新事务隐藏。配置参数gp_appendonly_compaction控制VACUUM命令的功能。

下表描述了__gp_aovisimap_compaction_info 函数输出表。

Table 5. __gp_aovisimap_compaction_info输出表

列	描述

content	Greenplum数据库segment ID。
datafile	segment上数据文件的ID。
compaction_possible	该值为t或f。 值t表示执行VACUUM操作时压缩数据文件中的数据。 服务器配置参数gp_appendonly_compaction_threshold会影响此值。
hidden_tupcount	在数据文件中，隐藏（删除或更新）的行数。
total_tupcount	在数据文件中，总行数。
percent_hidden	在数据文件中，隐藏（已删除或更新）行与总行的比率（百分比）。

`__gp_aoseg_name('table_name')`

此函数返回追加优化表的磁盘段文件中包含的元数据信息。

Table 6. `__gp_aoseg_name`输出表

列	描述
segno	文件段号。
eof	该文件段的文件有效结尾。
tupcount	段中的元组总数，包括不可见的元组。
varblockcount	文件段中的varblocks总数。
eof_uncompressed	文件末尾（如果文件段未压缩）。
modcount	数据修改操作的数量。
state	文件段的状态。指示段是活动的还是压缩后可以删除的。

`__gp_aoseg_history(oid)`

此函数返回追加优化表的磁盘段文件中包含的元数据信息。它显示aoseg元信息的所有不同版本（堆元组）。数据很复杂，但是对系统有深入了解的用户可能会发现它对于调试很有用。

输入参数是追加优化表的oid。

调用`__gp_aoseg_history_name ('table_name')` 可获得与表名参数相同的结果。

Table 7. `__gp_aoseg_history`输出表

列	描述

gp_tid	元组的id。
gp xmin	最早事务的id。
gp_xmin_status	gp_xmin事务的状态。
gp_xmin_commit_	gp_xmin事务的提交分布式id。
gp xmax	最晚的事务id。
gp xmax_status	最晚的事务的状态。
gp xmax_commit_	gp xmax事务的提交分布式id。
gp_command_id	查询命令的id。
gp_infomask	包含状态信息的位图。
gp_update_tid	更新的元组的ID (如果该行已更新)。
gp_visibility	元组可见性状态。
segno	段文件中的段号。
tupcount	元组数量，包括隐藏元组。
eof	段的文件的有效结尾。
eof_uncompressed	该段的文件末尾，如果数据未压缩。
modcount	数据修改计数。
state	段的状态。

__gp_aocsseg(oid)

此函数返回包含在面向列的追加优化表的磁盘段文件中的元数据信息，不包括不可见的行。每行描述表中一列的段。

输入参数是面向列的追加优化表的oid。以表名作为参数调用 __gp_aocsseg_name ('table_name') 获得相同的结果。

Table 8. __gp_aocsseg(oid)输出表

列	描述
gp_tid	表的ID。
segno	段号。
column_num	列号。
physical_segno	段文件中段的编号。
tupcount	段中的行数，不包括隐藏的元组。
eof	段的文件的有效结尾。
eof_uncompressed	该段文件的末尾 (如果数据未压缩)。
modcount	段的数据修改操作计数。
state	段的状态。

gp_aocsseg_history(oid)

此函数返回包含在面向列的追加优化表的磁盘段文件中的元数据信息。每行描述表中一列的段。数据很复杂，但是对系统有深入了解的用户可能会发现它对于调试很有用。

输入参数是面向列的追加优化表的oid。以表名称作为参数调用gp_aocsseg_history_name ('table_name') 获得相同的结果。

Table 9. gp_aocsseg_history输出表

列	描述
gp_tid	元组的oid。
gp_xmin	最早的事务。
gp_xmin_status	gp_xmin事务的状态。
gp_xmin_	gp_xmin的文本表示形式。
gp xmax	最近的事务。
gp xmax_status	gp xmax事务的状态。
gp xmax_	gp_max的文本表示形式。
gp_command_id	在元组上运行的命令的ID。
gp_infomask	包含状态信息的位图。
gp_update_tid	新的元组的ID (如果该行已更新)。
gp_visibility	元组可见性状态。
segno	段文件中的段号。
column_num	列号。
physical_segno	包含列数据的段。
tupcount	段中的元组总数。
eof	段的文件的有效结尾。
eof_uncompressed	该段文件的末尾 (如果数据未压缩)。
modcount	数据修改操作的计数。
state	段的状态。

gp_aovisimap(oid)

此函数根据可见性映射返回元组id，段文件和每个不可见元组的行号。

输入参数是追加优化表的oid。

使用gp_aovisimap_name ('table_name') 获得与表名作为参数相同的结果。

列	描述
tid	ID

	元组。
segno	段文件的编号。
row_num	已删除或更新的行的行号。

__gp_aovisimap_hidden_info(oid)

此函数返回段文件中针对追加优化表的隐藏和可见元组数。

输入参数是追加优化表的oid。

调用__gp_aovisimap_hidden_info_name ('table_name') 可获得与表名参数相同的结果。

列	描述
segno	段文件的编号。
hidden_tupcount	段文件中隐藏元组的数量。
total_tupcount	段文件中的元组总数。

__gp_aovisimap_entry(oid)

此函数返回有关表的每个可见性映射条目的信息。

输入参数是追加优化表的oid。

调用__gp_aovisimap_entry_name ('table_name') 可获得与表名参数相同的结果。

Table 10. __gp_aovisimap_entry输出表

列	描述
segno	可见性映射条目的段号。
first_row_num	条目的第一行号。
hidden_tupcount	条目中的隐藏元组数。
bitmap	可见性位图的文本表示。

查看Greenplum数据库服务器日志文件

Greenplum数据库系统的每个组件 (master, standby, primary和mirror) 均保留其自己的服务器日志文件。使用gp_log_*系列视图，您可以对服务器日志文件执行SQL查询，以找到感兴趣的特定条目。这些视图的使用需要超级用户权限。

- [gp_log_command_timings](#)
- [gp_log_database](#)

- [gp_log_master_concise](#)
- [gp_log_system](#)

Parent topic: [gp_toolkit管理模式](#)

gp_log_command_timings

该视图使用外部表来读取主数据库上的日志文件，并报告在数据库会话中执行的SQL命令的执行时间。使用此视图需要超级用户权限。

Table 11. gp_log_command_timings视图

列	描述
logsession	会话标识符（以“con”为前缀）。
logcmdcount	会话中的命令号（以“cmd”为前缀）。
logdatabase	数据库的名称。
loguser	数据库用户的名称。
logpid	进程ID（前缀为“p”）。
logtimemin	此命令的第一条日志消息的时间。
logtimemax	该命令的最后一条日志消息的时间。
logduration	语句从开始到结束时间的持续时间。

gp_log_database

该视图使用外部表读取整个Greenplum系统的服务器日志文件（master, primary和mirror），并列出与当前数据库关联的日志条目。可以通过会话ID（logsession）和命令ID（logcmdcount）标识关联的日志条目。使用此视图需要超级用户权限。

Table 12. gp_log_database视图

列	描述
logtime	日志消息的时间戳。
loguser	数据库用户的名称。
logdatabase	数据库的名称。
logpid	关联的进程ID（前缀为“p”）。
logthread	关联的线程数（以“th”为前缀）。
loghost	segment或master主机名。
logport	segment或master端口号。
logsessiontime	会话连接打开时间。
logtransaction	全局事务ID。

logsession	会话标识符 (以"con"为前缀)。
logcmdcount	会话中的命令号 (以"cmd"为前缀)。
logsegment	segment内容标识符 (对于primary标识符, 前缀为"seg", 对于mirror, 前缀为"mir"。 master服务器的内容ID始终为-1)。
logslice	切片ID (正在执行的查询计划的一部分)。
logdistxact	分布式事务ID。
loglocalxact	本地事务ID。
logsubxact	子事务ID。
logseverity	LOG, ERROR, FATAL, PANIC, DEBUG1或DEBUG2。
logstate	与日志消息关联的SQL状态代码。
logmessage	日志或错误消息文本。
logdetail	与错误消息关联的详细消息文本。
loghint	与错误消息关联的提示消息文本。
logquery	内部生成的查询文本。
logquerypos	游标索引到内部生成的查询文本中。
logcontext	生成此消息的上下文。
logdebug	用于调试的带有详细信息的查询字符串。
logcursorpos	查询字符串中的游标索引。
logfunction	生成此消息的函数。
logfile	生成此消息的日志文件。
logline	日志文件中生成此消息的行。
logstack	与该消息关联的堆栈跟踪的全文。

gp_log_master_concise

该视图使用外部表从master日志文件读取日志字段的子集。 使用此视图需要超级用户权限。

Table 13. gp_log_master_concise视图

列	描述
logtime	日志消息的时间戳。
logdatabase	数据库的名称。
logsession	会话标识符 (以"con"为前缀)。
logcmdcount	会话中的命令号 (以"cmd"为前缀)。
logmessage	日志或错误消息文本。

gp_log_system

该视图使用一个外部表来读取整个Greenplum系统 (master, segment和mirror) 的服务器日志文件，并列出所有日志条目。可以通过会话ID (logsession) 和命令ID (logcmdcount) 标识关联的日志条目。使用此视图需要超级用户权限。

Table 14. gp_log_system视图

列	描述
logtime	日志消息的时间戳。
loguser	数据库用户的名称。
logdatabase	数据库的名称。
logpid	关联的进程ID (前缀为"p")。
logthread	关联的线程数 (以"th"为前缀)。
loghost	segment或master主机名。
logport	segment或master端口号。
logsessiontime	会话连接打开时间。
logtransaction	全局事务ID。
logsession	会话标识符 (以"con"为前缀)。
logcmdcount	会话中的命令号 (以"cmd"为前缀)。
logsegment	segment内容标识符 (对于primary标识符, 前缀为"seg", 对于mirror, 前缀为"mir"。 master服务器的内容ID始终为-1)。
logslice	切片ID (正在执行的查询计划的一部分)。
logdistxact	分布式事务ID。
loglocalxact	本地事务ID。
logsubxact	子事务ID。
logseverity	LOG, ERROR, FATAL, PANIC, DEBUG1或DEBUG2。
logstate	与日志消息关联的SQL状态代码。
logmessage	日志或错误消息文本。
logdetail	与错误消息关联的详细消息文本。
loghint	与错误消息关联的提示消息文本。
logquery	内部生成的查询文本。
logquerypos	游标索引到内部生成的查询文本中。
logcontext	生成此消息的上下文。
logdebug	用于调试的带有详细信息的查询字符串。
logcursorpos	查询字符串中的游标索引。
logfunction	生成此消息的函数。
logfile	生成此消息的日志文件。
logline	日志文件中生成此消息的行。

logstack

与该消息关联的堆栈跟踪的全文。

检查服务器配置文件

Greenplum数据库系统的每个组件（master, standby, primary和mirror）都有其自己的服务器配置文件（postgresql.conf）。以下gp_toolkit对象可用于检查系统中所有primary的postgresql.conf文件的参数设置：

- [gp_param_setting\('parameter_name'\)](#)
- [gp_param_settings_seg_value_diffs](#)

Parent topic: [gp_toolkit管理模式](#)

gp_param_setting('parameter_name')

此函数采用服务器配置参数的名称，并返回master和每个活动segment的postgresql.conf值。所有用户均可使用此函数。

Table 15. gp_param_setting('parameter_name')函数

列	描述
paramsegment	segment内容ID（仅显示活动segment）。master内容ID始终为-1。
paramname	参数名称。
paramvalue	参数值。

示例：

```
SELECT * FROM gp_param_setting('max_connections');
```

gp_param_settings_seg_value_diffs

分类为本地参数的服务器配置参数（意味着每个segment都从其自己的postgresql.conf文件获取参数值），应在所有segment上进行相同的设置。该视图显示的本地参数设置不一致。不包括应该具有不同值的参数（例如port）。所有用户均可访问此视图。

Table 16. gp_param_settings_seg_value_diffs视图

列	描述
psdname	参数的名称。

psdvalue	参数的值。
psdcount	具有此值的segment数。

检查失败的segments

[gp_pgdatabase_invalid](#) 视图可用于检查宕机的segment。

Parent topic: [gp_toolkit管理模式](#)

gp_pgdatabase_invalid

此视图显示有关在系统目录中标记为宕机的segment的信息。所有用户均可访问此视图。

Table 17. gp_pgdatabase_invalid视图

列	描述
pgdbidbid	segment的dbid。每个segment都有唯一的dbid。
pgdbiisprimary	segment的角色是否是primary? (t或f)
pgdbcontent	该segment的内容ID。primary和mirror将具有相同的内容ID。
pgdbvalid	segment是否在线有效? (t或f)
pgdbdefinedprimary	在系统初始化时，此segment是否分配为primary? (t或f)

检查资源组活动和状态

Note: 仅当基于资源组的资源管理处于活动状态时，本节中描述的资源组活动和状态视图才有效。

资源组管理事务，以避免耗尽系统CPU和内存资源。每个数据库用户都被分配了一个资源组。在运行事务之前，Greenplum数据库会根据为用户资源组配置的限制评估用户提交的每个事务。

您可以使用[gp_resgroup_config](#)视图检查每个资源组的配置。您可以使用[gp_resgroup_status*](#)视图显示当前事务状态和每个资源组的资源使用情况。

- [gp_resgroup_config](#)
- [gp_resgroup_status](#)
- [gp_resgroup_status_per_host](#)
- [gp_resgroup_status_per_segment](#)

Parent topic: [gp_toolkit管理模式](#)

gp_resgroup_config

使用`gp_resgroup_config`视图，管理员可以查看资源组的当前CPU，内存和并发限制。该视图还显示建议的限制设置。更改限制后，建议的限制将不同于当前限制，但是无法立即应用新值。

This view is accessible to all users.

Table 18. `gp_resgroup_config`

列	描述
groupid	资源组的ID。
groupname	资源组的名称。
concurrency	为资源组指定的并发 (CONCURRENCY) 值。
proposed_concurrency	资源组的待定并发值。
cpu_rate_limit	为资源组指定的CPU限制 (CPU_RATE_LIMIT) 值或-1。
memory_limit	为资源组指定的内存限制 (MEMORY_LIMIT) 值。
proposed_memory_limit	资源组的待定内存限制值。
memory_shared_quota	为资源组指定的共享内存配额 (MEMORY_SHARED_QUOTA) 值。
proposed_memory_shared_quota	资源组的待定共享内存配额值。
memory_spill_ratio	为资源组指定的内存溢出率 (MEMORY_SPILL_RATIO) 值。
proposed_memory_spill_ratio	资源组的待定内存溢出率值。
memory_auditor	资源组的内存审核。
cpuset	为资源组保留的CPU核心，或-1。

gp_resgroup_status

使用`gp_resgroup_status`视图，管理员可以查看资源组的状态和活动。它显示了每个资源组正在等待运行的查询数量以及系统中当前正在活动的查询数量。该视图还显示资源组的当前内存和CPU使用率。

Note: 资源组使用主机系统上配置的Linux控制组 (cgroup)。cgroup用于管理主机系统资源。当资源组使用作为嵌套cgroup组的一部分的cgroup时，资源组限制是相对于父cgroup分配的。有关嵌套cgroup和Greenplum数据库资源组限制的信息，请参阅[理解角色和组件资源组](#)。

所有用户均可访问此视图。

Table 19. gp_resgroup_status视图

列	描述
rsgname	资源组的名称。
groupid	资源组的ID。
num_running	资源组中当前正在执行的事务数。
num_queueing	资源组当前排队的事务数。
num_queued	自上次启动Greenplum数据库集群以来，资源组的排队事务总数，不包括num_queueing。
num_executed	自Greenplum数据库集群上次启动以来，资源组中已执行事务的总数，不包括num_running。
total_queue_duration	自上次启动Greenplum数据库集群以来，所有事务排队的总时间。
cpu_usage	每个Greenplum数据库segment主机上资源组的实时CPU使用率。
memory_usage	每个Greenplum数据库segment主机上资源组的实时内存使用情况。

cpu_usage字段是JSON格式的key: value字符串，用于为每个资源组标识每个segment的CPU使用率百分比。key是segment ID，value是segment主机上资源组的CPU使用率百分比。在segment主机上运行的所有segment的CPU总使用率不应超过gp_resource_group_cpu_limit。示例cpu_usage列输出：

```
{"-1":0.01, "0":0.31, "1":0.31}
```

在此示例中，segment 0和segment 1在同一主机上运行；它们的CPU使用率是相同的。

memory_usage字段也是JSON格式的key: value字符串。字符串内容因资源组的类型而异。对于分配给角色的每个资源组（默认内存审核器vmtracker），此字符串标识每个segment上已使用，可用，已授予和建议的固定和共享内存配额分配。key是segment ID，value是以MB为单位显示的内存值。下面的示例显示分配给角色的资源组的单个segment的memory_usage列输出：

```
"0":{ "used":0, "available":76, "quota_used":-1, "quota_available":60,
"quota_granted":60, "quota_proposed":60, "shared_used":0,
"shared_available":16, "shared_granted":16, "shared_proposed":16}
```

对于分配给外部组件的每个资源组，memory_usage JSON格式的字符串标识了每个segment上使用的内存和内存限制。以下示例显示单个segment的外部组件资源组的memory_usage列输出：

```
"1":{ "used":11, "limit_granted":15}
```

Note: 请参阅下文所述

的gp_resgroup_status_per_host和gp_resgroup_status_per_segment视图，以更友好地显示CPU和内存使用情况。

gp_resgroup_status_per_host

[gp_resgroup_status_per_host](#)视图以主机为单位显示每个资源组的实时CPU和内存使用率 (MB)。该视图还显示主机上每个资源组的可用和已授予组的固定和共享内存。

Table 20. gp_resgroup_status_per_host视图

列	描述
rsgname	资源组的名称。
groupid	资源组的ID。
hostname	segment主机的主机名。
cpu	主机上资源组的实时CPU使用率。
memory_used	主机上资源组的实时内存使用情况。此总数包括资源组固定和共享内存。它还包括资源组使用的全局共享内存。
memory_available	主机上可用的资源组的未使用的固定和共享内存。此总数不包括可用资源组全局共享内存。
memory_quota_used	主机上资源组的实时固定内存使用情况。
memory_quota_available	主机上资源组可用的固定内存。
memory_quota_proposed	Greenplum数据库已分配给主机上资源组的固定内存总量。
memory_shared_used	主机上的资源组使用的组共享内存。如果资源组使用任何全局共享内存，则此数量也将包括在总数中。
memory_shared_available	主机上资源组可用的组共享内存量。资源组全局共享内存不包括在此总计中。
memory_shared_granted	Greenplum数据库已分配给主机上资源组的组共享内存部分。资源组全局共享内存不包括在此值中。
memory_shared_proposed	主机上的资源组请求的组共享内存的总数。

如果群集没有足够的内存来分配，则memory_shared_granted值可能小于memory_shared_proposed。当资源组使用全局共享内存时，它可能大于memory_shared_proposed。随着时间的推移，memory_shared_granted和memory_shared_proposed应该达到相同的值。

gp_resgroup_status_per_host视图的示例输出：

rsgname	groupid	hostname	cpu	memory_used	memory_available	memory_quota_used	memory_quota_available	memory_quota_proposed	memory_shared_used	memory_shared_available	memory_shared_granted	memory_shared_proposed
admin_group	6438	my-desktop	0.84	1	271	136	136	136	136	136	0	0
	68		68			136						
	136			136			136					
default_group	6437	my-desktop	0.00	0	816	400	400	400	400	400	0	0
	0		400			416						
	416			416							416	
(2 rows)												

gp_resgroup_status_per_segment

[gp_resgroup_status_per_segment](#)视图基于每个segment实例和每个主机显示每个资源组的实时CPU和内存使用率 (MB)。该视图还显示主机上每个资源组和segment实例组合的可用和已授予组的固定和共享内存。

Table 21. gp_resgroup_status_per_segment视图

列	描述
rsgname	资源组的名称。
groupid	资源组的ID。
hostname	segment主机的主机名。
segment_id	segment主机上segment实例的内容ID。
cpu	主机上的segment实例的资源组的实时CPU使用率。
memory_used	主机上的segment实例的资源组的实时内存使用情况。此总数包括资源组固定和共享内存。它还包括资源组使用的全局共享内存。
memory_available	主机上segment实例的资源组的未使用的固定和共享内存。
memory_quota_used	主机上segment实例的资源组的实时固定内存使用情况。
memory_quota_available	资源组可用于主机上的segment实例的固定内存。
memory_quota_proposed	Greenplum数据库已为主机上的segment分配给资源组的固定内存总量。对于主机上的所有资源组和segment实例组合，此值都相同。

memory_shared_used	资源组用于主机上的segment实例的组共享内存。
memory_shared_available	主机上的segment实例可用的组共享内存量。 资源组全局共享内存不包括在此总计中。
memory_shared_granted	Greenplum数据库已为主机上的segment实例分配给资源组的组共享内存部分。 资源组全局共享内存不包括在此值中。
memory_shared_proposed	主机上的资源组请求的组共享内存的总数。 对于主机上的所有资源组和segment实例组合，此值都相同。

该视图的查询输出与gp_resgroup_status_per_host视图的查询输出相似，并针对每个主机上的每个segment实例划分CPU和内存（已使用和可用）。

检查资源队列活动和状态

Note: 仅当基于资源队列的资源管理处于活动状态时，本节中描述的资源队列活动和状态视图才有效。

资源队列的目的是在任何给定时间限制系统中活动查询的数量，以避免耗尽内存、CPU和磁盘I/O等系统资源。所有数据库用户都分配给一个资源队列，并且在运行该用户之前，将首先针对资源队列限制评估用户提交的每个语句。gp_resq_*系列视图可用于检查通过它们各自的资源队列当前提交给系统的语句的状态。请注意，超级用户执行的语句免于资源排队。

- [gp_resq_activity](#)
- [gp_resq_activity_by_queue](#)
- [gp_resq_priority_statement](#)
- [gp_resq_role](#)
- [gp_resqueue_status](#)

Parent topic: [gp_toolkit管理模式](#)

gp_resq_activity

对于具有活动工作负载的资源队列，此视图为通过资源队列提交的每个活动语句显示一行。所有用户均可访问此视图。

Table 22. gp_resq_activity视图

列	描述
resqprocid	分配给该语句的进程ID（在master上）。
resqrole	用户名。

resqid	资源队列对象标识。
resqname	资源队列名称。
resqstart	已向系统发布时间声明。
resqstatus	语句状态: running,waiting或cancelled。

gp_resq_activity_by_queue

对于具有活动工作负载的资源队列，此视图显示队列活动的摘要。所有用户均可访问此视图。

Table 23. gp_resq_activity_by_queue视图

列	描述
resqid	资源队列对象标识。
resqname	资源队列名称。
resqlast	发布到队列的最后一条语句的时间。
resqstatus	最后一条语句的状态: running,waiting或cancelled。
resqtotal	此队列中的语句总数。

gp_resq_priority_statement

该视图显示了Greenplum数据库系统中当前正在运行的所有语句的资源队列优先级，会话ID和其他信息。所有用户均可访问此视图。

Table 24. gp_resq_priority_statement视图

列	描述
rqpdatname	会话连接到的数据库名称。
rquseuname	执行语句的用户。
rqpession	会话ID。
rqpcommand	该会话中的语句编号（命令ID和会话ID唯一标识一条语句）。
rppriority	此语句的资源队列优先级 (MAX, HIGH, MEDIUM, LOW)。
rqpweight	与该语句的优先级关联的整数值。
rqpquery	语句的查询文本。

gp_resq_role

此视图显示与角色关联的资源队列。所有用户均可访问此视图。

Table 25. gp_resq_role视图

列	描述
rrrolname	角色（用户）名称。
rrrsqname	分配给此角色的资源队列名称。如果尚未将角色明确分配给资源队列，则它将位于默认资源队列 (<i>pg_default</i>) 中。

gp_resqueue_status

该视图使管理员可以查看资源队列的状态和活动。它显示了特定资源队列中有多少查询正在等待运行以及当前系统中有多少查询处于活动状态。

Table 26. gp_resqueue_status视图

列	描述
queueid	资源队列的ID。
rsqname	资源队列的名称。
rsqcountlimit	资源队列的活动查询阈值。值-1表示没有限制。
rsqcountvalue	资源队列中当前正在使用的活动查询槽的数量。
rsqcostlimit	资源队列的查询成本阈值。值-1表示没有限制。
rsqcostvalue	资源队列中当前所有语句的总成本。
rsqmemorylimit	资源队列的内存限制。
rsqmemoryvalue	资源队列中当前所有语句使用的总内存。
rsqwaiters	资源队列中当前正在等待的语句数。
rsqholders	此资源队列中当前在系统上运行的语句数。

检查查询磁盘溢出空间使用情况

*gp_workfile_** 视图显示有关当前正在使用磁盘溢出空间的所有查询的信息。如果Greenplum数据库没有足够的内存来在内存中执行查询，则会在磁盘上创建工作文件。此信息可用于故障排除和优化查询。视图中的信息还可用于指定Greenplum数据库配置参数*gp_workfile_limit_per_query*和*gp_workfile_limit_per_segment*的值。

- [gp_workfile_entries](#)
- [gp_workfile_usage_per_query](#)

- [gp_workfile_usage_per_segment](#)

Parent topic: [gp_toolkit管理模式](#)

gp_workfile_entries

该视图为每个运算符在当前时间使用一个磁盘空间来存储segment上工作文件的每一行。所有用户均可访问该视图，但是非超级用户只能查看其有权访问的数据库信息。

Table 27. gp_workfile_entries

列	类型	参考	描述
command_cnt	integer		查询的命令ID。
content	smallint		segment实例的内容标识符。
current_query	text		进程正在执行的查询。
datname	name		Greenplum数据库名称。
directory	text		工作文件的路径。
optype	text		创建工作文件的查询运算符类型。
procpid	integer		服务器进程的进程ID。
sess_id	integer		会话ID。
size	bigint		工作文件的大小（以字节为单位）。
numfiles	bigint		创建的文件数。
slice	smallint		查询计划切片。查询计划中正在执行的部分。
state	text		创建工作文件的查询状态。
username	name		角色名称。
workmem	integer		分配给运算符的内存量，以KB为单位。

gp_workfile_usage_per_query

对于当前segment上使用工作文件的磁盘空间的每个查询，此视图都包含一行。所有用户均可访问该视图，但是非超级用户只能查看其有权访问的数据库信息。

Table 28. gp_workfile_usage_per_query

列	类型	参考	描述
command_cnt	integer		查询的命令ID。
content	smallint		segment实例的内容标识符。
current_query	text		进程正在执行的查询。
datname	name		Greenplum数据库名称。
procpid	integer		服务器进程的进程ID。
sess_id	integer		会话ID。
size	bigint		工作文件的大小（以字节为单位）。
numfiles	bigint		创建的文件数。
state	text		创建工作文件的查询状态。
username	name		角色名称。

gp_workfile_usage_per_segment

该视图为每个segment包含一行。 每行显示当前时间segment上用于工作文件的磁盘空间总量。 所有用户均可访问该视图，但是非超级用户只能查看其有权访问的数据库信息。

Table 29. gp_workfile_usage_per_segment

列	类型	参考	描述
content	smallint		segment实例的内容标识符。
size	bigint		segment上工作文件的总大小。
numfiles	bigint		创建的文件数。

查看用户和组（角色）

将用户（角色）分组在一起以简化对象特权的管理通常很方便：通过这种方式，可以将特权授予或撤消整个组。在Greenplum数据库中，这是通过创建代表组的角色，然后将组角色的成员身份授予各个用户角色来完成的。

[gp_roles_assigned](#) 视图可用于查看系统中的所有角色及其分配的成员（如果角色也是组角色）。

Parent topic: [gp_toolkit管理模式](#)

gp_roles_assigned

此视图显示系统中的所有角色及其分配的成员（如果角色也是组角色）。所有用户均可访问此视图。

Table 30. gp_roles_assigned视图

列	描述
raroleid	角色对象ID。如果此角色具有成员（用户），则将其视为组角色。
rarolename	角色（用户或组）名称。
ramemberid	属于此角色成员的角色对象ID。
ramembername	属于此角色的角色的名称。

检查数据库对象大小和磁盘空间

gp_size_*系列视图可用于确定分布式Greenplum数据库，模式，表或索引的磁盘空间使用情况。以下视图计算了所有primary中对象的总大小（大小计算中不包括mirror）。

- [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](#)

表和索引大小视图按对象ID（而不是名称）列出关系。要按名称检查表或索引的大小，必须在pg_class表中查找关系名称（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;
```

Parent topic: [gp_toolkit管理模式](#)

gp_size_of_all_table_indexes

此视图显示表的所有索引的总大小。所有用户都可以访问此视图，但是非超级用户将只能看到他们有权访问的关系。

Table 31. gp_size_of_all_table_indexes视图

列	描述
soatoid	表的对象ID
soatisize	所有表索引的总大小 (以字节为单位)
soatschemaname	模式名称
soat tablename	表名

gp_size_of_database

此视图显示数据库的总大小。所有用户都可以访问该视图，但是非超级用户将只能看到他们有权访问的数据库。

Table 32. gp_size_of_database视图

列	描述
sodddatname	数据库名称
sodddatsize	数据库大小 (以字节为单位)

gp_size_of_index

此视图显示索引的总大小。所有用户都可以访问此视图，但是非超级用户将只能看到他们有权访问的关系。

Table 33. gp_size_of_index视图

列	描述
soioid	索引的对象ID
soitableoid	索引所属表的对象ID
soisize	索引大小 (以字节为单位)
soiindexschemaname	索引模式的名称
soiindexname	索引名称
soitableschemaname	表模式的名称
soitablelename	表名称

gp_size_of_partition_and_indexes_disk

此视图显示分区子表及其索引在磁盘上的大小。所有用户均可访问此视图，但是非超

级用户将只能看到他们有权访问的关系。

Table 34. gp_size_of_partition_and_indexes_disk视图

列	描述
sopaidparentoid	父表的对象ID
sopaidpartitionoid	分区表的对象ID
sopaidpartitiontablesize	分区表大小 (以字节为单位)
sopaidpartitionindexessize	该分区上所有索引的总大小
Sopaidparentscemaname	父模式的名称
Sopaidparenttablename	父表的名称
Sopaidpartitionschemaname	分区模式的名称
sopaidpartitiontablename	分区表的名称

gp_size_of_schema_disk

此视图显示当前数据库中公共模式和用户创建的模式的模式大小。所有用户都可以访问此视图，但是非超级用户将只能看到他们有权访问的架构。

Table 35. gp_size_of_schema_disk视图

列	描述
sosdmsp	模式名称
sosdschematablesize	模式中表的总大小 (以字节为单位)
sosdschemaidxsize	模式中索引的总大小 (以字节为单位)

gp_size_of_table_and_indexes_disk

此视图显示表及其索引在磁盘上的大小。所有用户都可以访问此视图，但是非超级用户将只能看到他们有权访问的关系。

Table 36. gp_size_of_table_and_indexes_disk视图

列	描述
sotaidoid	父表的对象ID
sotaidthablesize	表的磁盘大小
sotaidthidxsize	表上所有索引的总大小
sotaidschemaname	模式名称
sotaidth tablename	表名

gp_size_of_table_and_indexes_licensing

此视图显示用于许可目的的表及其索引的总大小。 使用此视图需要超级用户权限。

Table 37. gp_size_of_table_and_indexes_licensing视图

列	描述
sotailoid	表的对象ID
sotailtablesize	表的总磁盘大小
sotailtablesizeuncompressed	如果该表是压缩的追加优化表，则以字节为单位显示未压缩的表大小。
sotailindexessize	表中所有索引的总大小
sotailschemaname	模式名称
sotailtablename	表名

gp_size_of_table_disk

此视图显示磁盘上表的大小。 所有用户均可访问此视图，但是非超级用户将只能看到他们有权访问的表

Table 38. gp_size_of_table_disk视图

列	描述
sotdoid	表的对象ID
sotysize	表的大小（以字节为单位）。 该大小仅是主表的大小。 该大小不包括辅助对象（例如，超大（toast）属性）或AO表的其他存储对象。
sotdtoastsize	TOAST表的大小（超大属性存储）（如果有）。
sotdadditionalsize	反映追加优化（AO）表的段和块目录表的大小。
sotdschemaname	模式名称
sotdtablename	表名

gp_size_of_table_uncompressed

此视图显示追加优化（AO）表的未压缩表大小。 否则，将显示磁盘上的表大小。 使用此视图需要超级用户权限。

Table 39. gp_size_of_table_uncompressed视图

列	描述
sotuoid	表的对象ID
sotusize	如果表是压缩的AO表，则表示表的未压缩大小（以

	字节为单位)。否则, 为磁盘上表的大小。
sotuschemaname	模式名称
sotutablename	表名

gp_disk_free

该外部表在活动segment主机上运行df (disk free) 命令, 并报告结果。非活动mirror不包括在计算中。使用此外部表需要超级用户权限。

Table 40. gp_disk_free外部表

列	描述
dfsegment	segment的内容ID (仅显示活动segment)
dfhostname	segment主机的主机名
dfdevice	设备名称
dfspace	segment文件系统中的可用磁盘空间 (以千字节为单位)

检查数据分布不均

Greenplum数据库中的所有表都是分布式的, 这意味着它们的数据被划分到系统中的所有segment中。如果数据分布不均匀, 则查询处理性能可能会受到影响。以下视图可以帮助诊断表是否具有不均匀的数据分布:

- [gp_skew_coefficients](#)
- [gp_skew_idle_fractions](#)

Parent topic: [gp_toolkit管理模式](#)

gp_skew_coefficients

该视图通过计算存储在每个segment上的数据的变异系数 (CV) 来显示数据分布偏斜。所有用户均可访问此视图, 但是非超级用户将只能看到他们有权访问的表

Table 41. gp_skew_coefficients视图

列	描述
skcoid	表的对象ID。
skcnamespace	定义表的命名空间。
skcrelname	表名。
skccoeff	变异系数 (CV) 计算为标准偏差除以平均值。它同时考虑了平均值和围绕数据序列平均值的变异性。值越低

越好。 较高的值表示较大的数据偏斜。

gp_skew_idle_fractions

该视图通过计算表扫描期间空闲系统的百分比来显示数据分布偏差，这是处理数据偏差的指标。所有用户均可访问此视图，但是非超级用户将只能看到他们有权访问的表

Table 42. gp_skew_idle_fractions视图

列	描述
sifoid	表的对象ID。
sifnamespace	定义表的名称空间。
sifrelname	表名。
siffraction	在表扫描期间空闲的系统百分比，这表示数据分布不均匀或查询处理偏斜。例如，值0.1表示10%的歪斜，值0.5表示50%的歪斜，依此类推。歪斜率超过10%的表应评估其分配策略。

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

gpperfmon 数据库是一个专用数据库，Greenplum segment 主机上的数据收集代理程序将查询和系统统计信息保存在这个数据库中。

gpperfmon 数据库通过 `gpperfmon_install` 命令行工具创建。这个工具然后创建 `gpmon` 数据库角色并启用 master 和 segment 主机上的数据收集代理程序。更多关于该工具使用和数据收集代理配置的信息，请参考 *Greenplum* 数据库工具指南 中的 `gpperfmon_install` 参考手册。

gpperfmon 数据库由三组表组成。它们分别用于在不同阶段捕捉查询和系统状态信息。

- `_now` 系列表存储当前系统指标，例如活动查询等。
- `_tail` 系列表用于在数据保存到 `_history` 系列表之前暂存数据。`_tail` 系列表仅供内部使用，不提供用户查询。
- `_history` 系列表用于存储历史数据。

`_now` 和 `_tail` 表的数据存储为文本文件，保存在 master 主机文件系统中，通过外部表在 gpperfmon 数据库中访问。`history` 系列表是 gpperfmon 数据库的常规内存表。只有运行超过一定时间(默认是 20 秒)的查询才会保存到历史数据表中。你可以通过设置 `$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf` 中的 `min_query_time` 参数来修改这个门限值。设置为 0 将会保存所有查询历史信息。

Note: gpperfmon 不支持 ALTER SQL 命令。ALTER 查询不会被记录在 gpperfmon 历史查询表中。

历史查询表按月进行分区。关于移除老分区的信息，请参见 [历史查询表分区保留](#)。

该数据库包含以下几类表：

- `database_*` 系列表存储一个 Greenplum 数据库实例的查询负载信息。
- `diskspace_*` 系列表存储磁盘空间指标。
- `log_alert_*` 系列表存储 `pg_log` 中的错误和警告消息。
- `queries_*` 系列表存储高级查询状态信息。
- `segment_*` 系列表存储 Greenplum 数据库 segment 实例的内存分配和统计信息。
- `socket_stats_*` 系列表存储一个 Greenplum 数据库实例中 socket

使用统计指标信息。 注意: 这些表为将来使用保留, 当前没有填充信息。

- `system_*` 系列表存储系统工具程序指标。

gpperfmon 数据库还包含下列视图:

- `dynamic_memory_info` 视图展示每个主机中所有 segments 的汇总, 以及每个主机中 动态内存使用量。
- `memory_info` 视图展示来自 `system_history` 和 `segment_history` 表的每个主机的内存信息.

历史查询表分区保留

gpperfmon 数据库中的 历史查询 表按月进行分区。分区 在需要时会以两个月为增量自动添加。

`$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf` 配置文件中的 `partition_age` 参数可以设置为要保留的每月最大分区数。添加新分区时, 将自动删除早于指定值的分区。

`partition_age` 的默认值是 0, 这意味着管理员必须手动删除 不需要的分区。

警报日志处理和日志轮换

当 `gp_gperfmon_enable` 服务期配置参数设置为 `true` 时, Greenplum 数据库 syslogger 会把警报消息写到一个 `.csv` 文件中。该文件位于 `$MASTER_DATA_DIRECTORY/gpperfmon/logs` 目录。

通过修改 `postgresql.conf` 文件中的 `gpperfmon_log_alert_level` 服务期配置参数, 可以将写入日志消息级别设置为 `none`, `warning`, `error`, `fatal`, 或者 `panic`。默认的消息基本是 `warning`。

日志存储的目录可以通过设置

`$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf` 配置文件中的 `log_location` 配置变量来改变。

syslogger 每 24 小时, 或在当前日志文件达到或超过 1MB 时轮换警报日志。

SQL

如果单条错误消息包含一个长语句或者长堆栈信息时，轮换日志文件可能超过 1MB。此外，syslogger 会分批处理错误消息，每个日志记录过程分别对应一个单独的分块。日志块的大小取决于操作系统；例如，在 Red Hat Enterprise Linux 系统中，它是 4096 字节。如果许多 Greenplum 数据库会话同时生成错误消息，则在检查文件大小并触发日志轮换之前，日志文件可能会显着增长。

gpperfmon 数据收集过程

当 Greenplum 数据库启用 gpperfmon 支持下启动时，它将创建一个 gpmmmon 代理进程。gpmmmon 然后在 Greenplum 数据库集群的 master 主机和每个 segment 主机上启动一个 gpsmon 代理进程。Greenplum 数据库的 postmaster 进程监视 gpmmmon 进程并在需要时重启该进程；gpmmmon 进程监视和在需要时重启 gpsmon 进程。

gpmmmon 进程以循环方式运行，并按一个可配置间隔检索 gpsmon 系列进程聚合的数据，并将其添加到 _now 和 _tail 外部数据库表的数据文件，然后存入 _history 常规内存数据库表中。

Note: gpperfmon 中的 log_alert 表流程不同于此，因为警报消息由 Greenplum 数据库系统 logger 发送，而不是通过 gpsmon 发送。参见 [警报日志处理和日志轮换](#) 了解更多信息。

`$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf` 配置文件中的两个配置参数控制着 gpmmmon 被触发的频度：

- `quantum` 参数是以秒为单位的频率，gpmmmon 依此频率向 segment 节点的 gpsmon 代理进程请求数据，并将获取到的数据添加到 `_now` 和 `_tail` 外部表数据文件中。`quantum` 参数的有效设定值是 10, 15, 20, 30, 以及 60. 默认值是 15.
- `harvest_interval` 参数是以秒为单位的频率，依此频率将 `_tail` 表中的数据移动到 `_history` 表中。`harvest_interval` 至少是 30. 默认值为 120.

参见 *Greenplum* 数据库工具指南 中的 `gpperfmon_install` 管理工具参考手册，查阅 gpperfmon 配置参数完整列表。

以下步骤描述了，当 gpperfmon 支持启用时，数据从 Greenplum 数据库进入 gpperfmon 的流程。

1. 在执行查询时，Greenplum 数据库查询调度程序和查询执行程序进程以 UDP 报文形式发送查询状态消息。
`gp_gpperfmon_send_interval` 服务器配置变量决定数据库发送这些消息的频率。默认为每秒发送一次。

2. 每个主机上的 gpmon 进程接收 UDP 报文, 合并并汇总所包含的数据, 并添加其他主机指标, 例如 CPU 和内存使用率。
3. gpmon 进程继续累积数据, 直到它们从 gpmon 接收到转储命令为止。
4. gpmon 进程通过发送其累积的状态数据和日志警报到一个 gpmon 事件侦听线程来响应 dump 命令。
5. gpmon 事件处理程序将指标保存到 master 主机的 \$MASTER_DATA_DIRECTORY/gpperfmon/data 目录下的 .txt 文件。

在每一个 quantum 间隔 (默认为 15 秒), gpmon 执行以下步骤:

1. 发送一个 dump 命令到 gpmon 进程。
2. 收集和转换 .txt 文件 (保存在 the \$MASTER_DATA_DIRECTORY/gpperfmon/data 目录中) 为 .dat 外部数据文件, 支持通过 gpperfmon 数据库中的 _now 和 _tail 外部表访问。
例如, 磁盘空间指标被添加到 diskspacenow.dat 和 _diskspace_tail.dat 带分隔符的文本文件中. 这些文本文件经由 gpperfmon 数据库中的 diskspacenow 和 _diskspace_tail 表访问.

在每一个 harvest_interval (默认为 120 秒), gpmon 对每一个 _tail 文件执行以下步骤:

1. 重命名 _tail 为一个 _stage 文件.
2. 创建一个新的 _tail 文件.
3. 把数据从 _stage 文件追加进 _tail 文件.
4. 允许一个 SQL 命令吧数据从 _tail 外部表插入进相应的 _history 表中.
例如, _database_tail 外部表的内容被插进 database_history 常规 (内存) 表中.
5. Deletes the _tail file 在内容被加载进数据库表中后, 删除 _tail 文件.
6. 收集所有 \$MASTER_DATA_DIRECTORY/gpperfmon/logs 目录下的 gpdb-alert-* .csv 文件 (除了 syslogger 已经打开并正在写入的最新文件) 到单个文件 alert_log_stage 中.

7. 把 alert_log_stage 文件内容加载进 gpperfmon 数据库的 log_alert_history 表中。
8. 清空 alert_log_stage 文件.

以下主题分别描述 gpperfmon 数据库每表中的内容。

- [**database_*** 表](#)

- [**diskspace_*** 表](#)

- [**interface_stats_*** 表](#)

- [**log_alert_*** 表](#)

- [**queries_*** 表](#)

- [**segment_*** 表](#)

- [**socket_stats_*** 表](#)

- [**system_*** 表](#)

- [**dynamic_memory_info** 视图](#)

- [**memory_info** 视图](#)

Parent topic: [Greenplum数据库参考指南](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ **Greenplum 数据库数据类型**

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

Greenplum 数据库有一套丰富的本地数据类型供用户使用。用户也可以使用 `CREATE TYPE` 命令定义新的数据类型。本参考手册对所有内置数据类型进行说明。除了这里列出的类型外，还有一些内部使用的数据类型，例如 `oid`（对象标识符），但这些不在本手册中介绍。

在 `contrib` 目录中的可选模块也可能安装新的数据类型。例如 `hstore` 模块引入了一种新数据类型和连带函数来与 key-value 键值对一起工作。参见 [hstore](#)。 `citext` 模块添加了一种大小写不敏感的文本数据类型。参见 [citext](#)。

下面的数据类型在SQL中使用: `bit`, `bit varying`, `boolean`, `character varying`, `varchar`, `character`, `char`, `date`, `double precision`, `integer`, `interval`, `numeric`, `decimal`, `real`, `smallint`, `time` (带时区或不带时区), 以及 `timestamp` (带时区或不带时区)。

每一种数据类型都有一种外部表示格式，这由数据类型的输入函数和输出函数决定。许多内置数据类型有明显的外部格式。然而，几种特别的数据类型要么只存在于 PostgreSQL (以及 Greenplum 数据库)，例如 `geometric paths`，要么有几种可能的表示格式，例如时间和日期类型。一些输入和输出函数是不可逆转的，也就是说，相比较原始输入，输出函数的结果可能存在精度损失。

表 1. Greenplum 数据库内置数据类型

名称	别名	大小	范围	说明
<code>bigint</code>	<code>int8</code>	8 字节	-9223372036854775808 到 9223372036854775807	大范围整数
<code>bigserial</code>	<code>serial8</code>	8 字节	1 到 9223372036854775807	大范围自增型整数
<code>bit [(n)]</code>		<i>n</i> 位	位串常量	定长位串
<code>bit varying [(n)]¹</code>	<code>varbit</code>	实际位数量	位串常量	变长位串
<code>boolean</code>	<code>bool</code>	1	true/false, t/f, yes/no, y/n, 1/0	逻辑布尔 (true/false)

Greenplum的PL/Perl语言

		字节		值
box		32 字节	((x1,y1),(x2,y2))	平面内长方形 - 不能在分布 键列使用中。
bytea ¹		1 字节 + 二进制 字符 串 大 小	8比特字符 序列	变长二进制字 符串
character [(n)] ¹	char [(n)]	1 字节 + n	最长 n 个字符的字符 串	定长, (输入 不足时自动 用) 空白填充
character varying [(n)] ¹	varchar [(n)]	1 字节 + 字符 串 大 小		

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

Greenplum数据库中的字符集支持允许您以各种字符集存储文本，包括单字节字符集（如ISO 8859系列）和多字节字符集（如EUC（扩展Unix代码），UTF-8），和Mule内部代码。客户端可以透明地使用所有支持的字符集，但少数只能在服务器上使用（即作为一种服务器端编码）。使用gpinitcluster初始化Greenplum数据库阵列时，会选择默认字符集。创建数据库时可以覆盖它，因此您可以拥有多个数据库，每个数据库具有不同的字符集。

Table 1. Greenplum数据库字符集 [1](#)

名称	描述	语言	是否服务端？	字节/字符	别名
BIG5	Big Five	繁体中文	No	1-2	WIN950, Windows950
EUC_CN	Extended UNIX Code-CN	简体中文	Yes	1-3	
EUC_JP	Extended UNIX Code-JP	日文	Yes	1-3	
EUC_KR	Extended UNIX Code-KR	韩文	Yes	1-3	
EUC_TW	Extended UNIX Code-TW	繁体中文, 台湾	Yes	1-3	
GB18030	国家标准	中文	No	1-2	
GBK	扩展国标	简体中文	No	1-2	WIN936, Windows936
ISO_8859_5	ISO 8859-5, ECMA 113	Latin/Cyrillic	Yes	1	
ISO_8859_6	ISO 8859-6, ECMA 114	Latin/Arabic	Yes	1	
ISO_8859_7	ISO 8859-7, ECMA 118	Latin/Greek	Yes	1	
ISO_8859_8	ISO 8859-8, ECMA 121	Latin/Hebrew	Yes	1	
JOHAB	JOHA	Korean (Hangul)	Yes	1-3	
KOI8	KOI8-R(U)	Cyrillic	Yes	1	KOI8R

LATIN1	ISO 8859-1, ECMA 94	Western European	Yes	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	Central European	Yes	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	South European	Yes	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	North European	Yes	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	Turkish	Yes	1	ISO88599
LATIN6	ISO 8859- 10, ECMA 144	Nordic	Yes	1	ISO885910
LATIN7	ISO 8859- 13	Baltic	Yes	1	ISO885913
LATIN8	ISO 8859- 14	Celtic	Yes	1	ISO885914
LATIN9	ISO 8859- 15	LATIN1 with Euro and accents	Yes	1	ISO885915
LATIN10	ISO 8859- 16, ASRO SR 14111	Romanian	Yes	1	ISO885916
MULE_INTERNAL	Mule internal code	Multilingual Emacs	Yes	1-4	
SJIS	Shift JIS	Japanese	No	1-2	MsKANJI, ShiftJIS, WIN932, Windows932
SQL_ASCII	unspecified 2 ...	any	No	1	
UHC	Unified Hangul Code	Korean	No	1-2	WIN949, Windows949
UTF8	Unicode, 8- bit	all	Yes	1-4	Unicode
WIN866	Windows CP866	Cyrillic	Yes	1	ALT
WIN874	Windows CP874	Thai	Yes	1	
WIN1250	Windows CP1250	Central European	Yes	1	

WIN1251	Windows CP1251	Cyrillic	Yes	1	WIN
WIN1252	Windows CP1252	Western European	Yes	1	
WIN1253	Windows CP1253	Greek	Yes	1	
WIN1254	Windows CP1254	Turkish	Yes	1	
WIN1255	Windows CP1255	Hebrew	Yes	1	
WIN1256	Windows CP1256	Arabic	Yes	1	
WIN1257	Windows CP1257	Baltic	Yes	1	
WIN1258	Windows CP1258	Vietnamese	Yes	1	ABC, TCVN, TCVN5712, VSCII

Parent topic: [Greenplum数据库参考指南](#)

设置字符集

gpinit system通过在初始化时读取`gp_init_config`文件中`ENCODING`参数的设置来定义Greenplum数据库系统的默认字符集。默认字符集是`UNICODE`或`UTF8`。

除了用作系统范围的默认值之外，您还可以创建具有不同字符集的数据库。例如：

```
=> CREATE DATABASE korean WITH ENCODING 'EUC_KR';
```

重点: 虽然您可以为数据库指定所需的任何编码，但选择与您选择的语言环境不同的编码是不明智的。`LC_COLLATE`和`LC_CTYPE`设置意味着特定的编码，并且依赖于区域设置的操作（例如排序）可能会错误解释处于不兼容编码的数据。

由于这些语言环境设置被`gpinit system`冻结，因此在不同数据库中使用不同编码的明显的灵活性更具理论性而非实用性。

安全使用多种编码的一种方法是在初始化期间将语言环境设置为`C`或`POSIX`，从而禁用任何真实的语言环境感知。

服务器和客户端之间的字符集转换

Greenplum数据库支持服务器和客户端之间针对特定字符集组合的自动字符集转
master `pg_conversion` catalog Greenplum

换。转换信息存储在系统表中。数据库附带了一些预定义的转换，或者您可以使用SQL命令CREATE CONVERSION创建新的转换。

Table 2. 客户端/服务器字符集转换

服务器字符集	可用的客户端字符集
BIG5	不支持作为服务器编码
EUC_CN	EUC_CN, MULE_INTERNAL, UTF8
EUC_JP	EUC_JP, MULE_INTERNAL, SJIS, UTF8
EUC_KR	EUC_KR, MULE_INTERNAL, UTF8
EUC_TW	EUC_TW, BIG5, MULE_INTERNAL, UTF8
GB18030	不支持作为服务器编码
GBK	不支持作为服务器编码
ISO_8859_5	ISO_8859_5, KOI8, MULE_INTERNAL, UTF8, WIN866, WIN1251
ISO_8859_6	ISO_8859_6, UTF8
ISO_8859_7	ISO_8859_7, UTF8
ISO_8859_8	ISO_8859_8, UTF8
JOHAB	JOHAB, UTF8
KOI8	KOI8, ISO_8859_5, MULE_INTERNAL, UTF8, WIN866, WIN1251
LATIN1	LATIN1, MULE_INTERNAL, UTF8
LATIN2	LATIN2, MULE_INTERNAL, UTF8, WIN1250
LATIN3	LATIN3, MULE_INTERNAL, UTF8
LATIN4	LATIN4, MULE_INTERNAL, UTF8
LATIN5	LATIN5, UTF8
LATIN6	LATIN6, UTF8
LATIN7	LATIN7, UTF8
LATIN8	LATIN8, UTF8
LATIN9	LATIN9, UTF8
LATIN10	LATIN10, UTF8
MULE_INTERNAL	MULE_INTERNAL, BIG5, EUC_CN, EUC_JP, EUC_KR, EUC_TW, ISO_8859_5, KOI8, LATIN1 to LATIN4, SJIS, WIN866, WIN1250, WIN1251
SJIS	不支持作为服务器编码
SQL_ASCII	不支持作为服务器编码
UHC	不支持作为服务器编码
UTF8	所有支持的编码

WIN866	WIN866
ISO_8859_5	KOI8, MULE_INTERNAL, UTF8, WIN1251
WIN874	WIN874, UTF8
WIN1250	WIN1250, LATIN2, MULE_INTERNAL, UTF8
WIN1251	WIN1251, ISO_8859_5, KOI8, MULE_INTERNAL, UTF8, WIN866
WIN1252	WIN1252, UTF8
WIN1253	WIN1253, UTF8
WIN1254	WIN1254, UTF8
WIN1255	WIN1255, UTF8
WIN1256	WIN1256, UTF8
WIN1257	WIN1257, UTF8
WIN1258	WIN1258, UTF8

要启用自动字符集转换，您必须告诉Greenplum数据库您要在客户端中使用的字符集（编码）。有几种方法可以实现此目的：

- 在psql中使用\encoding命令，它允许您动态更改客户端编码。
- 使用 `SET client_encoding TO.`

要设置客户端编码，请使用以下SQL命令

```
=> SET CLIENT_ENCODING TO 'value';
```

要查询当前的客户端编码：

```
=> SHOW client_encoding;
```

要返回到默认编码：

```
=> RESET client_encoding;
```

- 使用PGCLIENTENCODING环境变量。在客户端环境中定义PGCLIENTENCODING时，将在与服务器建立连接时自动选择该客户端编码。（这可以随后使用上面提到的任何其他方法覆盖。）
- 设置配置参数client_encoding。如果在master postgresql.conf文件中设置了client_encoding，则在建立与Greenplum数据库的连接时会自动选择该客户端编码。（这可以随后使用上面提到的任何其他方法覆盖。）

如果无法转换特定字符“假设您为服务器选择了EUC_JP而对客户端选择了LATIN1，那么某些日文字符在LATIN1中没有表示”，则会报告错误。

如果客户端字符集定义为SQL_ASCII，则无论服务器的字符集如何，都将禁用编码转换。除非使用all-ASCII数据，否则使用SQL_ASCII是不明智的。

- ¹ 并非所有API都支持所有列出的字符集。例如， JDBC驱动程序不支持MULE_INTERNAL, LATIN6, LATIN8和LATIN10。
- ² SQL_ASCII设置与其他设置的行为大不相同。字节值0-127根据ASCII标准进行解释，字节值128-255作为未解释的字符。如果使用任何非ASCII数据，则将SQL_ASCII设置用作客户端编码是不明智的。不支持SQL_ASCII作为服务器编码。

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

有很多Greenplum服务器配置参数能够影响 Greenplum数据库系统的行为。如同常规的PostgreSQL数据库系统一样，许多配置参数有相同的名称、设置以及行为。

- [参数类型和值](#)描述参数数据类型和值。
- [设置参数](#)描述了谁可以改变它们以及何时何处可以设置它们的限制。
- [参数类别](#)根据功能组织参数。
- [配置参数](#)以字母顺序列出参数描述。

参数类型和值

所有的参数不区分大小写。每个参数取值为四种类型之一：布尔，整数，浮点，或 字符串。

布尔值可能被写成ON, OFF, TRUE, FALSE, YES, NO, 1, 0(所有都是不区分大小写)。

枚举类型参数的指定方式与字符串参数相同，但仅限于一组有限的值。枚举参数值不区分大小写。

某些设置指定内存或者时间值。每个都有一个隐式单位，它是千字节，块（通常为8千字节），毫秒，秒，或者分钟。有效内存单位是kB（千字节），MB（兆字节），和GB（千兆字节）。有效的时间单位是ms（毫秒），s（秒），min（分钟），h（小时），和d（天）。注意内存单元的乘数为1024，而不是1000。有效的时间表达式包含数字和单位。当指定内存和时间单位时使用SET命令，将值括在引号中。例如：

```
SET statement_mem TO '200MB';
```

Note: 值和单位名字之间没有空格。

设置参数

许多配置参数对谁可以更改它们，何处改变它们以及何时改变他们都做了限制，例如，用户必须是Greenplum 数据库的超级用户。其他参



Greenplum的PL/Perl语言

数需要重启系统才能更改生效。分类为*session*级别的参数可以在系统层级设置（在`postgresql.conf`文件中），在数据库层级设置（使用`ALTER DATABASE`），在角色层级设置（使用`ALTER ROLE`），在数据库角色级设置（`ALTER ROLE...IN DATABASE...SET`），或者在会话层级设置（使用`using SET`）。系统参数只能在`postgresql.conf`文件中设置。

在Greenplum数据库中，Master实例和Segment实例都有它自己的`postgresql.conf`文件（位于各自的数据目录中）。一些参数被认为是*local*参数，意味着每个Segment实例都查看自己的`postgresql.conf`文件来获取该参数的值。用户必须在系统上的每个实例中都设置local参数（Master实例和Segment实例）。其他参数被认为是*master*参数。*master*参数只能在Master实例上设置。

这个表描述了服务器配置参数描述中“可设置分类”列中的值。

Table 1. Settable Classifications

设置分类	描述
master或local	<p><i>master</i>参数只能在Greenplum的Master实例的<code>postgresql.conf</code>文件中设置。然后，该参数的值在运行时被传递到（或者忽略）到Segment实例。</p> <p><i>local</i>参数必须在Master实例和每个Segment实例的<code>postgresql.conf</code>文件中设置。每个Segment实例会在其自己的配置文件中去获得参数。<i>local</i>参数设置总是需要系统重启才能生效。</p>
session或system	<p><i>Session</i>参数可以在数据库会话中即时更改，并且可以具有以下设置的层次结构：在系统层级（<code>postgresql.conf</code>）、在数据库层级（<code>ALTER DATABASE...SET</code>）、在角色层级（<code>ALTER ROLE...SET</code>）、在数据库和角色层级或者会话层级（<code>SET</code>）。如果参数设置为多个级别，则最细度的设置优先（例如，会话覆盖数据库和角色，数据库和角色覆盖角色，角色覆盖数据库，数据库覆盖系统）。</p> <p><i>system</i>参数只能通过<code>postgresql.conf</code>文件修改。</p>
restart或reload	当更改 <code>postgresql.conf</code> 文件中的参数值时，有些需要重启Greenplum数据库才能生效。其他参数只需

	要重新加载服务器配置文件就能刷新（使用gpstop -u），不需要停止系统。
superuser	这些会话参数只能通过数据库超级用户才能设置，常规用户不能设置这个参数。
read only	这些参数不能被数据库用户和超级用户设置，当前的参数值可以显示但是不会更改。

Greenplum数据库® 6.0文档

- 参考指南
- SQL Command Reference
- SQL 2008可选特性兼容性
- Greenplum环境变量
- 保留标识符和SQL关键字
- 系统目录参考
- gp_toolkit管理模式
- gpperfmon 数据库
- Greenplum 数据库数据类型
- 字符集支持
- 服务器配置参数

内置函数摘要

- Greenplum MapReduce规范
- Greenplum PL/pgSQL过程语言
- Greenplum PL/R 语言扩展

Greenplum数据库支持内置函数和运算符，包括可用于窗口表达式的分析函数和窗口函数。有关使用内置Greenplum数据库函数的信息，请参阅Greenplum数据库管理员指南中的“使用函数和运算符”。

- [Greenplum数据库函数类型](#)
- [内置函数和运算符](#)
- [JSON函数和操作符](#)
- [窗口函数](#)
- [高级聚合函数](#)
- [文本搜索函数和操作符](#)
- [范围函数和运算符](#)

Parent topic: [Greenplum数据库参考指南](#)

Greenplum数据库函数类型

Greenplum数据库评估SQL表达式中使用的函数和运算符。某些函数和运算符只允许在master服务器上执行，因为它们可能导致Greenplum数据库segment实例中的不一致。该表描述了Greenplum数据库函数类型。

Table 1. Greenplum数据库中的函数

函数类型	Greenplum是否支持	描述	注释
IMMUTABLE	是	仅依赖于其参数列表中的信息。给定相同的参数值，始终返回相同的结果。	
STABLE	在大多数情况下是的	在单个表扫描中，对相同的参数值返回相同的结果，但结果将通过SQL语句进行更改。	结果取决于数据库查找或参数值。 <code>current_timestamp</code> 系列函数是STABLE；值在执行中不会改变。
VOLATILE	受限制的	函数值可以在单个表扫描中更改。例如： <code>random()</code> , <code>timeofday()</code> 。	任何具有副作用的函数都是易变的，即使其结果是可预测的。例如： <code>setval()</code> 。

在Greenplum数据库中，数据跨segment分割 - 每个segment是不同的PostgreSQL数据库。为了防止出现不一致或意外的结果，如果它们包含SQL命令或以任何方式修改数据库，则不要在segment级别执行归类为VOLATILE的函数。例如，不允许在Greenplum数据库中对分布式数据执行者如`setval()`之类的函数，因为它们可能导致segment实例之间的数据不一致。

为确保数据一致性，可以在master服务器上评估和运行的语句中安全地使用VOLATILE和STABLE函数。例如，以下语句在master上运行（没有FROM子句的语句）：

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

如果语句具有包含分布式表的FROM子句，并且FROM子句中的函数返回一组行，则该语句可以在segment上运行：

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

Greenplum数据库不支持返回表引用（rangeFuncs）的函数或使用refCursor数据类型的函数。

内置函数和运算符

下表列出了PostgreSQL支持的内置函数和运算符的类别。Greenplum数据库支持所有函数和运算符，如PostgreSQL，但STABLE和VOLATILE函数除外，它们受[Greenplum数据库函数类型](#)中的限制。有关这些内置函数和运算符的更多信息，请参阅PostgreSQL文档的[函数和运算符](#)部分。

Table 2. 内置函数和运算符

操作符/函数类别	VOLATILE函数	STABLE函数	限制
逻辑操作符			
比较操作符			
数学函数和操作符	random setseed		
字符串函数和操作符	所有内建转换函数	convert pg_client_encoding	
二进制字符串和函数操作符			
位串函数和操作符			
模式匹配			
数据类型格式函数		to_char to_timestamp	
日期/时间函数和操作符	timeofday	age current_date current_time current_timestamp localtime localtimestamp now	
枚举支持函数			
几何函数和操作符			
网络地址函数和操作符			
序列操作函数	nextval() setval()		
条件表达式			
数组函数和操作符		所有数组函数	
聚合函数			
子查询表达式			
行和数组比较			
返回集合函数	generate_series		
系统信息函数		所有会话信息函数 所有访问特权查询函数 所有模式可见性查询函数	

		所有系统目录信息函数 所有注释信息函数 所有事务ID和快照	
系统管理函数 口	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	注意：函数pg_column_size显示存储值所需的字节，可能使用TOAST压缩。
XML函数 口 和类似函数的表达式		cursor_to_xml(cursor refcursor, count int, nulls boolean, tableforest boolean, targetns text) cursor_to_xmlschema(cursor refcursor, nulls boolean, tableforest boolean, targetns text) database_to_xml(nulls boolean, tableforest boolean, targetns text) database_to_xmlschema(nulls boolean, tableforest boolean, targetns text) database_to_xml_and_xmlschema(nulls boolean, tableforest boolean, targetns text) query_to_xml(query text, nulls boolean, tableforest boolean, targetns text) query_to_xmlschema(query text, nulls boolean, tableforest boolean, targetns text) query_to_xml_and_xmlschema(query text, nulls boolean, tableforest boolean, targetns text) schema_to_xml(schema name, nulls boolean, tableforest boolean, targetns text) schema_to_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text) schema_to_xml_and_xmlschema(schema name, nulls boolean, tableforest boolean,	

		targetns text) table_to_xml(tbl regclass, nulls boolean, tableforest boolean, targetns text) table_to_xmleschema(tbl regclass, nulls boolean, tableforest boolean, targetns text) table_to_xml_and_xmleschema(tbl regclass, nulls boolean, tableforest boolean, targetns text) xmlagg(xml) xmlconcat(xml[, ...]) xmlelement(name name [, xmlattributes(value [AS attname] [, ...]) [, content, ...]) xmlexists(text, xml) xmlforest(content [AS name] [, ...]) xml_is_well_formed(text) xml_is_well_formed_document(text) xml_is_well_formed_content(text) xmpparse ({ DOCUMENT CONTENT } value) xpath(text, xml) xpath(text, xml, text[]) xpath_exists(text, xml) xpath_exists(text, xml, text[]) xmlpi(name target [, content]) xmlroot(xml, version text no value [, standalone yes no no value]) xmlserialize ({ DOCUMENT CONTENT } value AS type) xml(text) text(xml) xmlcomment(xml) xmlconcat2(xml, xml)	
--	--	--	--

JSON函数和操作符

- 数据库包含用于创建和操作 数据的内置函数和运算符。
- [JSON操作符](#)
 - [JSON创建函数](#)
 - [JSON聚合函数](#)
 - [JSON处理函数](#)

Note: 对于json数据类型值，即使JSON对象包含重复键，也会保留所有键/值对。对于重复键，JSON处理函数将最后一个值视为可操作的值。对于jsonb数据类型，不保留重复的对象键。如果输入包含重复键，则仅保留最后一个值。参考 [关于JSON数据](#)。

JSON操作符

此表描述了可用于json和jsonb数据类型的运算符。

Table 3. json和jsonb操作符

操作符	右操作数类型	描述	示例	示例结果
->	int	获取JSON数组元素（从零索引）。	'[{"a": "foo"}, {"b": "bar"}, {"c": "baz"}] :: json->2	{"c": "baz"}
->	text	通过键获取JSON对象字段。	'{"a": {"b": "foo"}}' :: json->'a'	{"b": "foo"}
->>	int	获取JSON数组元素作为text。	'[1, 2, 3]' :: json->>2	3
->>	text	获取JSON对象字段作为text。	'{"a": 1, "b": 2}' :: json->>'b'	2
#>	text[]	在指定的路径获取JSON对象。	'{"a": {"b": {"c": "foo"}}}' :: json#>'{a,b}'	{"c": "foo"}
#>>	text[]	以指定路径获取JSON对象作为text。	'{"a": [1, 2, 3], "b": [4, 5, 6]}' :: json#>>'{a,2}'	3

Note: 对于json和jsonb数据类型，这些运算符都有并行变体。字段、元素和路径提取运算符返回与其左侧输入（json或jsonb）相同的数据类型，但指定为返回text的那些除外，它将值强制转换为text。如果JSON输入没有与请求匹配的正确结构，则字段、元素和路径提取操作符将返回NULL，而不是失败；例如，如果不存在这样的元素。

需要jsonb数据类型作为左操作数的运算符如下表所述。其中许多运算符可以由jsonb运算符类索引。有关jsonb包含和存在语义的完整描述，请参阅[jsonb容器与存在](#)。有关如何使用这些运算符有效地索引jsonb数据的信息，请参阅[jsonb索引](#)。

Table 4. jsonb操作符

操作符	右操作数类型	描述	示例
@>	jsonb	左JSON值是否包含正确的值？	'{"a": 1, "b": 2}' :: jsonb @> '{"b": 2}' :: jsonb
<@	jsonb	左JSON值是否包含在右值内？	'{"b": 2}' :: jsonb <@ '{"a": 1, "b": 2}' :: jsonb
?	text	键/元素字符串是否存在JSON值中？	'{"a": 1, "b": 2}' :: jsonb ? 'b'
?	text[]	是否存在任何这些键/元素字符串？	'{"a": 1, "b": 2, "c": 3}' :: jsonb ? array['b', 'c']
?&	text[]	是否存在所有这些键/元素字符串？	'["a", "b"]' :: jsonb ?& array['a', 'b']

下表中的标准比较运算符仅适用于jsonb数据类型，不适用于json数据类型。它们遵循[jsonb索引](#)中描述的B树操作的排序规则。

Table 5. jsonb比较操作符

操作符	描述
<	小于
>	大于
<=	小于等于
>=	大于等于
=	等于
<>或 !=	不等于

Note: !=运算符在解析器阶段转换为<>。不可能实现执行不同操作的!=和<>运算符。

JSON创建函数

此表描述了创建json数据类型值的函数。（目前，jsonb没有等效函数，但您可以将其中一个函数的结果转换为jsonb。）

Table 6. JSON创建函数

函数	描述	示例	示例结果
<code>to_json(anyelement)</code>	以JSON对象的形式返回值。数组和复合体以递归方式处理，并转换为数组和对象。如果输入包含从类型到json的强制转换，则使用强制转换函数执行转换；否则，将生成JSON标量值。对于除数字，布尔值或空值之外的任何标量类型，将使用正确引用和转义文本表示，以使其成为有效的JSON字符串。	<code>to_json('Fred said "Hi."':text)</code>	"Fred said \"Hi.\""
<code>array_to_json(anyarray [, pretty_bool])</code>	将数组作为JSON数组返回。多维数组成为数组的JSON数组。 如果pretty_bool为true，则会在dimension-1元素之间添加换行符。	<code>array_to_json('{{1,5}, {99,100}}':int[])</code>	[[1,5], [99,100]]
<code>row_to_json(record [, pretty_bool])</code>	将行作为JSON对象返回。 如果pretty_bool为true，则会在level-1元素之间添加换行符。	<code>row_to_json(row(1,'foo'))</code>	{"f1":1, "f2":"foo"}
<code>json_build_array(VARIADIC "any")</code>	从VARIADIC参数列表构建可能异构类型的JSON数组。	<code>json_build_array(1,2,'3',4,5)</code>	[1, 2, "3", 4, 5]
<code>json_build_object(VARIADIC "any")</code>	从VARIADIC参数列表中构建JSON对象。参数列表按顺序获取并转换为一组键/值对。	<code>json_build_object('foo',1,'bar',2)</code>	{"foo": 1, "bar": 2}
<code>json_object(text[])</code>	从文本数组中构建JSON对	<code>json_object('{a, 1, b, "def", c,')</code>	{"a": "1", "b":

	<p>象。该数组必须是一维或二维数组。</p> <p>一维数组必须具有偶数个元素。元素被视为键/值对。</p> <p>对于二维数组，每个内部数组必须具有恰好两个元素，这些元素被视为键/值对。</p>	<pre>3.5}')) json_object('{{a, 1},{b, "def"}, {c, 3.5}}')</pre>	<pre>"def", "c": "3.5"} "def", "c": "3.5"} json_object('{{a, b}', '{1,2}') { "a": "1", "b": "2"}</pre>
<code>json_object(keys text[], values text[])</code>	从文本数组中构建JSON对象。这种形式的 <code>json_object</code> 从两个独立的数组中生成对获取键和值。在所有其他方面，它与单参数形式相同。	<code>json_object('{{a, b}', '{1,2}')</code>	<code>{ "a": "1", "b": "2"}</code>

Note: `array_to_json`和`row_to_json`与`to_json`具有相同的行为，除了提供漂亮的打印选项。为`to_json`描述的行为同样适用于由其他JSON创建函数转换的每个单独的值。

Note: `hstore` 扩展具有从`hstore`到`json`的强制转换，因此通过JSON创建函数转换的`hstore`值将表示为JSON对象，而不是原始字符串值。

JSON聚合函数

此表显示函数将记录聚合到JSON对象数组和值对聚合到JSON对象

Table 7. JSON聚合函数

函数	参数类型	返回类型	描述
<code>json_agg(record)</code>	<code>record</code>	<code>json</code>	将记录聚合为JSON对象数组。
<code>json_object_agg(name, value)</code>	<code>("any" , "any")</code>	<code>json</code>	将名称/值对聚合为JSON对象。

JSON处理函数

此表显示可用于处理`json`和`jsonb`值的函数。

许多这些处理函数和运算符将JSON字符串中的Unicode转义转换为适当的单个字符。如果输入数据类型是`jsonb`，则这不是问题，因为转换已经完成。但是，对于`json`数据类型输入，这可能会导致抛出错误。请参阅 [关于JSON数据](#)。

Table 8. JSON处理函数

函数	返回类型	描述	示例	示例结果
<code>json_array_length(json)</code> <code>jsonb_array_length(jsonb)</code>	<code>int</code>	返回最外层JSON数组中的元素数。	<code>json_array_length('[1,2,3,{"f1":1,"f2": [5,6]},4]')</code>	5
<code>json_each(json)</code> <code>jsonb_each(jsonb)</code>	<code>setof key text, value json</code> <code>setof key text, value jsonb</code>	将最外层的JSON对象扩展为一组键/值对。	<code>select * from json_each('{"a": "foo", "b": "bar"}')</code>	<pre>key value ----+---- a "foo" b "bar"</pre>

json_each_text(json) jsonb_each_text(jsonb)	setof key text, value text	将最外层的JSON对象扩展为一组键/值对。返回的值将是text类型。	select * from json_each_text('{"a":"foo", "b":"bar"}')	key value ---+--- a foo b bar
json_extract_path(from_json json, VARIADIC path_elems text[]) jsonb_extract_path(from_json jsonb, VARIADIC path_elems text[])	json jsonb	返回path_elems指向的JSON值（相当于#>运算符）。	json_extract_path('{"f2":{"f3":1}, "f4": {"f5":99, "f6":"foo"} }', 'f4')	{"f5":99, "f6":"foo"}
json_extract_path_text(from_json json, VARIADIC path_elems text[]) jsonb_extract_path_text(from_json jsonb, VARIADIC path_elems text[])	text	返回path_elems指向的JSON值作为文本。相当于#>>运算符。	json_extract_path_text('{"f2":{"f3":1}, "f4": {"f5":99, "f6":"foo"} }', 'f4', 'f6')	foo
json_object_keys(json) jsonb_object_keys(jsonb)	setof text	返回最外层JSON对象中的键集。	json_object_keys('{"f1":"abc", "f2": {"f3":"a", "f4":"b"} }')	json_object_keys ----- f1 f2
json_populate_record(base anyelement, from_json json) jsonb_populate_record(base anyelement, from_json jsonb)	anyelement	将from_json中的对象扩展为其列与base定义的记录类型匹配的行。见注1 注1 。	select * from json_populate_record(null::myrowtype, '{"a":1, "b":2}')	a b ---+--- 1 2
json_populate_recordset(base anyelement, from_json json) jsonb_populate_recordset(base anyelement, from_json jsonb)	setof anyelement	将from_json中最外层对象的数组扩展为一组行，这些行的列与base定义的记录类型匹配。见注1。	select * from json_populate_recordset(null::myrowtype, '[{"a":1, "b":2}, {"a":3, "b":4}]')	a b ---+--- 1 2 3 4
json_array_elements(json) jsonb_array_elements(jsonb)	setof json setof jsonb	将JSON数组扩展为一组JSON值。	select * from json_array_elements('[1,true, [2,false]]')	value ----- 1 true [2, false]
json_array_elements_text(json) jsonb_array_elements_text(jsonb)	setof text	将JSON数组扩展为一组文本值。	select * from json_array_elements_text('["foo", "bar"]')	value ----- foo bar
json_typeof(json) jsonb_typeof(jsonb)	text	以文本字符串形式返回最外层JSON值的类型。可能的类型是object, array, string, number, boolean和null。见注2	json_typeof('-123.4')	number
json_to_record(json) jsonb_to_record(jsonb)	record	从JSON对象构建任意记录。见注1。 与返回记录的所有函数一样，调用者必须使用AS子句显式定义记录的结构。	select * from json_to_record('{"a":1, "b": [1,2,3], "c":"bar"}') as x(a int, b text, d text)	a b d ---+---+--- 1 [1,2,3]

<pre>json_to_recordset(json) jsonb_to_recordset(jsonb)</pre>	<p>setof record</p>	<p>从JSON对象数组构建任意记录集请参见注1。</p> <p>与返回记录的所有函数一样，调用者必须使用AS子句显式定义记录的结构。</p>	<pre>select * from json_to_recordset('[{"a":1,"b":"foo"}, {"a":2,"c":"bar"}]') as x(a int, b text);</pre>	<pre>a b ---+--- 1 foo 2 </pre>
--	---------------------	---	---	--------------------------------------

Note:

1. 函数`json_populate_record()`, `json_populate_recordset()`, `json_to_record()`和`json_to_recordset()`的示例使用常量。但是，典型的用法是引用`FROM`子句中的表，并使用其中一个`json`或`jsonb`列作为函数的参数。然后可以在查询的其他部分中引用所提取的键值。例如，可以在`WHERE`子句和目标列表中引用该值。以这种方式提取多个值可以提高性能，而不是使用每个键操作符单独提取它们。JSON键与目标行类型中的相同列名匹配。这些函数的JSON类型强制转换可能不会产生某些类型的期望值。将不会在输出中省略未出现在目标行类型中的JSON字段，并且与任何JSON字段不匹配的目标列将为`NULL`。
2. 不应将`json_typeof`函数`null`返回值`null`与SQL `NULL`混淆。调用`json_typeof('null'::json)`将返回`null`，调用`json_typeof(NULL::json)`将返回`SQLNULL`。

窗口函数

以下是Greenplum数据库内置窗口函数。所有窗口函数都是不可变的。有关窗口函数的更多信息，请参阅Greenplum数据库管理员指南中的“窗口表达式”。

Table 9. 窗口函数

函数	返回类型	完整语法	描述
<code>cume_dist()</code>	<code>double precision</code>	<code>CUME_DIST() OVER ([PARTITION BY expr] ORDER BY expr)</code>	计算一组值中的值的累积分布。具有相等值的行始终评估为相同的累积分布值。
<code>dense_rank()</code>	<code>bigint</code>	<code>DENSE_RANK () OVER ([PARTITION BY expr] ORDER BY expr)</code>	计算有序行组中行的等级而不跳过等级值。具有相等值的行被赋予相同的等级值。
<code>first_value(expr)</code>	与输入 <code>expr</code> 类型相同	<code>FIRST_VALUE(expr) OVER ([PARTITION BY expr] ORDER BY expr [ROWS RANGE frame_expr])</code>	返回有序值集中的第一个值。
<code>lag(expr [,offset] [,default])</code>	与输入 <code>expr</code> 类型相同	<code>LAG(expr [, offset] [, default]) OVER ([PARTITION BY expr] ORDER BY expr)</code>	提供对同一个表的多个行的访问，而无需进行自连接。给定从查询返回的一系列行和光标的位置，LAG提供对该位置之前给定物理偏移处的行的访问。默认 <code>offset</code> 为1。 <code>default</code> 设置当偏移量超出窗口范围时返回的值。如果未指定 <code>default</code> ，则默认值为 <code>null</code> 。
<code>last_value(expr)</code>	与输入 <code>expr</code> 类型相同	<code>LAST_VALUE(expr) OVER ([PARTITION BY expr] ORDER BY expr [ROWS RANGE frame_expr])</code>	返回有序值集中的最后一个值。
<code>lead(expr [,offset] [,default])</code>	与输入 <code>expr</code> 类型相同	<code>LEAD(expr [,offset] [,exprdefault]) OVER ([PARTITION BY expr] ORDER BY expr)</code>	提供对同一个表的多个行的访问，而无需进行自连接。给定从查询返回的一系列行和光标的位置，lead提供对该位置之后给定物理偏移处的行的访问。如果未指定 <code>offset</code> ，则默认偏移量为1。 <code>default</code> 设置当偏移量超出窗口范围时返回的值。如果未指定 <code>default</code> ，则默认值为 <code>null</code> 。
<code>ntile(expr)</code>	<code>bigint</code>	<code>NTILE(expr) OVER ([PARTITION BY expr])</code>	将有序数据集划分为多个存储桶

		ORDER BY <i>expr</i>)	(由 <i>expr</i> 定义) , 并为每行分配存储桶编号。
percent_rank()	double precision	PERCENT_RANK () OVER ([PARTITION BY <i>expr</i>] ORDER BY <i>expr</i>)	计算假定行R减1的等级, 除以小于被评估的行数(在窗口分区内的)的1。
rank()	bigint	RANK () OVER ([PARTITION BY <i>expr</i>] ORDER BY <i>expr</i>)	计算有序值组中行的等级。排名标准具有相同值的行具有相同的排名。绑定行的数量被添加到等级编号以计算下一个等级值。在这种情况下, 排名可能不是连续的数字。
row_number()	bigint	ROW_NUMBER () OVER ([PARTITION BY <i>expr</i>] ORDER BY <i>expr</i>)	为应用它的每一行分配一个唯一的编号(窗口分区中的每一行或查询的每一行)。

高级聚合函数

以下内置高级分析函数是PostgreSQL数据库的Greenplum扩展。分析函数是不可变的。

Note: Greenplum MADlib分析扩展提供了额外的高级功能, 可以使用Greenplum数据库数据执行统计分析和机器学习。请参阅[Greenplum MADlib分析扩展](#)。

Table 10. 高级聚合函数

函数	返回类型	完整语法	描述
MEDIAN (<i>expr</i>)	timestamp, timestamptz, interval, float	MEDIAN (<i>expression</i>) <i>Example:</i> <pre>SELECT department_id, MEDIAN(salary) FROM employees GROUP BY department_id;</pre>	可以将二维数组作为输入。将这些数组视为矩阵。
PERCENTILE_CONT (<i>expr</i>) WITHIN GROUP (ORDER BY <i>expr</i> [DESC/ASC])	timestamp, timestamptz, interval, float	PERCENTILE_CONT(<i>percentage</i>) WITHIN GROUP (ORDER BY <i>expression</i>) <i>Example:</i> <pre>SELECT department_id, PERCENTILE_CONT (0.5) WITHIN GROUP (ORDER BY salary DESC) "Median_cont"; FROM employees GROUP BY department_id;</pre>	执行假设连续分布模型的逆分布函数。它采用百分位值和排序规范, 并返回与参数的numeric数据类型相同的数据类型。该返回值是执行线性插值后的计算结果。在此计算中忽略Null。
PERCENTILE_DISC (<i>expr</i>) WITHIN GROUP (ORDER BY <i>expr</i> [DESC/ASC])	timestamp, timestamptz, interval, float	PERCENTILE_DISC(<i>percentage</i>) WITHIN GROUP (ORDER BY <i>expression</i>) <i>Example:</i> <pre>SELECT department_id, PERCENTILE_DISC (0.5) WITHIN GROUP (ORDER BY salary DESC) "Median_desc"; FROM employees GROUP BY department_id;</pre>	执行假设离散分布模型的逆分布函数。它需要百分位值和排序规范。此返回值是集合中的元素。在此计算中忽略Null。
sum(array[])	smallint[] int[], bigint[], float[]	sum(array[[1,2],[3,4]]) <i>Example:</i>	执行矩阵求和。可以将被视为矩阵的二维数组作为输入。

		<pre>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}</pre>	
pivot_sum (label[], label, expr)	int[], bigint[], float[]	pivot_sum(array['A1','A2'], attr, value)	使用sum的数据透视聚合来解决重复的条目。
unnest (array[])	set of anyelement	unnest(array['one', 'row', 'per', 'item'])	将一维数组转换为行。返回一组anyelement, 多态的 PostgreSQL中的假型 。

文本搜索函数和操作符

下表总结了为全文搜索提供的函数和运算符。有关Greenplum数据库文本搜索工具的详细说明, 请参阅[使用全文搜索](#)。

Table 11. 文本搜索运算符

操作符	描述	示例	结果
<code>@@</code>	tsvector匹 配tsquery ?	<code>to_tsvector('fat cats ate rats') @@ to_tsquery('cat & rat')</code>	t
<code>@@@</code>	<code>@@</code> 的弃用同义词	<code>to_tsvector('fat cats ate rats') @@@ to_tsquery('cat & rat')</code>	t
<code> </code>	连接 tsvectors	<code>'a:1 b:2':tsvector 'c:1 d:2 b:3':tsvector</code>	'a':1 'b':2,5 'c':3 'd':4
<code>&&</code>	AND tsquerys一起	<code>'fat rat':tsquery && 'cat':tsquery</code>	('fat' 'rat') & 'cat'
<code> </code>	OR tsquerys一起	<code>'fat rat':tsquery 'cat':tsquery</code>	('fat' 'rat') 'cat'
<code>!!</code>	否定一个 tsquery	<code>!! 'cat':tsquery</code>	!'cat'
<code>@></code>	tsquery是否包含另一 个?	<code>'cat':tsquery @> 'cat & rat':tsquery</code>	f
<code><@</code>	tsquery是否包含在?	<code>'cat':tsquery <@ 'cat & rat':tsquery</code>	t

Note: tsquery包含运算符仅考虑两个查询中列出的词位, 忽略组合运算符。

除了表中显示的运算符之外, 还为类型tsvector和tsquery定义了普通的B树比较运算符 (=, <, 等)。这些对于文本搜索不是很有用, 但允许在这些类型的列上构建唯一索引。

Table 12. 文本搜索函数

函数	返回类型	描述	示例	结果
<code>get_current_ts_config()</code>	regconfig	获取默认文本搜索配置	<code>get_current_ts_config()</code>	english
<code>length(tsvector)</code>	integer	tsvector中的词位数	<code>length(fat:2,4 cat:3 rat:5A':tsvector)</code>	3
<code>numnode(tsquery)</code>	integer	词位数加tsquery中的运算符的数 量	<code>numnode('fat & rat' cat':tsquery)</code>	5
<code>plainto_tsquery([config</code>	tsquery	产生忽略标点符号的tsquery	<code>plainto_tsquery('english', 'The Fat Rats')</code>	'fat' & 'rat'

<code>regconfig ,] querytext)</code>				
<code>querytree(query tsquery)</code>	text	获得tsquery的可索引部分	<code>querytree('foo & ! bar'::tsquery)</code>	'foo'
<code>setweight(tsvector, "char")</code>	tsvector	为tsvector的每个元素赋予权重	<code>setweight('fat:2.4 cat:3 rat:5B'::tsvector, 'A')</code>	'cat':3A 'fat':2A,4A 'rat':5A
<code>strip(tsvector)</code>	tsvector	从tsvector中删除位置和权重	<code>strip('fat:2,4 cat:3 rat:5A'::tsvector)</code>	'cat' 'fat' 'rat'
<code>to_tsquery([config regconfig ,] query text)</code>	tsquery	规范化单词并转换为tsquery	<code>to_tsquery('english', 'The & Fat & Rats')</code>	'fat' & 'rat'
<code>to_tsvector([config regconfig ,] documenttext)</code>	tsvector	将文档文本缩减为tsvector	<code>to_tsvector('english', 'The Fat Rats')</code>	'fat':2 'rat':3
<code>ts_headline([config regconfig ,] documenttext, query tsquery [, options text])</code>	text	显示查询匹配	<code>ts_headline('x y z', 'z'::tsquery)</code>	x y z
<code>ts_rank([weights float4[],] vector tsvector, query tsquery [, normalization integer])</code>	float4	用于查询的排名文档	<code>ts_rank(textsearch, query)</code>	0.818
<code>ts_rank_cd([weights float4[],] vectortsvector, query tsquery [, normalizationinteger])</code>	float4	使用封面密度对查询进行排名	<code>ts_rank_cd('0.1, 0.2, 0.4, 1.0', textsearch, query)</code>	2.01317
<code>ts_rewrite(query tsquery, target tsquery, substitute tsquery)</code>	tsquery	在查询中用substitute替换目标	<code>ts_rewrite('a & b'::tsquery, 'a'::tsquery, 'foo bar'::tsquery)</code>	'b' & ('foo' 'bar')
<code>ts_rewrite(query tsquery, select text)</code>	tsquery	使用targets和substitutes从SELECT命令中替换	<code>SELECT ts_rewrite('a & b'::tsquery, 'SELECT t,s FROM aliases')</code>	'b' & ('foo' 'bar')
<code>tsvector_update_trigger()</code>	trigger	用于自动tsvector列更新的触发器函数	<code>CREATE TRIGGER ... tsvector_update_trigger(tsvcol, 'pg_catalog.swedish', title, body)</code>	
<code>tsvector_update_trigger_column()</code>	trigger	用于自动tsvector列更新的触发器函数	<code>CREATE TRIGGER ... tsvector_update_trigger_column(tsvcol, configcol, title, body)</code>	

Note: 接受可选的regconfig参数的所有文本搜索函数将在省略该参数时使用`default_text_search_config` 指定的配置。

下表中的功能单独列出，因为它们通常不用于日常文本搜索操作。它们有助于开发和调试新的文本搜索配置。

Table 13. 文本搜索调试函数

函数	返回类型	描述	示例	结果
<code>ts_debug([config regconfig,] documenttext, OUT alias text, OUT description text, OUT token text, OUT dictionaries regdictionary[], OUT dictionary regdictionary, OUT lexemes text[])</code>	setof record	测试配置	<code>ts_debug('english', 'The Brightest supernovae')</code>	(asciword,"Word, all ASCII",The,{english_stem},english_stem,{}) ...
<code>ts_lexize(dict regdictionary, token text)</code>	text[]	测试字典	<code>ts_lexize('english_stem', 'stars')</code>	{star}
<code>ts_parse(parser_name)</code>	setof	测试解析器	<code>ts_parse('default', 'foo')</code>	(1,foo) ...

<code>text, document text, OUT tokid integer, OUT token text)</code>	<code>record</code>		<code>- bar')</code>	
<code>ts_parse(parser_oid oid, document text, OUT tokid integer, OUT token text)</code>	<code>setof record</code>	测试解析器	<code>ts_parse(3722, 'foo - bar')</code>	<code>(1,foo) ...</code>
<code>ts_token_type(parser_name text, OUT tokid integer, OUT alias text, OUT description text)</code>	<code>setof record</code>	获取解析器定义的token类型	<code>ts_token_type('default')</code>	<code>(1,asciword,"Word, all ASCII") ...</code>
<code>ts_token_type(parser_oid oid, OUT tokid integer, OUT alias text, OUT description text)</code>	<code>setof record</code>	获取解析器定义的token类型	<code>ts_token_type(3722)</code>	<code>(1,asciword,"Word, all ASCII") ...</code>
<code>ts_stat(sqlquery text, [weights text,] OUT word text, OUT ndocinteger, OUT nentry integer)</code>	<code>setof record</code>	获取tsvectorcolumn的统计信息	<code>ts_stat('SELECT vector from apod')</code>	<code>(foo,10,15) ...</code>

范围函数和运算符

有关范围类型的概述，请参阅[范围类型](#)。

下表显示了可用于范围类型的运算符。

Table 14. 范围运算符

运算符	描述	示例	结果
<code>=</code>	相等	<code>int4range(1,5) = '[1,4]':>int4range</code>	<code>t</code>
<code><></code>	不相等	<code>numrange(1.1,2.2) <> numrange(1.1,2.3)</code>	<code>t</code>
<code><</code>	小于	<code>int4range(1,10) < int4range(2,3)</code>	<code>t</code>
<code>></code>	大于	<code>int4range(1,10) > int4range(1,5)</code>	<code>t</code>
<code><=</code>	小于等于	<code>numrange(1.1,2.2) <= numrange(1.1,2.2)</code>	<code>t</code>
<code>>=</code>	大于等于	<code>numrange(1.1,2.2) >= numrange(1.1,2.0)</code>	<code>t</code>
<code>@></code>	包含范围	<code>int4range(2,4) @> int4range(2,3)</code>	<code>t</code>
<code>@></code>	包含元素	<code>'[2011-01-01,2011-03-01)':>tsrange @> '2011-01-10':>timestamp</code>	<code>t</code>
<code><@</code>	范围包含在	<code>int4range(2,4) <@ int4range(1,7)</code>	<code>t</code>
<code><@</code>	元素包含在	<code>42 <@ int4range(1,7)</code>	<code>f</code>
<code>&&</code>	重叠 (有共同点)	<code>int8range(3,7) && int8range(4,12)</code>	<code>t</code>
<code><<</code>	严格小于	<code>int8range(1,10) << int8range(100,110)</code>	<code>t</code>
<code>>></code>	严格大于	<code>int8range(50,60) >> int8range(20,30)</code>	<code>t</code>
<code>&<</code>	没有超越右边	<code>int8range(1,20) &< int8range(18,20)</code>	<code>t</code>
<code>&></code>	没有超越左边	<code>int8range(7,20) &> int8range(5,10)</code>	<code>t</code>
<code>- -</code>	毗邻	<code>numrange(1.1,2.2) - - numrange(2.2,3.3)</code>	<code>t</code>
<code>+</code>	并集	<code>numrange(5,15) + numrange(10,20)</code>	<code>[5,20)</code>
<code>*</code>	交集	<code>int8range(5,15) * int8range(10,20)</code>	<code>[10,15)</code>

-	差集	<code>int8range(5,15) - int8range(10,20)</code>	[5,10)
---	----	---	--------

简单比较运算符<，>，<=和>=首先比较下限，只有在相等的情况下，才比较上限。这些比较通常对范围不是很有用，但提供的是允许在范围内构建B树索引。

当涉及空范围时，左/右/邻接运算符总是返回false；也就是说，空范围不被认为是在任何其他范围之前或之后。

如果结果范围需要包含两个不相交的子范围，则并集和差异运算符将失败，因为无法表示这样的范围。

下表显示了可用于范围类型的函数。

Table 15. 范围函数

函数	返回类型	描述	示例	结果
<code>lower(anyrange)</code>	范围的元素类型	范围的下限	<code>lower(numrange(1.1,2.2))</code>	1.1
<code>upper(anyrange)</code>	范围的元素类型	范围的上限	<code>upper(numrange(1.1,2.2))</code>	2.2
<code>isempty(anyrange)</code>	boolean	范围是否空？	<code>isempty(numrange(1.1,2.2))</code>	false
<code>lower_inc(anyrange)</code>	boolean	是否包含下限？	<code>lower_inc(numrange(1.1,2.2))</code>	true
<code>upper_inc(anyrange)</code>	boolean	是否包含上限？	<code>upper_inc(numrange(1.1,2.2))</code>	false
<code>lower_inf(anyrange)</code>	boolean	下限是否无穷小？	<code>lower_inf('(',')'::daterange)</code>	true
<code>upper_inf(anyrange)</code>	boolean	上限是否无穷大？	<code>upper_inf('(',')'::daterange)</code>	true
<code>range_merge(anyrange, anyrange)</code>	anyrange	包括两个给定范围的最小范围	<code>range_merge('[1,2)'::int4range, '[3,4)'::int4range)</code>	[1,4)

如果范围为空或请求的边界为无限，则`lower`和`upper`函数返回null。对于空范围，`lower_inc`，`upper_inc`，`lower_inf`和`upper_inf`函数都返回false。

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

该规范描述了用于定义Greenplum MapReduce作业的文档格式和模式。

MapReduce 是由Google开发的一种编程模型，用于在商用服务器阵列上处理和生成大型数据集。Greenplum MapReduce允许熟悉MapReduce模型的程序员编写map和reduce函数，并将它们提交给Greenplum数据库并行引擎进行处理。

要使Greenplum能够处理MapReduce函数，请在文档中定义函数，然后将文档传递给Greenplum MapReduce程序gpmapreduce，以便由Greenplum数据库并行引擎执行。Greenplum数据库系统分配输入数据，在一组机器上执行程序，处理机器故障，并管理所需的机器间通信。

关于gpmapreduce的信息请见*Greenplum*数据库工具指南。

Parent topic: [Greenplum数据库参考指南](#)

Greenplum MapReduce文档格式

本节介绍Greenplum MapReduce文档格式的一些基础知识，以帮助您开始创建自己的Greenplum MapReduce文档。Greenplum使用[YAML 1.1](#)文档格式，然后实现自己的模式，以定义MapReduce作业的各个步骤。

所有Greenplum MapReduce文件必须首先声明它们正在使用的YAML规范的版本。之后，三个破折号（---）表示文档的开头，三个点（...）表示文档的结尾而不启动新文档。注释行以井号（#）为前缀。可以在同一个文件中声明多个Greenplum MapReduce文档：

```
%YAML 1.1
---
# Begin Document 1
# ...
---
# Begin Document 2
# ...
```

在Greenplum MapReduce文档中，有三种基本类型的数据或节点：标量，序列和映射。

标量是由空格缩进的基本文本字符串。如果您有跨越多行的标量输入，则前面的管道（|）表示文字样式，其中所有换行符都很重要。

Greenplum的PL/Perl语言

或者，前一个尖括号（>）将单个换行符折叠到具有相同缩进级别的后续行的空格。如果字符串包含具有保留含义的字符，则必须引用该字符串，或者必须使用反斜杠（\）转义特殊字符。

```
# Read each new line literally
somekey: /    this value contains two lines
            and each line is read literally
# Treat each new line as a space
anotherkey: >
            this value contains two lines
            but is treated as one continuous line
# This quoted string contains a special character
ThirdKey: "This is a string: not a mapping"
```

序列是列表，列表中的每个条目都在其自己的行上，用短划线和空格（-）表示。或者，您可以将内联序列指定为方括号内的逗号分隔列表。序列提供一组数据并为其提供订单。将列表加载到Greenplum MapReduce程序时，将保留订单。

```
# list sequence
- this
- is
- a list
- with
- five scalar values
# inline sequence
[this, is, a list, with, five scalar values]
```

映射用于将数据值与称为键的标识符配对。映射对每个key:value对使用冒号和空格（:），或者也可以内联指定为花括号内的逗号分隔列表。该密钥用作从映射中检索数据的索引。

```
# a mapping of items
title: War and Peace
author: Leo Tolstoy
date: 1865
# same mapping written inline
{title: War and Peace, author: Leo Tolstoy, date: 1865}
```

密钥用于将元信息与每个节点相关联，并指定预期的节点类型（标量，序列或映射）。有关Greenplum MapReduce程序所需的键，请参阅[Greenplum MapReduce文档模式](#)。

Greenplum MapReduce程序按顺序处理文档的节点，并使用缩进（空格）来确定文档层次结构和节点之间的关系。使用空白区域非常重要。不应仅将白色空间用于格式化目的，并且根本不应使用制表符。

Greenplum MapReduce文档模式

Greenplum MapReduce使用YAML文档框架并实现自己的YAML模式。Greenplum MapReduce文档的基本结构是：

```
%YAML 1.1
---
VERSION: 1.0.0.2
DATABASE: dbname
USER: db_username
HOST: master_hostname
PORT: master_port
```

```
DEFINE:
- INPUT:
  NAME: input_name
  FILE:
    - hostname:/path/to/file
GPFDIST:
  - hostname:port/file_pattern
TABLE: table_name
QUERY: SELECT_statement
EXEC: command_string
COLUMNS:
  - field_name data_type
FORMAT: TEXT | CSV
DELIMITER: delimiter_character
ESCAPE: escape_character
NULL: null_string
QUOTE: csv_quote_character
ERROR_LIMIT: integer
ENCODING: database_encoding
```

```
- OUTPUT:
  NAME: output_name
  FILE: file_path_on_client
  TABLE: table_name
  KEYS:
    - column_name
  MODE: REPLACE | APPEND
```

```
- MAP:
  NAME: function_name
  FUNCTION: function_definition
  LANGUAGE: perl | python | c
  LIBRARY: /path/filename.so
  PARAMETERS:
    - nametype
  RETURNS:
    - nametype
  OPTIMIZE: STRICT IMMUTABLE
```

MODE: SINGLE | MULTI

- TRANSITION | CONSOLIDATE | FINALIZE:

NAME: *function_name*
FUNCTION: *function_definition*
LANGUAGE: perl | python | c
LIBRARY: */path/filename.so*

PARAMETERS:- *nametype***RETURNS**:- *nametype***OPTIMIZE**: STRICT IMMUTABLE**MODE**: SINGLE | MULTI

- REDUCE:

NAME: *reduce_job_name*
TRANSITION: *transition_function_name*
CONSOLIDATE: *consolidate_function_name*
FINALIZE: *finalize_function_name*
INITIALIZE: *value*

KEYS:- *key_name*

- TASK:

NAME: *task_name*
SOURCE: *input_name*
MAP: *map_function_name*
REDUCE: *reduce_function_name*

EXECUTE

- RUN:

SOURCE: *input_or_task_name*
TARGET: *output_name*
MAP: *map_function_name*
REDUCE: *reduce_function_name...*

VERSION

必须。Greenplum MapReduce YAML规范的版本。当前版本为1.0.0.1。

DATABASE

可选。指定Greenplum中要连接的数据库。如果未指定，则默认为默认数据库或\$PGDATABASE（如果已设置）。

USER

可选。指定要用于连接的数据库角色。如果未指定，则默认为当前用户或\$PGUSER（如果已设置）。您必须是Greenplum超级用户才能运行用不受信任的Python和Perl编写的函数。常规数据库用户可以运行用可信Perl编写的函数。您还必须是数据库超级

用户才能运行包含FILE, GPFDIST 和EXEC输入类型的MapReduce作业。

HOST

可选。指定Greenplum master主机名。如果未指定，则默认为localhost或\$PGHOST（如果已设置）。

PORT

可选。指定Greenplum主端口。如果未指定，则默认为5432或\$PGPORT（如果已设置）。

DEFINE

必须。此MapReduce文档的一系列定义。DEFINE部分必须至少有一个INPUT定义。

INPUT

必须。定义输入数据。每个MapReduce文档必须至少定义一个输入。文档中允许多个输入定义，但每个输入定义只能指定其中一种访问类型：文件，gpfdist 文件分发程序，数据库中的表，SQL命令或操作系统命令。有关gpfdist 的信息，请参阅*Greenplum*数据库实用程序指南。

NAME

此输入的名称。关于此MapReduce作业中其他对象的名称（例如map函数，task，reduce函数和输出名称），名称必须是唯一的。此外，名称不能与数据库中的现有对象（例如表，函数或视图）冲突。

FILE

一个或多个输入文件的序列，格式为：seghostname:/path/to/filename。您必须是Greenplum数据库超级用户才能使用FILE输入运行MapReduce作业。该文件必须位于Greenplum segment主机上。

GPFDIST

一个或多个运行gpfdist文件分发程序的序列，格式为：hostname[:port]/file_pattern。除非服务器配置参数[服务器配置参数](#)设置为on，否则您必须是Greenplum数据库超级用户才能使用GPFDIST输入运行MapReduce作业。

TABLE

数据库中现有表的名称。

QUERY

要在数据库中运行的SQL SELECT命令。

EXEC

要在Greenplum segment主机上运行的操作系统命令。

默认情况下，该命令由系统中的所有segment实例运行。例如，如果每个segment主机有四个segment实例，则该命令将在每个主机上运行四次。您必须是Greenplum数据库超级用户才能使用EXEC输入运行MapReduce作业，并且服务器配置参数[服务器配置参数](#)设置为on。

COLUMNS

可选。列指定为：column_name [data_type]。如果未指定，则默认值为value text。

[DELIMITER](#)字符用于分隔两个数据值字段（列）。行由换行符（0x0a）确定。

FORMAT

可选。指定数据的格式 - 分隔文本（TEXT）或逗号分隔值（CSV）格式。如果未指定数据格式，则默认为TEXT。

DELIMITER

可选[FILE](#), [GPFDIST](#) 和[EXEC](#)输入。指定用于分隔数据值的单个字符。默认值为TEXT模式下的制表符，CSV模式下为逗号。分隔符字符只能出现在任意两个数据值字段之间。不要在行的开头或结尾放置分隔符。

ESCAPE

对于[FILE](#), [GPFDIST](#) 和[EXEC](#)输入可选。指定用于C转义序列的单个字符（例如\n,\t,\100等）以及转义可能以行或列分隔符形式取出的数据字符。确保选择实际列数据中未使用的转义字符。默认转义字符是文本格式文件的\（反斜杠）和CSV格式文件的"（双引号），但是可以指定另一个字符来表示转义。也可以通过指定禁用转义值'OFF'作为转义值。这对于诸如文本格式的Web日志数据之类的数据非常有用，这些数据具有许多不打算转义的嵌入式反斜杠。

NULL

对于[FILE](#), [GPFDIST](#) 和[EXEC](#)输入可选。指定表示空值的字符串。默认值为TEXT格式的\N，以及CSV格式没有引号的空值。如果您不想将空值与空字符串区分开来，即使在TEXT模式下，您可能更喜欢空字符串。与此字符串匹配的任何输入数据项都将被视为空值。

QUOTE

对于[FILE](#), [GPFDIST](#) 和[EXEC](#)输入可选。指

定CSV格式文件的引用字符。默认值为双引号（"）。在CSV格式的文件中，如果数据值字段包含任何逗号或嵌入的新行，则必须用双引号括起来。包含双引号字符的字段必须用双引号括起来，并且嵌入双引号必须由一对连续的双引号表示。始终正确打开和关闭引号以便正确解析数据行非常重要。

ERROR_LIMIT

如果输入行具有格式错误，则只要在输入处理期间未在任何Greenplum segment实例上达到错误限制计数，它们将被丢弃。如果未达到错误限制，则将处理所有正常行并丢弃任何错误行。

ENCODING

用于数据的字符集编码。指定字符串常量（例如'SQL_ASCII'），整数编码号或DEFAULT以使用默认客户端编码。有关更多信息，请参阅[字符集支持](#)。

OUTPUT

可选。定义输出此MapReduce作业的格式化数据的位置。如果未定义输出，则默认为STDOUT（客户端的标准输出）。您可以将输出发送到客户端主机上的文件或数据库中的现有表。

NAME

此输出的名称。默认输出名称为STDOUT。关于MapReduce作业中其他对象的名称（例如map函数，task，reduce函数和输入名称），名称必须是唯一的。此外，名称不能与数据库中的现有对象（例如表，函数或视图）冲突。

FILE

指定MapReduce客户端计算机上的文件位置，以如下格式输出数据：/path/to/filename。

TABLE

指定数据库中用于输出数据的表的名称。如果在运行MapReduce作业之前该表不存在，则将使用[KEYS](#)指定的分发策略创建该表。

KEYS

[TABLE](#)输出的可选项。指定要用作Greenplum数据库分发键的列。如果[EXECUTE](#)任务包含[REDUCE](#)定义，则默认情况下REDUCE键将用作表分发键。否则，表的第一列将用作分发键。

MODE

[TABLE](#)输出的可选项。如果未指定，则默认为创建表（如果该表尚不存在），但如果表存在则输出错误。声明APPEND将输出数据添加到现有表（前提是

表模式与输出格式匹配），而不删除任何现有数据。如果表存在，则声明REPLACE将删除该表，然后重新创建它。如果不存在，APPEND和REPLACE都将创建一个新表。

MAP

必须。每个MAP函数采用以 (key , value) 对构造的数据，处理每对，并生成零个或多个输出 (key , value) 对。然后，Greenplum MapReduce框架从所有输出列表中收集具有相同密钥的所有对，并将它们组合在一起。然后将此输出传递给REDUCE任务，该任务由TRANSITION | CONSOLIDATE | FINALIZE函数组成。有一个名为IDENTITY的预定义MAP函数，它返回的 (key , value) 对不变。虽然 (key , value) 是默认参数，但您可以根据需要指定其他原型。

TRANSITION | CONSOLIDATE | FINALIZE

TRANSITION, CONSOLIDATE和FINALIZE 都是REDUCE 的组成部分。需要TRANSITION函数。CONSOLIDATE和FINALIZE 函数是可选的。默认情况下，所有将state 作为其输入PARAMETERS 的第一个，但也可以定义其他原型。

TRANSITION函数遍历给定键的每个值，并在state变量中累积值。当在键的第一个值上调用转换函数时，state将设置为REDUCE 作业的INITIALIZE指定的值（或数据类型的默认状态值）。转换需要两个参数作为输入；密钥减少的当前状态和下一个值，然后产生一个新state。

如果指定了CONSOLIDATE函数，则在segment级别执行TRANSITION处理，然后在Greenplum互连上重新分配密钥以进行最终聚合（两阶段聚合）。仅重新分配给定密钥的结果state 值，从而导致更低的互连流量和更高的并行度。CONSOLIDATE像TRANSITION一样处理，除了 $(state + value) \Rightarrow state$ ，它是 $(state + state) \Rightarrow state$ 。如果指定了FINALIZE 函数，它将采用CONSOLIDATE（如果存在）或TRANSITION生成的最终state ，并在发出最终结果之前进行任何最终处理。TRANSITION和CONSOLIDATE函数不能返回一组值。如果需要REDUCE作业来返回一个集合，则需要FINALIZE 将最终状态转换为一组输出值。

NAME

必须。函数的名称。关于此MapReduce作业中其他对象的名称（例如函数，任务，输入和输出名称），名

称必须是唯一的。您还可以指定Greenplum数据库内置函数的名称。如果使用内置函数，请不要提供`LANGUAGE`或`FUNCTION`正文。

FUNCTION

可选。使用指定的`LANGUAGE`指定函数的完整主体。如果未指定`FUNCTION`，则使用与`NAME`对应的内置数据库函数。

LANGUAGE

使用`FUNCTION`时需要。指定用于解释函数的实现语言。此版本具有对perl, python和c的语言支持。如果调用内置数据库函数，则不应指定`LANGUAGE`。

LIBRARY

`LANGUAGE`为C时必需（不允许用于其他语言函数）。要使用此属性，`VERSION`必须为1.0.0.2。必须在运行MapReduce作业之前安装指定的库文件，并且该文件必须存在于所有Greenplum主机（master和segment）上的相同文件系统位置。

PARAMETERS

可选。函数输入参数。默认类型是`text`。

`MAP default - key text, value text`

`TRANSITION default - state text, value text`

`CONSOLIDATE default - state1 text, state2 text` (必须具有相同数据类型的两个输入参数)

`FINALIZE default - state text` (仅限单个参数)

RETURNS

可选。默认返回类型是`text`。

`MAP default - key text, value text`

`TRANSITION default - state text` (仅限单个参数)

`CONSOLIDATE default - state text` (仅限单个参数)

`FINALIZE default - value text`

OPTIMIZE

该函数的可选优化参数：

`STRICT` - 函数不受NULL值的影响

IMMUTABLE - 函数将始终返回给定输入的相同值

MODE

可选。指定函数返回的行数。

MULTI - 每个输入记录返回0行或更多行。函数的返回值必须是要返回的行数组，或者必须使用Python中的`yield`或Perl中的`return_next`将函数写为迭代器。**MULTI**是**MAP**和**FINALIZE**函数的默认模式。

SINGLE - 每个输入记录只返回一行。

SINGLE是**TRANSITION**和**CONSOLIDATE**函数支持的唯一模式。当与**MAP**和**FINALIZE**函数一起使用时，**SINGLE**模式可以提供适度的性能改进。

REDUCE

必须。**REDUCE**定义命名**TRANSITION** | **CONSOLIDATE** | **FINALIZE**函数，包括将(`key , value`)对缩减到最终结果集。您还可以执行几个预定义的**REDUCE**作业，这些作业都在名为`value`的列上运行：

IDENTITY - 返回(键, 值)对不变

SUM - 计算数值数据的总和

AVG - 计算数字数据的平均值

COUNT - 计算输入数据的计数

MIN - 计算数值数据的最小值

MAX - 计算数值数据的最大值

NAME

必须。这个**REDUCE**工作的名称。关于此**MapReduce**作业中的其他对象的名称(函数, 任务, 输入和输出名称)，名称必须是唯一的。此外，名称不能与数据库中的现有对象(例如表, 函数或视图)冲突。

TRANSITION

必须。**TRANSITION**函数名称。

CONSOLIDATE

可选。**CONSOLIDATE**函数名称。

FINALIZE

可选。**FINALIZE**函数名称。

INITIALIZE

`text`和`float`数据类型的可选项。所有其他数据类型都需要。文本的默认值为`''`。`float`的默认值为`0.0`。设置`TRANSITION`函数的初始`state`值。

KEYS

可选。默认为`[key, *]`。使用多列缩减时，可能需要指定哪些列是键列，哪些列是值列。默认情况下，未传递给`TRANSITION`函数的任何输入列都是键列，名为`key`的列始终是键列，即使它传递给`TRANSITION`函数也是如此。特殊指示符`*`表示未传递给`TRANSITION`函数的所有列。如果该指示符不存在于键列表中，则丢弃任何不匹配的列。

TASK

可选。`TASK`在Greenplum MapReduce作业管道中定义了完整的端到端`INPUT/MAP/REDUCE`阶段。它与`EXECUTE`类似，但不会立即执行。可以被称为`INPUT`的任务对象进入进一步处理阶段。

NAME

必须。此任务的名称。关于此MapReduce作业中其他对象的名称（例如`map`函数，`reduce`函数，输入和输出名称），名称必须是唯一的。此外，名称不能与数据库中的现有对象（例如表，函数或视图）冲突。

SOURCE

`INPUT`或其他`TASK`的名称。

MAP

可选。`MAP`函数的名称。如果未指定，则默认为`IDENTITY`。

REDUCE

可选。`REDUCE`函数的名称。如果未指定，则默认为`IDENTITY`。

EXECUTE

必须。`EXECUTE`定义Greenplum MapReduce作业管道中的最终`INPUT/MAP/REDUCE`阶段。

RUN

SOURCE

必须。`INPUT`或`TASK`的名称。

TARGET

可选。`OUTPUT`的名称。默认值为`STDOUT`。

MAP

可选。 **MAP** 函数名称。 如果未指定， 默认为 **IDENTITY**。

REDUCE

可选。 **REDUCE** 函数的名称。 默认为 **IDENTITY**。

示例Greenplum MapReduce文档

```

# This example MapReduce job processes documents and looks
for keywords in them.
# It takes two database tables as input:
#   - documents (doc_id integer, url text, data text)
#   - keywords (keyword_id integer, keyword text)#
# The documents data is searched for occurrences of
keywords and returns results of
# url, data and keyword (a keyword can be multiple words,
such as "high performance # computing")
%YAML 1.1
---
VERSION:1.0.0.1

# Connect to Greenplum Database using this database and
role
DATABASE:webdata
USER:jsmith

# Begin definition section
DEFINE:

# Declare the input, which selects all columns and rows
from the
# 'documents' and 'keywords' tables.
- INPUT:
NAME:doc
TABLE:documents
- INPUT:
NAME:kw
TABLE:keywords
# Define the map functions to extract terms from documents
and keyword
# This example simply splits on white space, but it would
be possible
# to make use of a python library like nltk (the natural
language toolkit)
# to perform more complex tokenization and word stemming.
- MAP:
NAME:doc_map
LANGUAGE:python
FUNCTION:|
    i = 0                      # the index of a word within the
document
terms = {}# a hash of terms and their indexes within the

```

document

```

# Lower-case and split the text string on space
for term in data.lower().split():
    i = i + 1# increment i (the index)

        # Check for the term in the terms list:
        # if stem word already exists, append the i value
        # to the array entry
        # corresponding to the term. This counts multiple
        # occurrences of the word.
        # If stem word does not exist, add it to the
        # dictionary with position i.
        # For example:
        # data: "a computer is a machine that manipulates data"
        # "a" [1, 4]
        # "computer" [2]
        # "machine" [3]
        # ...
        if term in terms:
            terms[term] += ','+str(i)
        else:
            terms[term] = str(i)

# Return multiple lines for each document. Each line
# consists of
# the doc_id, a term and the positions in the data where
# the term appeared.
# For example:
#     (doc_id => 100, term => "a", [1,4]
#     (doc_id => 100, term => "computer", [2]
#     ...
for term in terms:
yield([doc_id, term, terms[term]])
OPTIMIZE:STRICT IMMUTABLE
PARAMETERS:
- doc_id integer
    - data text
RETURNS:
- doc_id integer
    - term text
    - positions text

# The map function for keywords is almost identical to
# the one for documents
# but it also counts of the number of terms in the
# keyword.
- MAP:
NAME:kw_map
LANGUAGE:python
FUNCTION:|
    i = 0
    terms = {}
    for term in keyword.lower().split():
        i = i + 1
        if term in terms:
            terms[term] += ','+str(i)
        else:

```

```

        terms[term] = str(i)

    # output 4 values including i (the total count for term in
    terms):
    yield([keyword_id, i, term, terms[term]])
        OPTIMIZE:STRICT IMMUTABLE
PARAMETERS:
- keyword_id integer
    - keyword text
RETURNS:
- keyword_id integer
    - nterms integer
    - term text
    - positions text

# A TASK is an object that defines an entire
INPUT/MAP/REDUCE stage
# within a Greenplum MapReduce pipeline. It is like
EXECUTION, but it is
# executed only when called as input to other processing
stages.
# Identify a task called 'doc_prep' which takes in the
'doc' INPUT defined earlier
# and runs the 'doc_map' MAP function which returns doc_id,
term, [term_position]
- TASK:
NAME:doc_prep
SOURCE:doc
MAP:doc_map

# Identify a task called 'kw_prep' which takes in the 'kw'
INPUT defined earlier
# and runs the kw_map MAP function which returns kw_id,
term, [term_position]
- TASK:
NAME:kw_prep
SOURCE:kw
MAP:kw_map

# One advantage of Greenplum MapReduce is that MapReduce
tasks can be
# used as input to SQL operations and SQL can be used to
process a MapReduce task.
# This INPUT defines a SQL query that joins the output of
the 'doc_prep'
# TASK to that of the 'kw_prep' TASK. Matching terms are
output to the 'candidate'
# list (any keyword that shares at least one term with the
document).
- 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 doc_prep doc INNER JOIN kw_prep kw ON
(doc.term = kw.term)

```

```

# In Greenplum MapReduce, a REDUCE function is comprised of
# one or more functions.
# A REDUCE has an initial 'state' variable defined for each
# grouping key. that is
# A TRANSITION function adjusts the state for every value
# in a key grouping.
# If present, an optional CONSOLIDATE function combines
# multiple
# 'state' variables. This allows the TRANSITION function to
# be executed locally at
# the segment-level and only redistribute the accumulated
# 'state' over
# the network. If present, an optional FINALIZE function
# can be used to perform
# final computation on a state and emit one or more rows of
# output from the state.
#
# This REDUCE function is called 'term_reducer' with a
# TRANSITION function
# called 'term_transition' and a FINALIZE function called
# 'term_finalizer'
- REDUCE:
NAME:term_reducer
TRANSITION:term_transition
FINALIZE:term_finalizer

- TRANSITION:
NAME:term_transition
LANGUAGE:python
PARAMETERS:
- state text
    - term text
    - nterms integer
    - doc_positions text
    - kw_positions text
FUNCTION: |

# 'state' has an initial value of '' and is a colon
# delimited set
# of keyword positions. keyword positions are comma
# delimited sets of
# integers. For example, '1,3,2:4:'
# If there is an existing state, split it into the
# set of keyword positions
# otherwise construct a set of 'nterms' keyword positions
- all empty
if state:
    kw_split = state.split(':')
else:
    kw_split = []
    for i in range(0,nterms):
        kw_split.append('')

# 'kw_positions' is a comma delimited field of
# integers indicating what
# position a single term occurs within a given keyword.

```

```

        # Splitting based on ',' converts the string into a
        python list.

        # add doc_positions for the current term
        for kw_p in kw_positions.split(','):
            kw_split[int(kw_p)-1] = doc_positions

            # This section takes each element in the 'kw_split'
            array and strings
            # them together placing a ':' in between each
            element from the array.
            # For example: for the keyword "computer software computer
            hardware",
            # the 'kw_split' array matched up to the document
            data of
            # "in the business of computer software software
            engineers"
            # would look like: ['5', '6,7', '5', '']
            # and the outstate would look like: 5:6,7:5:
            outstate = kw_split[0]
            for s in kw_split[1:]:
                outstate = outstate + ':' + s
            return outstate

        - FINALIZE:
NAME: term_finalizer
LANGUAGE: python
RETURNS:
        - count integer
MODE: MULTI
FUNCTION: |
        if not state:
            return 0
        kw_split = state.split(':')

        # This function does the following:
# 1) Splits 'kw_split' on ':'
        #     for example, 1,5,7:2,8 creates '1,5,7' and
        '2,8'
        # 2) For each group of positions in 'kw_split', splits the
        set on ','
        #     to create ['1','5','7'] from Set 0: 1,5,7 and
        #     eventually ['2', '8'] from Set 1: 2,8
        # 3) Checks for empty strings
        # 4) Adjusts the split sets by subtracting the position of
        the set
        #     in the 'kw_split' array
# ['1','5','7'] - 0 from each element = ['1','5','7']
# ['2', '8'] - 1 from each element = ['1', '7']
        # 5) Resulting arrays after subtracting the offset
        in step 4 are
        #     intersected and their overlapping values kept:
        #     ['1','5','7'].intersect['1', '7'] = [1,7]
        # 6) Determines the length of the intersection,
        which is the number of
        # times that an entire keyword (with all its pieces)
        matches in the
        #     document data.
previous = None

```

```

        for i in range(0,len(kw_split)):
            isplit = kw_split[i].split(',')
            if any(map(lambda(x): x == '', isplit)):
                return 0
            adjusted = set(map(lambda(x): int(x)-i, isplit))
            if (previous):
                previous = adjusted.intersection(previous)
            else:
                previous = adjusted

        # return the final count
if previous:
    return len(previous)

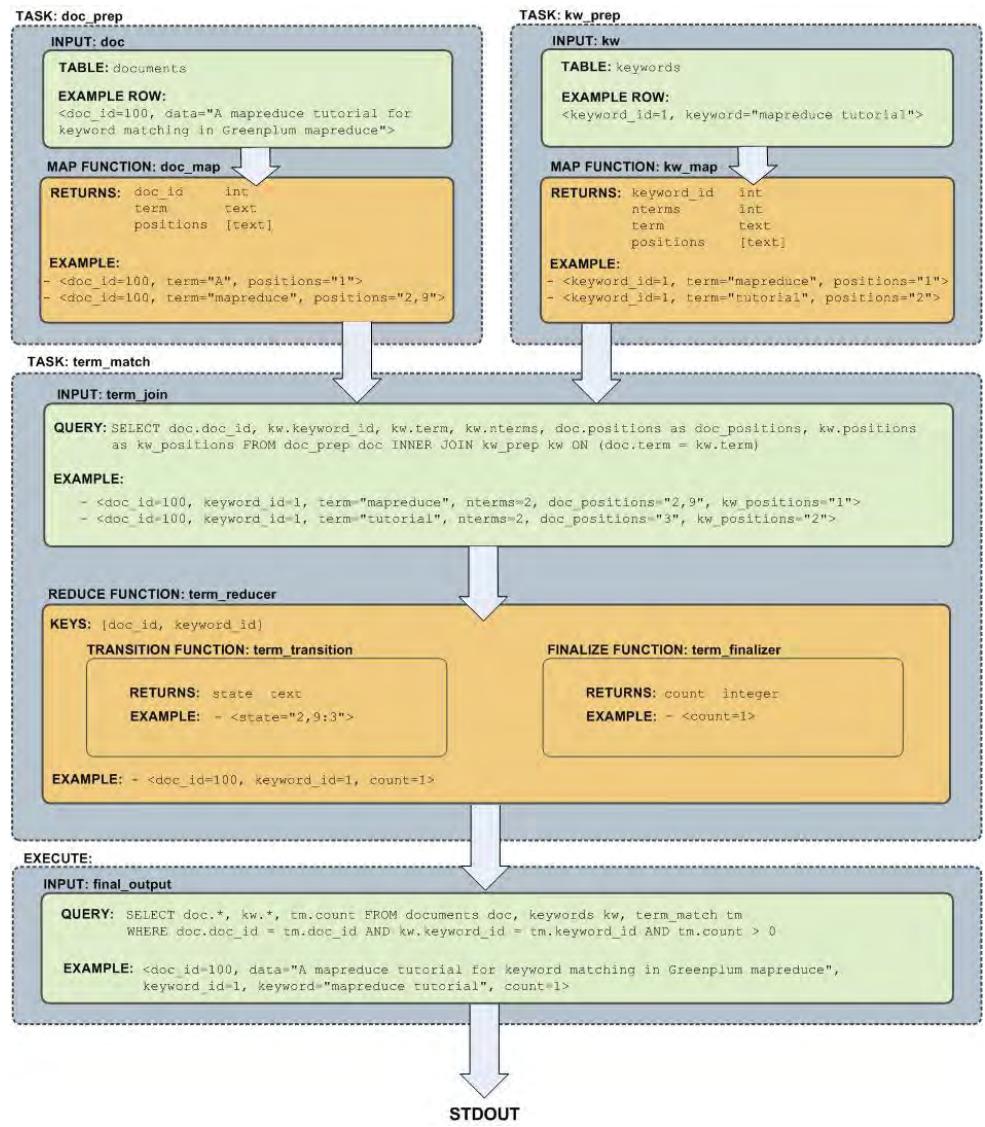
# Define the 'term_match' task which is then executed as part
# of the 'final_output' query. It takes the INPUT 'term_join' defined
# earlier and uses the REDUCE function 'term_reducer' defined earlier
- TASK:
NAME:term_match
SOURCE:term_join
REDUCE:term_reducer
- INPUT:
NAME:final_output
QUERY: |
    SELECT doc.*, kw.*, tm.count
    FROM documents doc, keywords kw, term_match tm
    WHERE doc.doc_id = tm.doc_id
    AND kw.keyword_id = tm.keyword_id
    AND tm.count > 0

# Execute this MapReduce job and send output to STDOUT
EXECUTE:
- RUN:
SOURCE:final_output
TARGET:STDOUT

```

MapReduce示例的流程图

下图显示了示例中定义的MapReduce作业的作业流程：





Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

本节概述了Greenplum数据库PL/pgSQL语言。

- [关于Greenplum数据库PL/pgSQL](#)
- [PL/pgSQL计划缓存](#)
- [PL/pgSQL示例](#)
- [参考](#)

Parent topic: [Greenplum数据库参考指南](#)

关于Greenplum数据库PL/pgSQL

Greenplum数据库PL/pgSQL是一种可加载的过程语言，默认情况下使用Greenplum数据库进行安装和注册。您可以使用SQL语句，函数和运算符创建用户定义的函数。

使用PL/pgSQL，您可以在数据库服务器内对计算块和一系列SQL查询进行分组，从而具有过程语言的强大功能和SQL的易用性。此外，使用PL/pgSQL，您可以使用Greenplum数据库SQL的所有数据类型，运算符和函数。

PL/pgSQL语言是Oracle PL/SQL的子集。Greenplum数据库PL/pgSQL基于Postgres PL/pgSQL。Postgres PL/pgSQL文档位于<https://www.postgresql.org/docs/9.4/plpgsql.html>

使用PL/pgSQL函数时，函数属性会影响Greenplum数据库创建查询计划的方式。您可以将属性IMMUTABLE，STABLE或VOLATILE指定为LANGUAGE子句的一部分，以对函数类型进行分类。有关创建函数和函数属性的信息，请参阅Greenplum数据库参考指南中的[CREATE FUNCTION](#)命令。

您可以将PL/SQL代码块作为匿名代码块运行。请参阅Greenplum数据库参考指南中的[DO](#)命令。

Greenplum数据库SQL限制

使用Greenplum数据库PL/pgSQL时，限制包括

Greenplum的PL/Perl语言

- 不支持触发器
- 游标只向前移动（不可滚动）
- 不支持可更新游标（`UPDATE ... WHERE CURRENT OF`和`DELETE ... WHERE CURRENT OF`）。

有关Greenplum数据库SQL一致性的信息，请参阅*Greenplum*数据库参考指南中的[Greenplum 特点概要](#)。

PL/pgSQL语言

PL/pgSQL是一种块结构语言。 函数定义的完整文本必须是块。 块定义为：

```
[ label ]
[ DECLARE
    declarations ]
BEGIN
    statements
END [ label ];
```

块中的每个声明和每个语句都以分号（;）结束。 出现在另一个块中的块在END之后必须有分号，如上一个块所示。 结束函数体的END不需要分号。

仅当您要识别要在EXIT语句中使用的块或限定块中声明的变量的名称时，才需要label。 如果在END之后提供label，则它必须与块开头的label匹配。

Important: 对于使用用于事务控制的数据库命令PL/pgSQL分组语句，不要混淆使用BEGIN和END关键字。 PL/pgSQL BEGIN和END关键字仅用于分组；他们不会开始或结束事务。 函数总是在外部查询建立的事务中执行 - 它们无法启动或提交该事务，因为它们没有上下文可供执行。 但是，包含EXCEPTION子句的PL/pgSQL块有效地形成了一个子事务，可以在不影响外部事务的情况下回滚。 有关EXCEPTION子句的更多信息，请参阅 <https://www.postgresql.org/docs/9.4/plpgsql-control-structures.html#PLPGSQL-ERROR-TRAPPING>。

关键字不区分大小写。 除非双引号，否则标识符将隐式转换为小写，就像在普通SQL命令中一样。

PL/pgSQL代码中的注释与普通SQL中的注释相同：

- 双短划线（--）开始一条延伸到该行末尾的注释。

- /* 启动一个块注释，扩展到匹配的 */。
阻止评论嵌套。

块的语句部分中的任何语句都可以是子块。 子块可用于逻辑分组或将变量本地化为一小组语句。

在子块中声明的变量在子块的持续时间内屏蔽外部块的任何类似命名的变量。 如果使用块的标签限定其名称，则可以访问外部变量。 例如，此函数多次声明一个名为quantity的变量：

```
CREATE FUNCTION testfunc() RETURNS integer AS $$  
<< outerblock >>  
DECLARE  
    quantity integer := 30;  
BEGIN  
    RAISE NOTICE 'Quantity here is %', quantity; -- Prints  
30  
    quantity := 50;  
    --  
    -- Create a subblock  
    --  
    DECLARE  
        quantity integer := 80;  
    BEGIN  
        RAISE NOTICE 'Quantity here is %', quantity; --  
Prints 80  
        RAISE NOTICE 'Outer quantity here is %',  
outerblock.quantity; -- Prints 50  
    END;  
    RAISE NOTICE 'Quantity here is %', quantity; -- Prints  
50  
    RETURN quantity;  
END;  
$$ LANGUAGE plpgsql;
```

执行SQL命令

您可以使用PL/pgSQL语句执行SQL命令，例如EXECUTE, PERFORM和SELECT ... INTO。有关PL/pgSQL语句的信息，请参阅 <https://www.postgresql.org/docs/9.4/plpgsql-statements.html>。

Note: EXECUTE语句不支持PL/pgSQL语句SELECT INTO。

PL/pgSQL计划缓存

PL/pgSQL函数的波动率分类对Greenplum数据库如何缓存引用该函数的计划有影响。有关Greenplum数据库函数波动率类别的计划缓存注意事项的信息，请参阅*Greenplum*数据库管理员指南中的[函数波动率和计划缓存](#)。

当PL/pgSQL函数在数据库会话中第一次执行时，PL/pgSQL解释器会解析函数的SQL表达式和命令。解释器创建一个准备好的执行计划，因为每个表达式和SQL命令首先在函数中执行。PL/pgSQL解释器在数据库连接的生命周期中重用特定表达式和SQL命令的执行计划。虽然这种重用大大减少了解析和生成计划所需的总时间，但是直到执行该部分函数的运行时才能检测到特定表达式或命令中的错误。

如果对查询中使用的任何关系进行任何架构更改，或者重新定义查询中使用的任何用户定义函数，Greenplum数据库将自动重新规划已保存的查询计划。这使得在大多数情况下重复使用准备好的计划是透明的。

您在PL/pgSQL函数中使用的SQL命令必须在每次执行时引用相同的表和列。您不能将参数用作SQL命令中的表或列的名称。

PL/pgSQL为实际参数类型的每个组合缓存一个单独的查询计划，您可以在其中调用多态函数以确保数据类型差异不会导致意外故障。

有关PL/pgSQL语言中计划缓存注意事项的详细讨论，请参阅PostgreSQL[计划缓存](#) 文档。

PL/pgSQL示例

以下是PL/pgSQL用户定义函数的示例。

示例：函数参数的别名

传递给函数的参数使用诸如\$1, \$2之类的标识符命名。（可选）可以为\$n参数名称声明别名，以提高可读性。然后可以使用别名或数字标识符来引用参数值。

有两种方法可以创建别名。首选方法是在CREATE FUNCTION命令中为参数指定名称，例如：

```
CREATE FUNCTION sales_tax(subtotal real) RETURNS real AS $$  
BEGIN  
    RETURN subtotal * 0.06;  
END;
```

```
$$ LANGUAGE plpgsql;
```

您还可以使用声明语法显式声明别名：

```
name ALIAS FOR $n;
```

此示例使用DECLARE语法创建相同的函数。

```
CREATE FUNCTION sales_tax(real) RETURNS real AS $$  
DECLARE  
    subtotal ALIAS FOR $1;  
BEGIN  
    RETURN subtotal * 0.06;  
END;  
$$ LANGUAGE plpgsql;
```

示例：使用表列的数据类型

声明变量时，可以使用%TYPE构造指定变量或表列的数据类型。这是声明其类型是表列的数据类型的变量的语法：

```
name table.column_name%TYPE;
```

您可以使用%TYPE构造来声明将保存数据库值的变量。例如，假设您的users表中有一个名为user_id的列。声明名为my_userid的变量，其数据类型与users.user_id列相同：

```
my_userid users.user_id%TYPE;
```

%TYPE在多态函数中特别有价值，因为内部变量所需的数据类型可能会从一个调用更改为下一个调用。通过将%TYPE应用于函数的参数或结果占位符，可以创建适当的变量。

示例：基于表行的复合类型

复合类型的变量称为行变量。以下语法基于表行声明复合变量：

```
name table_name%ROWTYPE;
```

这样的行变量可以保存SELECT或FOR查询结果的整行，只要该查询的

列集与变量的声明类型匹配即可。 使用通常的点表示法访问行值的各个字段，例如`rowvar.column`。

函数的参数可以是复合类型（完整的表行）。 在这种情况下，相应的标识符`$n`将是行变量，并且可以从中选择字段，例如`$1.user_id`。

只能在行类型变量中访问表行的用户定义列，而不能访问OID或其他系统列。 行类型的字段为数据类型（如`char(n)`）继承表的字段大小或精度。

下一个示例函数使用行变量复合类型。 在创建函数之前，使用此命令创建函数使用的表。

```
CREATE TABLE table1 (
    f1 text,
    f2 numeric,
    f3 integer
) distributed by (f1);
```

此`INSERT`命令将数据添加到表中。

```
INSERT INTO table1 values
('test1', 14.1, 3),
('test2', 52.5, 2),
('test3', 32.22, 6),
('test4', 12.1, 4);
```

此函数使用基于`table1`的列`%TYPE`变量和`%ROWTYPE`复合变量。

```
CREATE OR REPLACE FUNCTION t1_calc( name text ) RETURNS
integer
AS $$$
DECLARE
    t1_row    table1%ROWTYPE;
    calc_int  table1.f3%TYPE;
BEGIN
    SELECT * INTO t1_row FROM table1 WHERE table1.f1 = $1 ;
    calc_int = (t1_row.f2 * t1_row.f3)::integer ;
    RETURN calc_int ;
END;
$$ LANGUAGE plpgsql VOLATILE;
```

Note: 前一个函数被归类为`VOLATILE`函数，因为函数值可能在单个表扫描中发生变化。

以下`SELECT`命令使用该函数。

```
select t1_calc( 'test1' );
```

Note: 示例PL/pgSQL函数将SELECT与INTO子句一起使用。它与SQL命令SELECT INTO不同。如果要从PL/pgSQL函数内的SELECT结果创建表，请使用SQL命令CREATE TABLE AS。

示例：使用可变数量的参数

只要所有可选参数都具有相同的数据类型，就可以声明PL/pgSQL函数接受可变数量的参数。必须将函数的最后一个参数标记为VARIADIC，并使用数组类型声明参数。您可以将包含VARIADIC参数的函数称为可变参数函数。

例如，这个可变参数函数返回数值变量数组的最小值：

```
CREATE FUNCTION mleast (VARIADIC numeric[])
RETURNS numeric AS $$

DECLARE minval numeric;
BEGIN
    SELECT min($1[i]) FROM generate_subscripts( $1, 1) g(i)
INTO minval;
    RETURN minval;
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION

SELECT mleast(10, -1, 5, 4.4);
mleast
-----
      -1
(1 row)
```

实际上，VARIADIC位置或其以外的所有实际参数都被收集到一维数组中。

您可以将已构造的数组传递给可变参数函数。当您想要在可变参数函数之间传递数组时，这尤其有用。在函数调用中指定VARIADIC，如下所示：

```
SELECT mleast(VARIADIC ARRAY[10, -1, 5, 4.4]);
```

这可以防止PL/pgSQL将函数的可变参数扩展为其元素类型。

示例：使用默认参数值

您可以使用某些或所有输入参数的默认值声明PL/pgSQL函数。只要

调用的函数少于声明的参数个数，就会插入默认值。因为参数只能从实际参数列表的末尾省略，所以必须在使用默认值定义的参数之后为所有参数提供默认值。

示例：

```

CREATE FUNCTION use_default_args(a int, b int DEFAULT 2, c
int DEFAULT 3)
RETURNS int AS $$

DECLARE
    sum int;

BEGIN
    sum := $1 + $2 + $3;
    RETURN sum;
END;
$$ LANGUAGE plpgsql;

SELECT use_default_args(10, 20, 30);
use_default_args
-----
60
(1 row)

SELECT use_default_args(10, 20);
use_default_args
-----
33
(1 row)

SELECT use_default_args(10);
use_default_args
-----
15
(1 row)

```

您也可以使用=符号代替关键字DEFAULT。

示例：使用多态数据类型

PL/pgSQL支持多态*anyelement*，*anyarray*，*anyenum* 和*anynonnullarray* 类型。使用这些类型，您可以创建一个可在多种数据类型上运行的PL/pgSQL函数。有关Greenplum数据库中多态类型支持的其他信息，请参阅[Greenplum数据库数据类型](#)。

当PL/pgSQL函数的返回类型声明为多态类型时，会创建一个名为\$0的特殊参数。\$0的数据类型标识从实际输入类型推导出的函数的返回类型。

在此示例中，您将创建一个多态函数，该函数返回两个值的总和：

```
CREATE FUNCTION add_two_values(v1 anyelement,v2 anyelement)
RETURNS anyelement AS $$

DECLARE
    sum ALIAS FOR $0;

BEGIN
    sum := v1 + v2;
    RETURN sum;
END;
$$ LANGUAGE plpgsql;
```

执行add_two_values()，提供整数输入值：

```
SELECT add_two_values(1, 2);
add_two_values
-----
3
(1 row)
```

add_two_values()的返回类型是整数，即输入参数的类型。现在执行add_two_values()，提供浮点输入值：

```
SELECT add_two_values (1.1, 2.2);
add_two_values
-----
3.3
(1 row)
```

在这种情况下，add_two_values()的返回类型是float。

您还可以在多态函数中指定VARIADIC参数。

示例：匿名块

此示例使用DO命令将前一个示例中的t1_calc()函数中的语句作为匿名块执行。在该示例中，匿名块从临时表中检索输入值。

```
CREATE TEMP TABLE list AS VALUES ('test1') DISTRIBUTED RANDOMLY;

DO $$

DECLARE
    t1_row    table1%ROWTYPE;
    calc_int table1.f3%TYPE;
BEGIN
    SELECT * INTO t1_row FROM table1, list WHERE table1.f1
```

```
= list.column1 ;
    calc_int = (t1_row.f2 * t1_row.f3)::integer ;
    RAISE NOTICE 'calculated value is %', calc_int ;
END $$ LANGUAGE plpgsql ;
```

参考

关于PL/pgSQL的PostgreSQL文档是在<https://www.postgresql.org/docs/9.4/plpgsql.html>。

另请参阅*Greenplum*数据库参考指南中的CREATE FUNCTION命令。

有关内置*Greenplum*数据库函数的摘要，请参阅*Greenplum*数据库参考指南中的内置函数摘要。有关使用*Greenplum*数据库函数的信息，请参阅*Greenplum*数据库管理员指南中的“查询数据”。

有关移植Oracle函数的信息，请参阅<https://www.postgresql.org/docs/9.4/plpgsql-porting.html>。有关在*Greenplum*数据库中安装和使用Oracle兼容性函数的信息，请参阅*Greenplum*数据库实用程序指南中的“Oracle兼容性函数”。

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

本章包含以下信息：

- [关于Greenplum数据库PL/R](#)
- [启用PL/R语言支持](#)
- [示例](#)
- [下载和安装R包](#)
- [显示R库信息](#)
- [在启动时加载R模块](#)
- [参考](#)

Parent topic: [Greenplum数据库参考指南](#)

关于Greenplum数据库PL/R

PL/R是一种过程式语言。 使用Greenplum数据库PL/R扩展， 您可以使用R编程语言编写数据库函数，并使用包含R函数和数据集的R包。

有关支持的PL/R版本的信息，请参阅*Greenplum*数据库发行说明。

启用PL/R语言支持

对于需要使用它的每个数据库，使用SQL命令CREATE EXTENSION注册PL/R语言。 由于PL/R是不可信的语言，因此只有超级用户才能将PL/R注册到数据库。 例如，以gpadmin用户身份运行此命令，以使用名为testdb的数据库注册语言：

```
$ psql -d testdb -c 'CREATE EXTENSION plr;'
```

PL/R注册为不受信任的语言。

示例

以下是简单的PL/R示例。

示例1：将PL/R用于单行运算符

此函数使用R函数`rnorm()`生成具有正态分布的数字数组。



```

CREATE OR REPLACE FUNCTION r_norm(n integer, mean float8,
    std_dev float8) RETURNS float8[ ] AS
$$
    x<-rnorm(n,mean,std_dev)
    return(x)
$$
LANGUAGE 'plr';

```

以下CREATE TABLE命令使用r_norm函数填充表。 r_norm函数创建一个包含10个数字的数组。

```

CREATE TABLE test_norm_var
    AS SELECT id, r_norm(10,0,1) as x
    FROM (SELECT generate_series(1,30::: bigint) AS ID) foo
DISTRIBUTED BY (id);

```

示例2：以表格形式返回PL/R data.frames

假设您的PL/R函数返回R data.frame作为其输出，除非您想使用数组数组，否则需要做一些工作才能将PL/R中的data.frame看作一个简单的SQL表：

- 在Greenplum数据库中创建一个与R data.frame具有相同尺寸的TYPE：

```
CREATE TYPE t1 AS ...
```

- 定义PL/R函数时使用此TYPE

```
... RETURNS SET OF t1 AS ...
```

下一个示例给出了示例SQL。

示例3：使用PL/R的分层回归

下面的SQL定义了一个TYPE并使用PL/R运行层次回归：

```

--Create TYPE to store model results
DROP TYPE IF EXISTS wj_model_results CASCADE;
CREATE TYPE wj_model_results AS (
    cs text, coefext float, ci_95_lower float, ci_95_upper float,
    ci_90_lower float, ci_90_upper float, ci_80_lower float,
    ci_80_upper float);

--Create PL/R function to run model in R
DROP FUNCTION IF EXISTS wj_plr_RE(float [ ], text [ ]);
CREATE FUNCTION wj_plr_RE(response float [ ], cs text [ ])
RETURNS SETOF wj_model_results AS
$$
library(arm)
y<- log(response)
cs<- cs
d_temp<- data.frame(y,cs)

```

```

m0 <- lmer (y ~ 1 + (1 | cs), data=d_temp)
cs_unique<- sort(unique(cs))
n_cs_unique<- length(cs_unique)
temp_m0<- data.frame(matrix0,n_cs_unique, 7))
for (i in 1:n_cs_unique){temp_m0[i,]<-
  c(exp(coef(m0)$cs[i,1] + c(0,-1.96,1.96,-1.65,1.65,
  -1.28,1.28)*se.ranef(m0)$cs[i]))}
names(temp_m0)<- c("Coefest", "CI_95_Lower",
"CI_95_Upper", "CI_90_Lower", "CI_90_Upper",
"CI_80_Lower", "CI_80_Upper")
temp_m0_v2<- data.frames(cs_unique, temp_m0)
return(temp_m0_v2)

$$
LANGUAGE 'plr';

--Run modeling plr function and store model results in a
--table
DROP TABLE IF EXISTS wj_model_results_roi;
CREATE TABLE wj_model_results_roi AS SELECT *
  FROM wj_plr_RE((SELECT wj_droi2_array),
  (SELECT cs FROM wj_droi2_array));

```

下载和安装R包

R包是包含R函数和数据集的模块。 您可以安装R软件包以扩展Greenplum数据库中的R和PL/R功能。

Note: 如果扩容Greenplum数据库并添加segment主机，则必须在新主机的R安装中安装R软件包。 I

- 对于R包，标识所有相关R包和每个包Web URL。 通过从以下导航页面中选择给定的包，可以找到该信息：

https://cran.r-project.org/web/packages/available_packages_by_name.html

例如，R包arm的页面表明该包需要以下R库：Matrix, lattice, lme4, R2WinBUGS, coda, abind, foreign和MASS。

您还可以尝试使用R CMD INSTALL命令安装软件包以确定依赖软件包。

对于Greenplum数据库PL/R扩展中包含的R安装，所需的R软件包随PL/R扩展一起安装。但是，Matrix包需要更新的版本。

- 从命令行，使用wget实用程序将arm包的tar.gz文件下载到Greenplum数据库master主机：

```
 wget https://cran.r-project.org/src/contrib/Archive/arm/arm_1.5-03.tar.gz
```

```
 wget https://cran.r-project.org/src/contrib/Archive/Matrix/Matrix_0.9996875-1.tar.gz
```

- 使用gpscp实用程序和hosts_all文件将tar.gz文件复制到Greenplum数据库集群的所有节点上的同一目录。 hosts_all文件包含所有Greenplum数据库segment主机的列表。您可能需要root访问权限才能执行此操作。

```
 gpscp -f hosts_all Matrix_0.9996875-1.tar.gz =:/home/gpadmin
```

```
gpscp -f /hosts_all arm_1.5-03.tar.gz ::/home/gpadmin
```

4. 在交互模式下使用gpssh实用程序登录每个Greenplum数据库segment主机 (gpssh -f all_hosts)。 使用R CMD INSTALL命令从命令提示符安装软件包。 请注意，这可能需要root访问权限。 例如，此R install命令安装arm包的包。

```
$R_HOME/bin/R CMD INSTALL Matrix_0.9996875-1.tar.gz arm_1.5-03.tar.gz
```

5. 确保软件包安装在所有segment的\$R_HOME/library目录中 (gpssh可用于安装软件包)。 例如，此gpssh命令列出R库目录的内容。

```
gpssh -s -f all_hosts "ls $R_HOME/library"
```

gpssh选项-s在远程主机上运行命令之前获取greenplum_path.sh文件。

6. 测试是否可以加载R包。

如果可以加载R包，则此函数执行简单测试：

```
CREATE OR REPLACE FUNCTION R_test_require(fname text)
RETURNS boolean AS
$BODY$
    return(require(fname,character.only=T))
$BODY$
LANGUAGE 'plr';
```

此SQL命令检查是否可以加载R包arm：

```
SELECT R_test_require('arm');
```

显示R库信息

您可以使用R命令行显示有关Greenplum数据库主机上已安装的库和函数的信息。您还可以在R安装中添加和删除库。要在主机上启动R命令行，请以gadmin用户身份登录主机，并从\$GPHOME/ext/R-3.3.3/bin目录运行脚本R。

此R函数列出R命令行中的可用R包：

```
> library()
```

显示特定R包的文档

```
> library(help="package_name")
> help(package="package_name")
```

显示R函数的帮助文件：

```
> help("function_name")
> ?function_name
```

要查看已安装的软件包，请使用R命令`installed.packages()`。这将返回一个矩阵，其中包含已安装的每个包的行。下面，我们来看看这个矩阵的前5行。

```
> installed.packages()
```

必须先安装并加载任何未出现在已安装的软件包矩阵中的软件包，然后才能使用其函数。

可以使用`install.packages()`安装R包：

```
> install.packages("package_name")
> install.packages("mypkg", dependencies = TRUE, type="source")
```

从R命令行加载包。

```
> library(" package_name ")
```

可以使用`remove.packages`删除R包

```
> remove.packages("package_name")
```

您可以使用R命令`-e`选项从命令行运行函数。例如，此命令在R包MASS上显示帮助。

```
$ R -e 'help("MASS")'
```

在启动时加载R模块

PL/R可以在解释器初始化期间自动加载保存的R代码。要使用此功能，请创建`plr_modules`数据库表，然后将要自动加载的R模块插入表中。如果表存在，PL/R会将其包含的代码加载到解释器中。

在Greenplum数据库系统中，表行通常是分布式的，因此每行只存在于一个segment实例中。但是，每个segment实例的R解释器需要加载所有模块，因此正态分布的表将不起作用。必须将`plr_modules`表创建为默认架构中的复制表，以便表中的所有行都出现在每个segment实例中。例如：

```
CREATE TABLE public.plr_modules {
    modseq int4,
    modsrd text
} DISTRIBUTED REPLICATED;
```

有关使用PL/R自动加载功能的更多信息，请参阅
<https://www.joeconway.com/plr/doc/plr-module-funcs.html>。

参考

<https://www.r-project.org/> - R Project主页

<https://cran.r-project.org/web/packages/PivotalR/> - PivotalR的主页，提供了一个R接口，用于对Greenplum数据库表和视图进行操作，类似于R data.frame。PivotalR还支持直接从R使用机器学习包[MADlib](#)。

R文档随Greenplum R软件包一起安装：

`$GPHOME/ext/R-3.3.3/doc`

R函数和参数

- 参考<https://www.joeconway.com/doc/plr-funcs.html>

在R中传递数据值

- 参考<https://www.joeconway.com/doc/plr-data.html>

R中的聚合函数

- 参考<https://www.joeconway.com/doc/plr-aggregate-funcs.html>

展

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

gp_toolkit管理模式

gpperfmon 数据库

Greenplum 数据库数据类型

字符集支持

服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

本节概述了Greenplum数据库PL/Python语言。

- [关于Greenplum PL/Python](#)
- [启用和删除PL/Python支持](#)
- [用PL/Python开发函数](#)
- [安装Python模块](#)
- [示例](#)
- [参考](#)

Parent topic: [Greenplum数据库参考指南](#)

关于Greenplum PL/Python

PL/Python是一种可加载的过程语言。 使用Greenplum数据库PL/Python扩展，您可以在Python中编写Greenplum数据库用户定义函数，利用Python功能和模块快速构建强大的数据库应用程序。

您可以将PL/Python代码块作为匿名代码块运行。 请参阅Greenplum数据库参考指南中的[DO命令](#)。

Greenplum数据库PL/Python扩展默认安装在Greenplum数据库中。 Greenplum数据库安装了Python和PL/Python的一个版本。 这是Greenplum数据库使用的Python安装的位置：

```
$GPHOME/ext/python/
```

Greenplum数据库PL/Python限制

- Greenplum数据库不支持PL/Python触发器。
- PL/Python仅作为Greenplum数据库不可信语言提供。
- 不支持可更新游标（UPDATE ... WHERE CURRENT ... WHERE CURRENT OF）。

启用和删除PL/Python支持

PL/Python语言随Greenplum数据库一起安装。要在数据库中创建和运行PL/Python用户定义函数 (UDF)，必须在数据库中注册PL/Python语言。

启用PL/Python支持

对于需要使用它的每个数据库，使用SQL命令CREATE EXTENSION注册PL/Python语言。由于PL/Python是一种不受信任的语言，因此只有超级用户才能将PL/Python注册到数据库中。例如，当gpadmin用户使用名为testdb的数据库注册PL/Python时，运行此命令：

```
$ psql -d testdb -c 'CREATE EXTENSION plpythonu;'
```

PL/Python注册为不受信任的语言。

删除PL/Python支持

对于不再需要PL/Python语言的数据库，请使用SQL命令DROP EXTENSION删除对PL/Python的支持。由于PL/Python是一种不受信任的语言，因此只有超级用户才能从数据库中删除对PL/Python语言的支持。例如，以gpadmin用户身份运行此命令将从名为testdb的数据库中删除对PL/Python的支持：

```
$ psql -d testdb -c 'DROP EXTENSION plpythonu;'
```

如果任何现有对象（如函数）依赖于语言，则默认命令将失败。指定CASCADE选项也可以删除所有依赖对象，包括使用PL/Python创建的函数。

用PL/Python开发函数

PL/Python用户定义函数的主体是Python脚本。调用该函数时，其参数将作为数组args[]的元素传递。命名参数也作为普通变量传递给Python脚本。结果从带有return语句的PL/Python函数返回，或者在结果集语句的情况下从yield语句返回。

数组和列表

使用Python列表将SQL数组值传递到PL/Python函数中。类似地，PL/Python函数将SQL数组值作为Python列表返回。在典型的PL/Python使用模式中，您将使用`[]`指定数组。

以下示例创建一个返回整数数组的PL/Python函数：

```
CREATE FUNCTION return_py_int_array()
RETURNS int[]
AS $$

    return [1, 11, 21, 31]
$$ LANGUAGE plpythonu;

SELECT return_py_int_array();
return_py_int_array
-----
{1,11,21,31}
(1 row)
```

PL/Python将多维数组视为列表列表。使用嵌套的Python列表将多维数组传递给PL/Python函数。当PL/Python函数返回一个多维数组时，每个级别的内部列表必须都具有相同的大小。

以下示例创建一个PL/Python函数，该函数将多维数组的整数作为输入。该函数显示提供的参数的类型，并返回多维数组：

```
CREATE FUNCTION return_multidim_py_array(x int4[])
RETURNS int4[]
AS $$

    plpy.info(x, type(x))
    return x
$$ LANGUAGE plpythonu;

SELECT * FROM return_multidim_py_array(ARRAY[[1,2,3],
[4,5,6]]);
INFO: ([ [1, 2, 3], [4, 5, 6]], <type 'list' >)
CONTEXT: PL/Python function "return_multidim_py_type"
return_multidim_py_array
-----
{{1,2,3},{4,5,6}}
(1 row)
```

PL/Python还接受其他Python序列（如元组）作为函数参数，以便向后兼容不支持多维数组的Greenplum版本。在这种情况下，Python序列始终被视为一维数组，因为它们与复合类型不一致。

复合类型

使用Python映射将复合类型参数传递给PL/Python函数。 映射的元素名称是复合类型的属性名称。 如果属性具有空值，则其映射值为None。

您可以将复合类型结果作为序列类型（元组或列表）返回。 在多维数组中使用复合类型时，必须将复合类型指定为元组，而不是列表。 您不能将复合类型数组作为列表返回，因为确定列表是表示复合类型还是其他数组维度是不明确的。 在典型的使用模式中，您将使用()指定复合类型元组。

在以下示例中，您将创建一个复合类型和一个返回复合类型数◆◆

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

本节包括有关PL/Container 1.1及更高版本的以下信息：

- [关于PL/Container语言扩展](#)
- [关于PL/Container资源管理](#)
- [PL/Container Docker镜像](#)
- [#topic_qbl_dkq_vcb](#)
- [#topic_upg110](#)
- [卸载PL/Container](#)
- [使用PL/Container函数](#)
- [关于PL/Container运行PL/Python](#)
- [关于PL/Container运行PL/R](#)
- [配置PL/Container](#)
- [安装Docker](#)
- [参考](#)

Warning: PL/Container与Greenplum Database 5.2.0及更高版本兼容。尚未测试PL/Container与Greenplum Database 5.1.0或5.0.0的兼容性。

Parent topic: [Greenplum数据库参考指南](#)

关于PL/Container语言扩展

Greenplum数据库PL/Container语言扩展 (PL/Container) 是一个接口，允许Greenplum数据库与Docker容器交互以在容器中执行用户定义的函数 (UDF)。Docker容器确保用户代码无法访问源主机的文件系统。此外，容器在没有网络访问或网络访问受限的情况下启动，无法连接回Greenplum数据库或打开任何其他外部连接。有关可用UDF语言的信息，请参阅[PL/Container Docker镜像](#)。

一般来说，Docker容器是一个Linux进程，它使用Linux内核功能（如cgroup，命名空间和联合文件系统）以托管方式运行。Docker镜像是容器的基础。Docker容器是Docker镜像的运行实例。启动Docker容器时，指定Docker镜像。Docker镜像是在主机系统上运行Docker容器时使用的根文件系统更改和执行参数的集合。镜像没有状态，永远不会改变。有关Docker的信息，请访问Docker网站<https://www.docker.com/>。

Greenplum数据库仅在第一次调用该容器中的函数时启动容器。例如，考虑使用所有可用segment选择表数据的查询，并使用PL/Container函数对数据应用转换。在这种情况下，Greenplum数据库将在每个segment上仅启动Docker容器一次，然后联系正在运行的容器以获取结果。

在开始查询执行的完整循环之后，执行程序向容器发送调用。容器可能会响应容器执行的SPI-SQL查询，以从数据库中获取一些数据，并将结果返回给查询

执行器。

当与其连接关闭时，容器将关闭。当您关闭启动容器的Greenplum数据库会话时会发生这种情况。在待机模式下运行的容器几乎不消耗CPU资源，因为它在套接字上等待。PL/Container内存消耗取决于您在全局词典中缓存的数据量。

Warning: 当Greenplum数据库在Docker容器中运行时，不支持PL/Container。

PL/Container语言扩展可用作开源模块。有关该模块的信息，请参阅位于<https://github.com/greenplum-db/plcontainer> 的GitHub存储库中的README文件。

关于PL/Container资源管理

Greenplum数据库在Docker容器中运行PL/Container用户定义的函数。Docker容器和Greenplum数据库服务器共享相同主机上的CPU和内存资源。在默认情况下，Greenplum数据库不知道运行PL/Container实例所消耗的资源。

在PL/Container 1.2及更高版本中，您可以使用Greenplum数据库资源组来控制运行PL/Container实例的总体CPU和内存资源使用情况，如以下部分所述。

PL/Container管理两个级别的资源使用 - 容器级别和runtime级别。您可以使用为PL/Container runtime配置的memory_mb和cpu_share设置来控制容器级CPU和内存资源。memory_mb控制每个容器实例可用的内存资源。cpu_share设置标识容器与其他容器相比的CPU使用率的相对权重。有关PL/Container配置信息，请参阅[PL/Container配置文件](#)。

默认情况下，您不能限制执行PL/Container容器实例的数量，也不能限制它们使用的内存或CPU资源总量。

使用资源组管理PL/Container资源

在PL/Container 1.2.0及更高版本中，您可以使用Greenplum数据库资源组来管理和限制PL/Container runtime中容器的总CPU和内存资源。有关启用，配置和使用Greenplum数据库资源组的更多信息，请参阅Greenplum数据库管理员指南中的[使用资源组](#)。

Note: 如果未显式配置PL/Container runtime的资源组，则其容器实例仅受系统资源的限制。容器可能会以Greenplum数据库服务器为代价消耗资源。

外部组件（如PL/Container）的资源组使用Linux控制组（cgroup）来管理组件级内存和CPU资源的使用。使用资源组管理PL/Container资源时，可以配置Greenplum数据库应用于共享相同PL/Container runtime配置的所有容器实例的内存限制和CPU限制。

创建资源组以管理PL/Containerruntime的资源时，除了所需的CPU和内存限制外，还必须指定MEMORY_AUDITOR=cgroup和CONCURRENCY=0。例如，以下命令为PL/Container runtime创建名为plpy_run1_rg的资源组：

```
CREATE RESOURCE GROUP plpy_run1_rg WITH (MEMORY_AUDITOR=cgroup,
CONCURRENCY=0,
CPU_RATE_LIMIT=10, MEMORY_LIMIT=10);
```

PL/Container不使用MEMORY_SHARED_QUOTA和MEMORY_SPILL_RATIO资源组内存限制。有关此SQL命令的详细信息，请参阅[CREATE RESOURCE GROUP](#)参考页。

您可以创建一个或多个资源组来管理正在运行的PL/Container实例。为PL/Container创建资源组后，将资源组分配给一个或多个PL/Container runtime。您使用资源组的groupid进行此分配。您可以从gp_resgroup_config gp_toolkit视图确定给定资源组名称的groupid。例如，以下查询显示名为plpy_run1_rg的资源组的groupid：

```
SELECT groupname, groupid FROM gp_toolkit.gp_resgroup_config
WHERE groupname='plpy_run1_rg';

groupname | groupid
-----+-----
plpy_run1_rg | 16391
(1 row)
```

通过为plcontainer runtime-add (新runtime) 或plcontainer runtime-replace (现有runtime) 命令指定-s resource_group_id=rg_groupid选项，可以将资源组分配给PL/Container runtime配置。例如，要将plpy_run1_rg资源组分配给名为python_run1的新PL/Container runtime：

```
plcontainer runtime-add -r python_run1 -i
pivotaldata/plcontainer_python_shared:devel -l python -s
resource_group_id=16391
```

您还可以使用plcontainer runtime-edit命令将资源组分配给PL/Container runtime。有关plcontainer命令的信息，请参阅[plcontainer应用程序](#)。

将资源组分配给PL/Container runtime后，共享相同runtime配置的所有容器实例都受为组配置的内存和CPU的限制。如果减小PL/Container资源组的内存限制，则在组内运行容器中执行的查询可能会因内存不足而失败。如果在运行容器实例时删除PL/Container资源组，Greenplum数据库会终止正在运行的容器。

为PL/Container配置资源组

要使用Greenplum数据库资源组来管理PL/Container资源，必须显式配置资源组和PL/Container。

Note: PL/Container版本1.2利用Greenplum数据库版本5.8.0中引入的新资源组功能。如果您降级到使用PL/Container版本1.1或者更早的Greenplum数据库系统，您必须使用plcontainer runtime-edit从PL/Container runtime配置中删除

任何resource_group_id设置。

过程

执行以下过程以配置PL/Container以使用Greenplum数据库资源组进行CPU和内存资源管理：

1. 如果尚未在Greenplum数据库部署中配置和启用资源组，请按照*Greenplum数据库管理员指南*中的[使用资源组](#)中的说明配置cgroup并启用Greenplum数据库资源组。
Note: 如果先前已在部署中配置并启用了资源组，请确保Greenplum数据库资源组gpdb.conf cgroups配置文件包含内存{}块，如上一个链接中所述。
2. 分析Greenplum数据库部署的资源使用情况。确定要分配给PL/Container Docker容器的资源组CPU和内存资源的百分比。
3. 确定如何在PL/Container runtime中分配您在上一步中确定的总PL/Container CPU和内存资源。确认：
 - 您需要的PL/Container资源组的数量。
 - 分配给每个资源组的内存和CPU资源的百分比。
 - 资源组到PL/Container runtime分配。
4. 创建您在上面的步骤中标识的PL/Container资源组。例如，假设您选择将25% Greenplum数据库的内存和CPU资源分配给PL/Container。如果进一步将这些资源拆分为2个资源组60/40，则以下SQL命令将创建资源组：

```
CREATE RESOURCE GROUP plr_run1_rg WITH (MEMORY_AUDITOR=cgroup,
CONCURRENCY=0,
CPU_RATE_LIMIT=15, MEMORY_LIMIT=15);
CREATE RESOURCE GROUP plpy_run1_rg WITH (MEMORY_AUDITOR=cgroup,
CONCURRENCY=0,
CPU_RATE_LIMIT=10, MEMORY_LIMIT=10);
```

5. 查找并记下与您创建的每个资源组关联的groupid。例如：

```
SELECT groupname, groupid FROM gp_toolkit.gp_resgroup_config
WHERE groupname IN ('plpy_run1_rg', 'plr_run1_rg');

groupname | groupid
-----+-----
plpy_run1_rg | 16391
plr_run1_rg | 16393
(1 row)
```

6. 将您创建的每个资源组分配给所需的PL/Container runtime配置。如果尚未创建runtime配置，请使用plcontainer runtime-add命令。如果runtime已存在，请使用plcontainer runtime-replace或plcontainer runtime-edit命令将资源组分配添加到runtime配置。例如：

```
plcontainer runtime-add -r python_run1 -i
```

```
pivotaldata/plcontainer_python_shared:devel -l python -s
resource_group_id=16391
plcontainer runtime-replace -r r_run1 -i
pivotaldata/plcontainer_r_shared:devel -l r -s
resource_group_id=16393
```

有关plcontainer命令的信息，请参阅[plcontainer实用程序](#)。

PL/Container Docker镜像

PL/Python镜像和PL/R镜像可从Pivotal Network的Greenplum数据库产品下载站点获得，网址为<https://network.pivotal.io/>。

- 用于Python的PL/Container - 安装了Python 2.7.12的Docker镜像。还安装了Python数据科学模块。该模块包含一组与数据科学相关的python库。
- 用于R的PL/Container - 安装了R-3.3.3的容器的Docker镜像。还安装了R Data Science软件包。该软件包包含一组与数据科学相关的R库。

Docker镜像tag表示PL/Container版本（例如，1.0.0）。例如，用于Python Docker镜像的PL/Container的完整Docker镜像名称类似于pivotaldata/plc_python_shared:1.0.0。这是默认PL/Container配置中引用的名称。此外，您可以创建自定义Docker镜像，安装镜像并将图像添加到PL/Container配置。

前提条件

确保您的Greenplum数据库系统满足以下前提条件：

- Red Hat Enterprise Linux (RHEL) 7.x (或更高版本) 和CentOS 7.x (或更高版本) 上的Greenplum数据库5.2.x支持PL/Container。
Note: RHEL/CentOS 6.x系统不支持PL/Container，因为这些平台不官方支持Docker。
- 这些是Docker主机操作系统的前提条件。
RHEL或CentOS 7.x - 支持的最低Linux操作系统内核版本为3.10。RHEL 7.x和CentOS 7.x使用此内核版本。
您可以使用命令uname -r检查内核版本
Note: Red Hat提供，维护和支持的Docker版本仅在RHEL 7上可用。
Docker功能开发与RHEL7.x基础架构组件相关，用于内核，设备映射（精简配置，直接lvm），sVirt和systemd。
- Docker安装在Greenplum数据库主机（master, primary和所有standby主机）上
 - 对于RHEL或CentOS 7.x - Docker 17.05
请见[安装Docker](#)。
- 在每个Greenplum数据库主机上，gpadmin用户应该是docker组的一部分，

以便用户能够管理Docker镜像和容器。

安装PL/Container语言扩展

要使用PL/Container，请安装PL/Container语言扩展，安装Docker镜像，并配置PL/Container以使用镜像。

1. 确保Greenplum数据库主机满足前提条件，参考[前提条件](#)。
2. 从源代码构建和安装PL/Container扩展，参考[构建和安装PL/Container语言扩展](#)。
3. 安装Docker镜像并配置PL/Container，参考[安装PL/Container Docker镜像](#)。

构建和安装PL/Container语言扩展

PL/Container语言扩展可用作开源模块。有关构建和安装作为Greenplum数据库的一部分的模块信息，请参阅位于<https://github.com/greenplum-db/plcontainer> 的GitHub存储库中的README文件。

安装PL/Container Docker镜像

PL/Container语言扩展包括plcontainer实用程序，该实用程序在Greenplum数据库主机上安装Docker镜像，并将配置信息添加到PL/Container配置文件。配置信息允许PL/Container使用Docker镜像创建Docker容器。有关plcontainer的信息，请参阅[plcontainer实用程序](#)。

PL/Container开源模块包含dockerfiles，用于构建可与PL/Container一起使用的Docker镜像。您可以构建一个Docker镜像来运行PL/Python UDF和一个Docker镜像来运行PL/R UDF。请参阅GitHub存储库中的dockerfiles，网址为<https://github.com/greenplum-db/plcontainer>。

在Greenplum数据库主机上安装Docker镜像。此示例使用plcontainer实用程序为Python安装Docker镜像并更新PL/Container配置。该示例假定要安装的Docker镜像位于/home/gpadmin中的文件中。

此plcontainer命令从Docker镜像文件安装PL/Python的Docker镜像。

```
plcontainer image-add -f /home/gpadmin/plcontainer-python-images-1.0.0.tar.gz
```

该实用程序在Greenplum数据库主机上安装Docker镜像时显示进度信息。

使用plcontainer image-list命令在本地主机上显示已安装的Docker镜像。

此命令将信息添加到PL/Container配置文件，以便PL/Container可以访问Docker镜像以创建Docker容器。

```
plcontainer runtime-add -r plc_py -i
pivotaldata/plcontainer_python_shared:devel -l python
```

该实用程序在更新Greenplum数据库实例上的PL/Container配置文件时显示进度信息。

您可以使用`plcontainer runtime-show -r plc_py`命令查看PL/Container配置信息。您可以使用`plcontainer runtime-edit`命令查看PL/Container配置XML文件。

卸载PL/Container

要卸载PL/Container，请删除Docker容器和镜像，然后从Greenplum数据库中删除PL/Container支持。

删除对PL/Container的支持后，您在数据库中创建的`plcontainer`用户定义函数将不再起作用。

卸载Docker容器和镜像

在Greenplum数据库主机上，卸载不再需要的Docker容器和镜像。

`plcontainer image-list`命令列出了安装在本地Greenplum数据库主机上的Docker镜像。

`plcontainer image-delete`命令从所有Greenplum数据库主机中删除指定的Docker镜像。

如果容器不是由PL/Container管理，则某些Docker容器可能存在于主机上。您可能需要使用Docker命令删除容器。这些`docker`命令管理本地主机上的Docker容器和镜像。

- 命令`docker ps -a`列出主机上的所有容器。命令`docker stop`关闭容器。
- 命令`docker images`列出主机上的镜像。
- 命令`docker rmi`删除镜像。
- 命令`docker rm`删除容器。

删除数据库的PL/Container支持

对于不再需要PL/Container的数据库，请删除对PL/Container的支持。

PL/Container 1.1及更高版本

对于PL/Container 1.1及更高版本，请从数据库中删除扩展。此`psql`命令运行`DROP EXTENSION`命令以删除数据库`mytest`中的PL/Container。

```
psql -d mytest -c 'DROP EXTENSION plcontainer cascade;'
```

该命令将删除`plcontainer`扩展并从数据库中删除PL/Container特定的函数和视图。

PL/Container 1.0

以`gpadmin`用户身份运行`plcontainer_uninstall.sql`脚本。例如，此命令删除`mytest`数据库中的`plcontainer`语言。

```
psql -d mytest -f
$GPHOME/share/postgresql/plcontainer/plcontainer_uninstall.sql
```

该脚本使用`CASCADE`删除`plcontainer`语言，以从数据库中删除PL/Container特定的函数和视图。

使用PL/Container函数

在Greenplum数据库系统的数据库中启用PL/Container时，语言`plcontainer`将在数据库中注册。当您将`plcontainer`指定为UDF定义中的语言时，可以使用PL/Container Docker镜像支持的过程语言创建和运行用户定义的函数。

使用PL/Container的UDF定义必须包含这些项。

- UDF的第一行必须是`# container: ID`
- `LANGUAGE`属性必须是`plcontainer`

`ID`是PL/Container用于标识Docker镜像的名称。当Greenplum数据库在主机上执行UDF时，主机上的Docker镜像用于启动运行UDF的Docker容器。在XML配置文件`plcontainer_configuration.xml`中，有一个`runtime XML`元素，它包含指定Docker容器启动信息的相应`id XML`元素。有关PL/Container如何将`ID`映射到Docker镜像的信息，请参阅[配置PL/Container](#)。有关示例UDF定义，请参阅[示例](#)。

PL/Container配置文件仅在运行PL/Container函数的每个Greenplum数据库会话中第一次调用PL/Container函数时读取。您可以通过在会话期间对视图`plcontainer_refresh_config`执行`SELECT`命令来强制重新读取配置文件。例如，此`SELECT`命令强制读取配置文件。

```
SELECT * FROM plcontainer_refresh_config;
```

运行该命令会执行PL/Container函数，该函数会更新master实例和segment实例上的配置并返回刷新的状态。

```
gp_segment_id | plcontainer_refresh_local_config
-----+-----
 1 | ok
 0 | ok
 -1 | ok
(3 rows)
```

此外，您可以通过在视图plcontainer_show_config上执行SELECT命令来显示会话中的所有配置。例如，此SELECT命令返回PL/Container配置。

```
SELECT * FROM plcontainer_show_config;
```

运行该命令将执行PL/Container函数，该函数显示master实例和segment实例的配置信息。这是视图输出的开始和结束的示例。

```
INFO: plcontainer: Container 'plc_py_test' configuration
INFO: plcontainer:     image =
'pivotaldata/plcontainer_python_shared:devel'
INFO: plcontainer:     memory_mb = '1024'
INFO: plcontainer:     use container network = 'no'
INFO: plcontainer:     use container logging = 'no'
INFO: plcontainer:     shared directory from host
'/usr/local/greenplum-db/.bin/plcontainer_clients' to container
'/clientdir'
INFO: plcontainer:     access = readonly

...
INFO: plcontainer: Container 'plc_r_example' configuration (seg0
slice3 192.168.180.45:40000 pid=3304)
INFO: plcontainer:     image =
'pivotaldata/plcontainer_r_without_clients:0.2' (seg0 slice3
192.168.180.45:40000 pid=3304)
INFO: plcontainer:     memory_mb = '1024' (seg0 slice3
192.168.180.45:40000 pid=3304)
INFO: plcontainer:     use container network = 'no' (seg0 slice3
192.168.180.45:40000 pid=3304)
INFO: plcontainer:     use container logging = 'yes' (seg0
slice3 192.168.180.45:40000 pid=3304)
INFO: plcontainer:     shared directory from host
'/usr/local/greenplum-db/bin/plcontainer_clients' to container
'/clientdir' (seg0 slice3 192.168.180.45:40000 pid=3304)
INFO: plcontainer:     access = readonly (seg0 slice3
192.168.180.45:40000 pid=3304)
gp_segment_id | plcontainer_show_local_config
-----+-----
 0 | ok
 -1 | ok
 1 | ok
```

PL/Container函数plcontainer_containers_summary()显示有关当前运行的Docker容器的信息。

```
SELECT * FROM plcontainer_containers_summary();
```

如果普通（非超级用户）Greenplum数据库用户运行该函数，则该函数仅显示用户创建的容器的信息。如果Greenplum数据库超级用户运行该函数，则会显示Greenplum数据库用户创建的所有容器的信息。这是2个容器运行时的示例输出。

SEGMENT_ID	UP_TIME	OWNER	MEMORY_USAGE(KB)	CNTAINER_ID
1	Up 8 seconds	gpadmin	12940	693a6cb691f1d2881ec0160a44dae2547a0d5b799875d4ec106c09c97da422ea
1	Up 2 minutes	gpadmin	13628	bc9a0c04019c266f6d8269ffe35769d118bfb96ec634549b2b1bd2401ea20158
(2 rows)				

示例

示例的# container行中的

值plc_python_shared和plc_r_shared是plcontainer_config.xml文件中定义的id XML元素。id元素映射到指定要启动的Docker镜像的image元素。如果您使用不同的ID配置PL/Container，请更改# container行的值。有关配置PL/Container和查看配置设置的信息，请参阅[配置PL/Container](#)。

这是使用plc_python_shared容器运行的PL/Python函数的示例：

```
CREATE OR REPLACE FUNCTION pylog100() RETURNS double precision AS
$$
# container: plc_python_shared
import math
return math.log10(100)
$$ LANGUAGE plcontainer;
```

这是使用plc_r_shared容器的类似函数的示例：

```
CREATE OR REPLACE FUNCTION rlog100() RETURNS text AS $$
# container: plc_r_shared
return(log10(100))
$$ LANGUAGE plcontainer;
```

如果UDF中的# container行指定了不在PL/Container配置文件中的ID，则当您尝试执行UDF时，Greenplum数据库会返回错误。

关于PL/Container运行PL/Python

在Python语言容器中，实现了模块plpy。该模块包含以下方法：

- `plpy.execute(stmt)` - 执行查询字符串`stmt`并将查询结果返回到字典对象列表中。为了能够访问结果字段，请确保查询返回命名字段。
- `plpy.prepare(stmt[, argtypes])` - 准备查询的执行计划。如果在查询中有参数引用，则使用查询字符串和参数类型列表调用它。
- `plpy.execute(plan[, argtypes])` - 执行准备好的计划。
- `plpy.debug(msg)` - 将DEBUG2消息发送到Greenplum数据库日志。
- `plpy.log(msg)` - 将LOG消息发送到Greenplum数据库日志。
- `plpy.info(msg)` - 将INFO消息发送到Greenplum数据库日志。
- `plpy.notice(msg)` - 将NOTICE消息发送到Greenplum数据库日志。
- `plpy.warning(msg)` - 将WARNING消息发送到Greenplum数据库日志。在Greenplum数据库中引发的ERROR消息导致查询停止并回滚事务。
- `plpy.fatal(msg)` - 将FATAL消息发送到Greenplum数据库日志。FATAL消息导致关闭Greenplum数据库会话并回滚事务。
- `plpy.subtransaction()` - 在显式子事务中管理`plpy.execute`调用。有关`plpy.subtransaction()`的其他信息，请参阅PostgreSQL文档中的[显式子事务](#)。

如果在嵌套的Python函数调用中引发级别ERROR或FATAL的错误，则该消息包括封闭函数的列表。

Python语言容器支持这些在构造ad-hoc查询时很有用的字符串引用函数。

- `plpy.quote_literal(string)` - 返回引用的字符串，用作SQL语句字符串中的字符串文字。内嵌的单引号和反斜杠正常加倍。`quote_literal()`在null输入（空输入）上返回null。如果参数可能为null，则`quote_nullable()`可能更合适。
- `plpy.quote_nullable(string)` - 返回引用的字符串，用作SQL语句字符串中的字符串文字。如果参数为null，则返回NULL。内嵌的单引号和反斜杠正常加倍。
- `plpy.quote_ident(string)` - 返回引用的字符串，用作SQL语句字符串中的标识符。仅在必要时添加引号（例如，如果字符串包含非标识符字符或将进行大小写折叠）。内嵌的引号正常加倍。

从PL/Python函数返回文本时，PL/Container将Python unicode对象转换为数据库编码中的文本。如果无法执行转换，则返回错误。

PL/Container不支持此Greenplum数据库PL/Python特性：

- 多维数组。

此外，Python模块有两个全局字典对象，它们在函数调用之间保留数据。它们被命名为GD和SD。GD用于在同一容器内运行的所有函数之间共享数据，而SD用于在每个单独函数的多个调用之间共享数据。请注意，当容器进程位于segment或master上时，只能在同一会话中访问数据。请注意，对于空闲会话，Greenplum数据库会终止segment进程，这意味着相关容器将被关闭，GD和SD中的数据将丢失。

有关PL/Python的信息，参考[Greenplum PL/Python语言扩展](#)。

有关plpy方法的信息，参考<https://www.postgresql.org/docs/8.4/plpython-database.htm>。

关于PL/Container运行PL/R

在R语言容器中，实现了模块pg.spi。该模块包含以下方法：

- pg.spi.exec(stmt) - 执行查询字符串stmt并在R data.frame中返回查询结果。为了能够访问结果字段，请确保您的查询返回命名字段。
- pg.spi.prepare(stmt[, argtypes]) - 准备查询的执行计划。如果在查询中有参数引用，则使用查询字符串和参数类型列表调用它。
- pg.spi.execp(plan[, argtypes]) - 执行准备好的计划。
- pg.spi.debug(msg) - 将DEBUG2消息发送到Greenplum数据库日志。
- pg.spi.log(msg) - 将LOG消息发送到Greenplum数据库日志。
- pg.spi.info(msg) - 将INFO消息发送到Greenplum数据库日志。
- pg.spi.notice(msg) - 将NOTICE消息发送到Greenplum数据库日志。
- pg.spi.warning(msg) - 将WARNING消息发送到Greenplum数据库日志。
- pg.spi.error(msg) - 将ERROR消息发送到Greenplum数据库日志。在Greenplum数据库中引发的ERROR消息导致查询停止并回滚事务。
- pg.spi.fatal(msg) - 将FATAL消息发送到Greenplum数据库日志。FATAL消息导致关闭Greenplum数据库会话并回滚事务。

PL/Container不支持此PL/R特性：

- 多维数组。

关于PL/R的信息，参考[Greenplum PL/R 语言扩展](#)。

关于pg.spi方法的信息，参考<http://www.joeconway.com/plr/doc/plr-spisupport-funcs-normal.html>

配置PL/Container

Greenplum数据库实用程序plcontainer管理Greenplum数据库系统中的PL/Container配置文件。该实用程序可确保配置文件在Greenplum数据库master和segment实例之间保持一致。

Warning: 在不使用该实用程序的情况下修改segment实例上的配置文件，可能会在不同的Greenplum数据库segment上创建导致不可预期行为的不同的、不兼容的配置。

使用该实用程序进行的配置更改将应用于所有Greenplum数据库segment上

的XML文件。但是，当前运行的会话的PL/Container配置使用会话启动期间存在的配置。要在正在运行的会话中更新PL/Container配置，请在会话中执行此命令。

```
SELECT * FROM plcontainer_refresh_config;
```

运行该命令会执行PL/Container函数，该函数会更新master和segment实例上的会话配置。

plcontainer实用程序

plcontainer实用程序安装Docker镜像并管理PL/Container配置。该实用程序包含两组命令。

- `image-*`命令管理Greenplum数据库系统主机上的Docker镜像。
- `runtime-*`命令管理Greenplum数据库实例上的PL/Container配置文件。您可以将Docker镜像信息添加到PL/Container配置文件，包括镜像名称，位置和共享文件夹信息。您还可以编辑配置文件。

要将PL/Container配置为使用Docker镜像，请在所有Greenplum数据库主机上安装Docker镜像，然后将配置信息添加到PL/Container配置中。

PL/Container配置值（例如镜像名称，runtime ID，参数值和名称）区分大小写。

plcontainer语法

```
plcontainer [command] [-h | --help] [--verbose]
```

其中`command`是下面的其中一个。

```
image-add {{-f | --file} image_file} | {{-u | --URL} image_URL}
image-delete {-i | --image} image_name
image-list

runtime-add {-r | --runtime} runtime_id
{-i | --image} image_name {-l | --language} {python | r}
[{-v | --volume} shared_volume [{-v | --volume}
shared_volume...}]
[{-s | --setting} param=value [{-s | --setting} param=value
...]]
runtime-replace {-r | --runtime} runtime_id
{-i | --image} image_name -l {r | python}
[{-v | --volume} shared_volume [{-v | --volume}
shared_volume...}]
[{-s | --setting} param=value [{-s | --setting} param=value
...]]
runtime-show {-r | --runtime} runtime_id
runtime-delete {-r | --runtime} runtime_id
```

```
runtime-edit [{-e | --editor} editor]
runtime-backup {-f | --file} config_file
runtime-restore {-f | --file} config_file
runtime-verify
```

plcontainer命令和选项

image-add *location*

在Greenplum数据库主机上安装Docker镜像。指定主机上Docker镜像文件的位置或Docker镜像的URL。这些是受支持的位置选项。

- {-f | --file} *image_file*指定包含Docker镜像的主机上的tar存档文件。此示例指向gpadmin主目录中的镜像文件

```
/home/gpadmin/test_image.tar.gz
```

- {-u | --URL} *image_URL*指定Docker存储库和镜像的URL。此示例URL指向本地Docker存储库

```
192.168.0.1:5000/images/mytest_plc_r:devel
```

安装Docker镜像后，使用[runtime-add](#)命令配置PL/Container以使用Docker镜像。

image-delete { -i | --image} *image_name*

从所有Greenplum数据库主机中删除已安装的Docker镜像。例如，指定包括tag的完整Docker镜像名称

```
pivotaldata/plcontainer_python_shared:1.0.0
```

image-list

列出主机上安装的Docker镜像。该命令仅列出本地主机上的镜像，而不是远程主机。该命令列出了所有已安装的Docker镜像，包括使用Docker命令安装的镜像。

runtime-add *options*

将配置信息添加到所有Greenplum数据库主机上的PL/Container配置文件中。如果指定的*runtime_id*存在，则该实用程序将返回错误，并且不会添加配置信息。

关于PL/Container配置信息，参考[PL/Container配置文件](#)。

这些是支持的选项：

{-i | --image} *docker-image*

必须。指定Greenplum数据库主机上安装的完整Docker镜像名称，包括tag。例如 `pivotaldata/plcontainer_python:1.0.0`。

如果未安装指定的Docker镜像，该实用程序将返回警告。

`plcontainer image-list`命令显示已安装的镜像信息，包括名称和tag（存储库和tag列）。

{-l | --language} *python | r*

必须。指定PL/Container语言类型，支持的值是 `python` (PL/Python) 和 `r` (PL/R)。为 `runtime` 添加配置信息时，该实用程序会根据您指定的语言向配置添加启动命令。`Python` 语言的启动命令。

```
/clientdir/pyclient.sh
```

R语言的启动命令。

```
/clientdir/rclient.sh
```

{-r | --runtime} runtime_id

需要。添加runtime ID。在PL/Container配置文件中添加runtime元素时，这是PL/Container配置文件中id元素的值。最大长度为63字节。

在Greenplum数据库UDF# container行指定名称。参考[示例](#)。

{-s | --setting} param=value

可选的。指定要添加到runtime配置信息的设置。您可以多次指定此选项。该设置适用于runtime_id 指定的runtime配置。该参数是PL/Container配置文件中`settings`元素的XML属性。这些是有效的参数。

- `cpu_share` - 在runtime配置中为每个容器设置CPU限制。默认值为1024。该值是CPU使用率与其他容器的相对权重。
 - `memory_mb` - 在runtime配置中为每个容器设置内存限制。默认值为1024。该值是一个整数，指定以MB为单位的内存量。
 - `resource_group_id` - 将指定的资源组分配给runtime配置。资源组限制共享此runtime配置的所有容器的CPU和内存资源总使用量。您必须指定资源组的groupid。有关管理PL/Container资源的信息，请参阅[关于PL/Container资源管理](#)。
 - `roles` - 指定允许为runtime配置运行容器的Greenplum数据库角色。您可以指定单个角色名称或逗号分隔的角色名称列表。默认值没有限制。
 - `use_container_logging` - 启用或禁用容器的Docker日志记录。值为yes (启用日志记录) 或no (禁用日志记录，默认值)。
- Greenplum数据库服务器配置参数[log_min_messages](#)控制日志级别。默认日志级别为warning。有关PL/Container日志信息的信息，请参阅[备注](#)。

{-v | --volume} shared-volume

可选的。指定要绑定装载的Docker卷。您可以多次指定此选项以定义多个卷。

共享卷的格式：`host-dir:container-dir:[rw|ro]`。该信息作为属性存储在PL/Container配置文件中的runtime元素的shared_directory元素中。

- `host-dir` - 主机系统上目录的绝对路径。Greenplum数据库管理员用户 (gpadmin) 必须具有对目录的适当访问权限。
- `container-dir` - Docker容器中目录的绝对路径。
- `[rw|ro]` - 从容器访问主机目录的只读或读写权限。

为新runtime添加配置信息时，该实用程序会添加此只读共享卷信息。

```
greenplum-home/bin/plcontainer_clients:/clientdir:ro
```

如果需要，您可以指定其他共享目录。如果指定的`container-dir`与实

用程序添加的相同，或者如果指定具有相同*container-dir* 的多个共享卷，则该实用程序将返回错误。

Warning: 允许对主机目录进行读写访问需要特殊注意事项。

- 指定对主机目录的读写访问权限时，请确保指定的主机目录具有正确的权限。
- 运行PL/Container用户定义的函数时，主机上运行的多个并发Docker容器可能会更改主机目录中的数据。确保这些函数支持对主机目录中的数据进行多个并发访问。

`runtime-backup { -f | --file } config_file`

将PL/Container配置文件复制到本地主机上的指定文件。

`runtime-delete { -r | --runtime } runtime_id`

删除所有Greenplum数据库实例上PL/Container配置文件中的runtime配置信息。如果文件中不存在指定的*runtime_id*，则实用程序将返回一条消息。

`runtime-edit [{ -e | --editor } editor]`

使用指定的编辑器编辑XML文

件`plcontainer_configuration.xml`。默认编辑器是vi。

保存文件会更新所有Greenplum数据库主机上的配置文件。如果更新的文件中存在错误，该实用程序将返回错误并且不会更新该文件。

`runtime-replace options`

替换所有Greenplum数据库实例上的PL/Container配置文件中的runtime配置信息。如果*runtime_id* 不存在，则将信息添加到配置文件中。该实用程序将启动命令和共享目录添加到配置中。

有关命令选项和添加到配置的信息，请参阅[runtime-add](#)。

`runtime-restore { -f | --file } config_file`

使用本地主机上指定文件的信息替换所有Greenplum数据库实例上的PL/Container配置文件`plcontainer_configuration.xml`中的信息。

`runtime-show [{ -r | --runtime } runtime_id]`

显示格式化的PL/Container runtime配置信息。如果未指定*runtime_id*，则显示所有runtime标识的配置。

`runtime-verify`

使用master服务器上的配置信息检查Greenplum数据库实例上的PL/Container配置信息。如果该实用程序发现不一致，系统将提示您使用本地副本替换远程副本。该实用程序还执行XML验证。

`-h | --help`

显示帮助文本。如果在没有命令的情况下指定，则显示所有`plcontainer`命令的帮助。如果使用命令指定，则显示该命令的帮

助。
--verbose
为命令启用详细日志记录。

示例

这些是管理PL/Container的常用命令的示例：

- 在所有Greenplum数据库主机上安装Docker镜像。此示例从文件加载Docker镜像。该实用程序在命令行上显示进度信息，因为该实用程序在所有主机上安装Docker镜像。

```
plcontainer image-add -f plc_newr.tar.gz
```

安装Docker镜像后，您可以在PL/Container配置文件中添加或更新runtime条目，以便PL/Container访问Docker镜像以启动Docker容器。

- 将容器条目添加到PL/Container配置文件中。此示例添加PL/R runtime的配置信息，并指定内存和日志记录的共享卷和设置。

```
plcontainer runtime-add -r runtime2 -i test_image2:0.1 -l r \
-v /host_dir2/shared2:/container_dir2/shared2:ro \
-s memory_mb=512 -s use_container_logging=yes
```

该实用程序在命令行上显示进度信息，因为它将runtime配置添加到配置文件，并将更新的配置分发到所有实例。

- 在配置文件中显示给定runtime标识的特定runtime

```
plcontainer runtime-show -r plc_python_shared
```

该实用程序显示与此输出类似的配置信息。

```
PL/Container Runtime Configuration:
-----
Runtime ID: plc_python_shared
Linked Docker Image: test1:latest
Runtime Setting(s):
Shared Directory:
---- Shared Directory From HOST '/usr/local/greenplum-db/bin/plcontainer_clients' to Container '/clientdir', access mode is 'ro'
---- Shared Directory From HOST '/home/gpadmin/share/' to Container '/opt/share', access mode is 'rw'
```

- 在您选择的交互式编辑器中编辑配置。此示例使用vim编辑器编辑配置文件。

```
plcontainer runtime-edit -e vim
```

保存文件时，该实用程序会在将文件分发到Greenplum数据库主机时在命令行上显示进度信息。

- 将当前的PL/Container配置保存到文件中。此示例将文件保存到本地文件/home/gpadmin/saved_plc_config.xml

```
plcontainer runtime-backup -f /home/gpadmin/saved_plc_config.xml
```

- 使用XML文件覆盖PL/Container配置文件。此示例使用/home/gpadmin目录中文件中的信息替换配置文件中的信息。

```
plcontainer runtime-restore -f  
/home/gpadmin/new_plcontainer_configuration.xml
```

该实用程序在将更新的文件分发到Greenplum数据库实例时，在命令行上显示进度信息。

PL/Container配置文件

PL/Container在所有Greenplum数据库segment的数据目录中维护配置文件plcontainer_configuration.xml。PL/Container配置文件是XML文件。在XML文件中，根元素configuration包含一个或多个runtime元素。您可以在PL/Container函数定义的# container:行中指定runtime元素的id。

在XML文件中，名称（如元素和属性名称）和值区分大小写。

这是一个示例文件。

```
<?xml version="1.0" ?>
<configuration>
    <runtime>
        <id>plc_python_example1</id>

        <image>pivotaldata/plcontainer_python_with_clients:0.1</image>
            <command>./pyclient</command>
        </runtime>
        <runtime>
            <id>plc_python_example2</id>

            <image>pivotaldata/plcontainer_python_without_clients:0.1</image>
                <command>/clientdir/pyclient.sh</command>
                <shared_directory access="ro" container="/clientdir">
                    host="/usr/local/greenplum-db/bin/plcontainer_clients"
                <setting memory_mb="512"/>
                <setting use_container_logging="yes"/>
                <setting cpu_share="1024"/>
                <setting resource_group_id="16391"/>
            </runtime>
            <runtime>
                <id>plc_r_example</id>

                <image>pivotaldata/plcontainer_r_without_clients:0.2</image>
                    <command>/clientdir/rclient.sh</command>
                    <shared_directory access="ro" container="/clientdir">
                        host="/usr/local/greenplum-db/bin/plcontainer_clients"
                    <setting use_container_logging="yes"/>
```

```

<setting roles="gpadmin,user1"/>
</runtime>
<runtime>
</configuration>

```

这些是PL/Container配置文件中的XML元素和属性。

configuration

XML文件的根元素。

runtime

系统中可用的每个特定容器的一个元素。这些是configuration元素的子元素。

id

必须。该值用于从PL/Container用户定义的函数引用Docker容器。

`id`值在配置中必须是唯一的。`id`必须以字符或数字（a-z, A-Z或0-9）开头，并且可以包含字符，数字或字符_（下划线），.（期间），或-（破折号）。最大长度为63字节。

`id`指定当PL/Container创建Docker容器以执行用户定义的函数时要使用的Docker镜像。

image

必须。该值是完整的Docker镜像名称，包括镜像tag。与在Docker中启动此容器指定它们的方式相同。配置允许有许多容器对象引用相同的镜像名称，这种方式在Docker中它们将由相同的容器表示。

例如，您可能有两个`runtime`元素，具有不同的`id`元素，`plc_python_128`和`plc_python_256`，两者都引用Docker镜像`pivotaldata/plcontainer_python:1.0.0`。第一个`runtime`指定128MB RAM限制，第二个指定由`setting`元素的`memory_mb`属性指定的256MB限制。

command

必须。该值是要在容器内部运行以在容器内启动客户端进程的命令。在创建`runtime`元素时，`plcontainer`实用程序会根据语言（-l 选项）添加`command`元素。

python语言的`command`元素。

```
<command>/clientdir/pyclient.sh</command>
```

R语言的`command`元素。

```
<command>/clientdir/rclient.sh</command>
```

只有在构建自定义容器并希望在容器启动之前实现一些其他初始化逻辑时，才应修改该值。

Note: 无法使用`plcontainer`实用程序设置此元素。您可以使用`plcontainer runtime-edit`命令更新配置文件。

shared_directory

可选的。此元素为具有访问信息的容器指定共享Docker共享卷。 允许多个`shared_directory`元素。 每个`shared_directory`元素指定一个共享卷。 `shared_directory`元素的XML属性：

- `host` - 主机系统上的目录位置。
- `container` - 容器内的目录位置。
- `access` - 主机目录的访问级别，可以是`ro`（只读）或`rw`（读写）。

创建`runtime`元素时，`plcontainer`实用程序会添加`shared_directory`元素。

```
<shared_directory access="ro" container="/clientdir"
host="/usr/local/greenplum-db/bin/plcontainer_clients"/>
```

对于每个`runtime`元素，`shared_directory`元素的`container`属性必须是唯一的。 例如，`runtime`元素不能有两个带有属性`container="/clientdir"`的`shared_directory`元素。

Warning: 允许对主机目录进行读写访问需要特别考虑。

- 指定对主机目录的读写访问权限时，请确保指定的主机目录具有正确的权限。
- 运行PL/Container用户定义的函数时，主机上运行的多个并发Docker容器可能会更改主机目录中的数据。 确保这些函数支持对主机目录中的数据进行多个并发访问。

settings

可选的。此元素指定Docker容器配置信息。每个`setting`元素都包含一个属性。`element`属性指定日志记录，内存或网络信息。例如，此元素启用日志记录。

```
<setting use_container_logging="yes" />
```

这些是合法属性。

cpu_share

可选的。在`runtime`中指定每个PL/Container容器的CPU使用率。 元素的值是正整数。默认值为1024。该值是CPU使用率与其他容器的相对权重。

例如，与具有默认值1024的容器相比，`cpu_share`为2048的容器被分配了两倍的CPU切片时间。

memory_mb=" size "

可选的。该值指定允许每个容器使用的内存量（MB）。每个容器都以这个RAM量和两倍的交换空间开始。容器内存消耗受主机系统`cgroups`配置的限制，这意味着在内存过量使用的情况下，容器由系统终止。

resource_group_id=" rg_groupid "

可选的。该值指定要分配给PL/Container `runtime`的资源组的`groupid`。 资源组限制共享此`runtime`配置的所有正在运行的容器的CPU和内存资源总使用量。 您必须指定资源组

的groupid。如果未将资源组分配给PL/Container runtime配置，则其容器实例仅受系统资源的限制。有关管理PL/Container资源的信息，请参阅[关于PL/Container资源管理](#)。

`roles="list_of_roles"`

可选的。该值是Greenplum数据库角色名称或以逗号分隔的角色列表。PL/Container运行仅对列出的角色使用PL/Container runtime配置的容器。如果未指定该属性，则任何Greenplum数据库角色都可以运行此容器runtime配置的实例。例如，您创建一个指定`plcontainer`语言的UDF，并标识具有`roles`属性集的`# container: runtime`配置。当角色（用户）运行UDF时，PL/Container会检查角色列表，并仅在角色位于列表中时运行容器。

`use_container_logging="{yes | no}"`

可选的。启用或禁用容器的Docker日志记录。属性值`yes`启用日志记录。属性值`no`禁用日志记录（默认值）。

Greenplum数据库服务器配置参数[log_min_messages](#)控制PL/Container日志级别。默认日志级别为`warning`。有关PL/Container日志信息的信息，请参阅[备注](#)。

默认情况下，PL/Container日志信息将发送到系统服务。

在Red Hat 7或CentOS 7系统上，日志信息将发送

到`journald`服务。在Red Hat 6或CentOS 6系统上，日志将发送到`syslogd`服务。

更新PL/Container配置

您可以使用`plcontainer runtime-add`命令将`runtime`元素添加到PL/Container配置文件中。命令选项指定`runtime` ID，Docker镜像和语言等信息。您可以使用`plcontainer runtime-replace`命令更新现有`runtime`元素。该实用程序更新`master`和所有`segment`实例上的配置文件。

PL/Container配置文件可以包含多个`runtime`元素，这些元素引用由XML元素`image`指定的相同Docker镜像。在示例配置文件中，`runtime`元素包含名为`plc_python_128`和`plc_python_256`的`id`元素，两者都引用Docker容器`pivotaldata/plcontainer_python:1.0.0`。第一个`runtime`元素定义为128MB RAM限制，第二个定义为256MB RAM限制。

```
<configuration>
  <runtime>
    <id>plc_python_128</id>
    <image>pivotaldata/plcontainer_python:1.0.0</image>
    <command>./client</command>
    <shared_directory access="ro" container="/clientdir">
      host="/usr/local/gpdb/bin/plcontainer_clients"/>
      <setting memory_mb="128"/>
    </shared_directory>
  </runtime>
  <runtime>
    <id>plc_python_256</id>
    <image>pivotaldata/plcontainer_python:1.0.0</image>
```

```

<command>./client</command>
<shared_directory access="ro" container="/clientdir">
host="/usr/local/gpdb/bin/plcontainer_clients"/>
<setting memory_mb="256"/>
<setting resource_group_id="16391"/>
</runtime>
<configuration>

```

备注

- PL/Container不支持Greenplum数据库域对象。
- PL/Container在所有Greenplum数据库segment实例的数据目录中维护配置文件plcontainer_configuration.xml: master, standby master, primary和mirror。此查询列出了Greenplum数据库系统数据目录：

```
SELECT hostname, datadir FROM gp_segment_configuration;
```

PL/Container配置文件示例位

于\$GPHOME/share/postgresql/plcontainer中。

- 当Greenplum数据库执行PL/Container UDF时，Query Executer (QE) 进程启动Docker容器并根据需要重用它们。经过一定的空闲时间后，QE进程退出并销毁其Docker容器。您可以使用Greenplum数据库服务器配置参数`gp_vmem_idle_resource_timeout`控制空闲时间。控制空闲时间可能有助于Docker容器重用，并避免创建和启动Docker容器的开销。
Warning: 更改`gp_vmem_idle_resource_timeout`值可能会因资源问题而影响性能。该参数还控制释放除Docker容器之外的Greenplum数据库资源。
- 如果PL/Container Docker容器超过允许的最大内存量，则会终止该容器并显示内存不足警告。在配置了Docker 1.7.1版的Red Hat 6或CentOS 6系统上，如果PL/Container Docker容器主程序 (PID 1) 终止，也会显示内存不足警告。
- 在某些情况下，当PL/Container在高并发环境中运行时，Docker守护程序会挂起并指示内存不足的日志条目。即使系统似乎有足够的可用内存，也可能发生这种情况。
这个问题似乎是由两个因素共同触发的，即PL/Container使用的Go语言 (golang) 运行时的激进虚拟内存需求，以及overcommit_memory的Greenplum数据库Linux服务器内核参数设置。该参数设置为2，不允许内存过量使用。
可能有用的解决方法是增加交换空间量并增加Linux服务器内核参数overcommit_ratio。如果更改后问题仍然存在，则可能存在内存不足问题。您应检查系统上的可用内存，并在需要时添加更多RAM。您还可以降低群集负载。
- PL/Container不限制Docker基本设备大小，Docker容器的大小。在某些情况下，Docker守护程序控制基本设备大小。例如，如果Docker存储驱动程序是devicemapper，则Docker守护程序--storage-opt选项标志`dm.basesize`控制基本设备大小。devicemapper的默认基本设备大小为10GB。Docker命令`docker info`显示Docker系统信息，包括存储驱动程序。基本设备大小显示在Docker 1.12及更高版本中。有关Docker存储驱动程

序的信息，请参阅Docker信息[守护存储驱动程序](#)。

设置Docker基本设备大小时，必须在所有Greenplum数据库主机上设置大小。

- 启用PL/Container日志记录后，可以使用Greenplum数据库服务器配置参数[log_min_messages](#)设置日志级别。默认日志级别为warning。该参数控制PL/Container日志级别，还控制Greenplum数据库日志级别。
 - 使用setting属性use_container_logging为每个runtime ID启用或禁用PL/Container日志记录。默认为无记录。
 - PL/Container日志信息是来自Docker容器中运行的UDF的信息。默认情况下，PL/Container日志信息将发送到系统服务。在Red Hat 7或CentOS 7系统上，日志信息将发送到journald服务。在Red Hat 6或CentOS 6系统上，日志信息将发送到syslogd服务。PL/Container日志信息将发送到Docker容器运行的主机的日志文件中。
 - Greenplum数据库日志信息将发送到Greenplum数据库master上的日志文件。

在测试或排除PL/Container UDF故障时，可以使用SET命令更改Greenplum数据库日志级别。您可以在运行PL/Container UDF之前在会话中设置参数。此示例将日志级别设置为debug1。

```
SET log_min_messages='debug1' ;
```

Note: 参数log_min_messages控制Greenplum数据库和PL/Container日志记录，即使PL/Container UDF未运行，增加日志级别也可能影响Greenplum数据库性能。

安装Docker

要使用PL/Container，必须在所有Greenplum数据库主机系统上安装Docker。这些说明显示了如何在CentOS 7上设置Docker服务。在RHEL 7上安装是一个类似的过程。

在执行Docker安装之前，请确保满足这些要求。

- 可以访问CentOS extras存储库。
- 用户具有sudo权限或是root权限。

另请参阅CentOS的Docker站点安装说明<https://docs.docker.com/engine/installation/linux/centos/>。有关Docker命令的列表，请参阅Docker引擎运行参考<https://docs.docker.com/engine/reference/run/>。

在CentOS 7上安装Docker

这些步骤安装docker软件包并以具有sudo权限的用户身份启动docker服务。

1. 安装Docker所需的依赖项

```
sudo yum install -y yum-utils device-mapper-persistent-data lvm2
```

2. 添加Docker仓库

```
sudo yum-config-manager --add-repo  
https://download.docker.com/linux/centos/docker-ce.repo
```

3. 更新yum cache

```
sudo yum makecache fast
```

4. 安装Docker

```
sudo yum -y install docker-ce
```

5. 启动Docker守护进程。

```
sudo systemctl start docker
```

6. 要授予对Docker守护程序和docker命令的访问权限, 请将Greenplum数据库管理员 (gpadmin) 分配给组docker。

```
sudo usermod -aG docker gpadmin
```

7. 退出会话并再次登录以更新权限。

8. 运行Docker命令以测试Docker安装。此命令列出当前运行的Docker容器。

```
docker ps
```

此命令将Docker配置为在主机系统启动时启动。

```
sudo systemctl start docker.service
```

在所有Greenplum数据库主机上安装Docker后, 重新启动Greenplum数据库系统以允许Greenplum数据库访问Docker。

```
gpstop -ra
```

在CentOS 6上安装Docker

这些步骤安装Docker软件包并以具有sudo权限的用户身份启动docker服务。

1. 安装EPEL包

```
sudo yum -y install epel-release
```

2. 安装Docker

```
sudo yum -y install docker-io
```

3. 创建docker组

```
sudo groupadd docker
```

4. 启动Docker

```
sudo service docker start
```

5. 要授予对Docker守护程序和docker命令的访问权限，请将Greenplum数据库管理员（gpadmin）分配给组docker。

```
sudo usermod -aG docker gpadmin
```

6. 退出会话并再次登录以更新权限。

7. 运行Docker命令以测试Docker安装。此命令列出当前运行的Docker容器。

```
docker ps
```

此命令将Docker配置为在主机系统启动时启动。

```
sudo chkconfig docker on
```

在所有Greenplum数据库主机上安装Docker后，重新启动Greenplum数据库系统以允许Greenplum数据库访问Docker。

```
gpstop -ra
```

参考

Docker主页<https://www.docker.com/>

Docker命令行接口<https://docs.docker.com/engine/reference/commandline/cli/>

Dockerfile参考 <https://docs.docker.com/engine/reference/builder/>

在Linux系统上安装Docker<https://docs.docker.com/engine/installation/linux/centos/>

通过systemd控制和配置Docker<https://docs.docker.com/engine/admin/systemd/>



Greenplum数据库® 6.0文档

- 参考指南
- SQL Command Reference
- SQL 2008可选特性兼容性
- Greenplum环境变量
- 保留标识符和SQL关键字
- 系统目录参考
- gp_toolkit管理模式
- gpperfmon 数据库
- Greenplum 数据库数据类型
- 字符集支持
- 服务器配置参数

- 内置函数摘要
- Greenplum MapReduce规范
- Greenplum PL/pgSQL过程语言
- Greenplum PL/R 语言扩展
- Greenplum PL/Python语言扩展
- Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

- Greenplum的PL/Perl语言扩展
- 用于分析的Greenplum MADlib扩展
- 附加提供的模块

Greenplum Partner Connector API

本节包含Greenplum数据库的PL/Java语言的概述。

- [关于PL/Java](#)
- [关于Greenplum数据库PL/Java](#)
- [编写PL/Java函数](#)
- [使用JDBC](#)
- [异常处理](#)
- [保存点](#)
- [日志](#)
- [安全](#)
- [一些PL/Java问题和解决方案](#)
- [示例](#)
- [参考](#)

Parent topic: [Greenplum数据库参考指南](#)

关于PL/Java

使用Greenplum数据库的PL/Java扩展，用户可以使用自己喜欢的Java IDE编写Java方法，并将包含这些方法的JAR文件安装到Greenplum数据库中。

Greenplum数据库的PL/Java包基于开源PL/Java 1.5.0。Greenplum数据库的PL/Java提供以下功能。

- 能够使用Java 8或Java 11执行PL/Java函数。
- 能够指定Java运行时。
- 在数据库中安装和维护Java代码的标准化实用程序（在SQL 2003提案之后设计）
- 参数和结果的标准化映射。支持复杂类型和集合。
- 使用Greenplum数据库内部SPI例程的嵌入式高性能JDBC驱动程序。
- 元数据支持JDBC驱动程序。包括DatabaseMetaData和ResultSetMetaData。
- 能够从查询中返回ResultSet，作为逐行构建ResultSet的替代方法。
- 完全支持保存点和异常处理。
- 能够使用IN, INPUT和OUT参数。
- 两种独立的Greenplum数据库语言：
 - pljava, TRUSTED PL/Java language
 - pljavau, UNTRUSTED PL/Java language
- 当一个事务或者保存点提交或者回滚时，事务和保存点监听器能够被编译执行。
- 在所选平台上与GNU GCI集成。

SQL中的一个函数将在Java类中指定一个静态方法。为了使函数能够执行，所指定的类必须能够通过Greenplum数据库服务器上的pljava_classpath配置参数来指定类路径。PL/Java扩展添加了一组有助于安装和维护java类的函数。类存储在普通的Java档案——JAR文件中。JAR文件可以选择性地包含部署描述符，该描述符又包含在部署或取消部署JAR时要执行的SQL命令。这些函数是按照SQL 2003提出的标准进行设计的。

PL/Java实现了传递参数和返回值的标准化方法。使用标准JDBC ResultSet类传递复杂类型和集合。

PL/Java中包含JDBC驱动程序。此驱动程序调用Greenplum数据库内部SPI例程。驱动程序是必不可少的，因为函数通常将调用数据库以获取数据。当PL/Java函数提取数据时，它们必须使用与输入PL/Java执行上下文的主函数使用的相同的事务边界。

PL/Java针对性能进行了优化。Java虚拟机在与后端相同的进程中执行，以最小化调用开销。

PL/Java的设计目的是为了使数据库本身能够实现Java的强大功能，以便数据库密集型业务逻辑可以尽可能靠近实际数据执行。

当后端和Java VM之间的桥梁被调用时，将使用标准Java本机接口（JNI）。

关于Greenplum数据库PL/Java

在标准PostgreSQL和Greenplum数据库中实现PL/Java有一些关键的区别。

函数

以下函数在Greenplum数据库中不被支持。在分布式的Greenplum数据库环境中，classpath的处理方式与PostgreSQL环境下不同。

- `sqlj.install_jar`
- `sqlj.replace_jar`
- `sqlj.remove_jar`
- `sqlj.get_classpath`
- `sqlj.set_classpath`

Greenplum数据库使用`pljava_classpath`服务器配置参数代替`sqlj.set_classpath`函数。

服务器配置参数

以下服务器配置参数由PL/Java在Greenplum数据库中使用。这些参数取代了标准PostgreSQL PL/Java实现中使用的`pljava.*`参数：

- `pljava_classpath`
冒号(:)分隔的包含任何PL/Java函数中使用的Java类的jar文件列表。所有的jar文件必须安装在所有的Greenplum数据库主机的相同位置。使用可信的PL/Java语言处理程序，jar文件路径必须相对于\$GPHOME/lib/postgresql/java/目录。使用不受信任的语言处理程序（javaU语言标记），路径可以相对于\$GPHOME/lib/postgresql/java/或使用绝对路径。
服务器配置参数`pljava_classpath_insecure`控制服务器配置参数`pljava_classpath`是否可以由用户设置，无需Greenplum数据库超级用户权限。当启用`pljava_classpath_insecure`时，正在开发PL/Java函数的Greenplum数据库开发人员不必是数据库超级用户身份才能来更改`pljava_classpath`。
Warning: 启用`pljava_classpath_insecure`通过为非管理员数据库用户提供能够运行未经授权的Java方法暴露了安全风险。
- `pljava_statement_cache_size`
为准备语句设置最近使用（MRU）缓存的大小（KB）。
- `pljava_release_lingering_savepoints`
如果为TRUE，在函数退出后，长期持续的保存点将会释放。如果为FALSE，它们将被回滚。
- `pljava_vmoptions`
定义Greenplum数据库Java VM的启动选项。

参阅Greenplum数据库参考指南有关Greenplum数据库服务器配置参数的信息。

启用PL/Java并安装JAR文件

以Greenplum数据库管理员`gpadmin`的身份执行以下步骤。

1. 通过执行`CREATE EXTENSION`命令注册语言，以在数据库中启用PL/Java。例如，此命令在`testdb`数据库中启用PL/Java：

```
$ psql -d testdb -c 'CREATE EXTENSION pljava;'
```

Note: 不推荐使用PL/Java `install.sql`脚本（在以前的版本中用于注册该语言）。

2. 将Java归档（JAR文件）复制到所有Greenplum数据库主机上的同一目录。本示例使用Greenplum数据库`gpscp`程序将文件`myclasses.jar`复制到目录\$GPHOME/lib/postgresql/java/：

```
$ gpscp -f gphosts_file myclasses.jar
=:/usr/local/greenplum-db/lib/postgresql/java/
```

文件`gphosts_file`包含一个Greenplum数据库主机的列表。

3. 设置`pljava_classpath`服务器配置参数在master的`postgresql.conf`文件中。对于此示例，参数值是冒号(:)分隔的JAR文件列表。例如：

```
$ gpconfig -c pljava_classpath -v 'examples.jar:myclasses.jar'
```

当用户使用`gp pkg`实用程序安装PL/Java扩展包时，将安装`examples.jar`文件。

Note: 如果将JAR文件安装在除`$GPHOME/lib/postgresql/java/`之外的目录中，则必须指定JAR文件的绝对路径。所有的Greenplum数据库主机上的每个JAR文件必须位于相同的位置。有关指定JAR文件位置的更多信息，参阅*Greenplum*数据库参考指南中的`pljava_classpath`服务器配置参数。

4. 重新加载`postgresql.conf`文件。

```
$ gpstop -u
```

5. (可选)Greenplum提供了一个包含可用于测试的示例PL/Java函数的`examples.sql`文件。运行此文件中的命令来创建测试函数(它使用`examples.jar`中的Java类)。

```
$ psql -f $GPHOME/share/postgresql/pljava/examples.sql
```

编写PL/Java函数

有关使用PL/Java编写函数的信息。

- [SQL声明](#)
- [类型映射](#)
- [NULL处理](#)
- [复杂类型](#)
- [返回复杂类型](#)
- [返回复杂类型](#)
- [返回集的函数](#)
- [返回SETOF<标量类型>](#)
- [返回SETOF<复杂类型>](#)

SQL声明

一个Java函数被声明为该类的一个类名称和静态方法。该类将用于为该函数声明的schema定义的classpath进行解析。如果没有为该schema定义classpath，则使用public schema。如果没有找到classpath，则使用系统类加载器解析该类。

可以声明以下函数来访问`java.lang.System`类上的静态方法`getProperties`:

```
CREATE FUNCTION getsysprop(VARCHAR)
RETURNS VARCHAR
AS 'java.lang.System.getProperty'
LANGUAGE java;
```

运行以下命令返回`Java user.home`属性:

```
SELECT getsysprop('user.home');
```

类型映射

标量类型以简单的方式映射。此表列出了当前的映射。

Table 1. PL/Java数据类型映射

PostgreSQL	Java
bool	boolean
char	byte
int2	short
int4	int
int8	long
varchar	java.lang.String
text	java.lang.String
bytea	byte[]
date	java.sql.Date
time	java.sql.Time (stored value treated as local time)
timetz	java.sql.Time
timestamp	java.sql.Timestamp (stored value treated as local time)
timestamptz	java.sql.Timestamp
complex	java.sql.ResultSet
setof complex	java.sql.ResultSet

所有其他类型都映射到java.lang.String，并将使用为各自类型注册的标准textin/textout例程。

NULL处理

映射到java基元的标量类型不能作为NULL值传递。要传递NULL值，这些类型可以有一个替代映射。用户可以通过在方法引用中明确的指定该映射来启用映射。

```
CREATE FUNCTION trueIfEvenOrNull(integer)
RETURNS bool
AS 'foo.fee.Fum.trueIfEvenOrNull(java.lang.Integer)'
LANGUAGE java;
```

Java代码将类似于：

```
package foo.fee;
public class Fum
{
    static boolean trueIfEvenOrNull(Integer value)
    {
        return (value == null)
            ? true
            : (value.intValue() % 2) == 0;
    }
}
```

以下两个语句都产生true：

```
SELECT trueIfEvenOrNull(NULL);
SELECT trueIfEvenOrNull(4);
```

为了从Java方法返回NULL值，可以使用与原始对象相对应的对象类型（例如，返回java.lang.Integer而不是int）。PL/Java解析机制无论如何都会找到该方法。由于Java对于具有相同名称的方法不能具有不同的返回类型，因此不会引入任何歧义。

复杂类型

复杂类型将始终作为只读的java.sql.ResultSet传递，只有一行。ResultSet位于其行上，因此不应该调用next()。使用ResultSet的标准getter方法检索复杂类型的值。

例如：

```

CREATE TYPE complexTest
AS(base integer, incbase integer, ctime timestamptz);
CREATE FUNCTION useComplexTest(complexTest)
RETURNS VARCHAR
AS 'foo.fee.Fum.useComplexTest'
IMMUTABLE LANGUAGE java;

```

在java类Fum中，我们添加以下静态方法：

```

public static String useComplexTest(ResultSet complexTest)
throws SQLException
{
    int base = complexTest.getInt(1);
    int incbase = complexTest.getInt(2);
    Timestamp ctime = complexTest.getTimestamp(3);
    return "Base = \\" + base +
    "\\", incbase = \\" + incbase +
    "\\", ctime = \\" + ctime + "\"";
}

```

返回复杂类型

Java没有规定任何创建ResultSet的方法。因此，返回ResultSet不是一个选项。SQL-2003草案建议将复杂的返回值作为IN/OUT参数处理。PL/Java以这种方式实现了一个ResultSet。如果用户声明一个返回复杂类型的函数，则需要使用带有最后一个参数类型为java.sql.ResultSet的布尔返回类型的Java方法。该参数将被初始化为一个空的可更新结果集，它只包含一行。

假设已经创建了上一节中的complexTest类型。

```

CREATE FUNCTION createComplexTest(int, int)
RETURNS complexTest
AS 'foo.fee.Fum.createComplexTest'
IMMUTABLE LANGUAGE java;

```

PL/Java方法解析现在将在Fum类中找到以下方法：

```

public static boolean complexReturn(int base, int increment,
        ResultSet receiver)
throws SQLException
{
    receiver.updateInt(1, base);
    receiver.updateInt(2, base + increment);
    receiver.updateTimestamp(3, new
        Timestamp(System.currentTimeMillis()));
    return true;
}

```

返回值表示接收方是否应被视为有效的元组 (true) 或NULL (false)。

返回集的函数

返回结果集时，不要在返回结果集之前构建结果集，因为构建大型结果集将消耗大量资源。最好一次产生一行。顺便提一句，那就是Greenplum数据库后端期望一个使用SETOF返回的函数。那用户就可以返回SETOF的一个标量类型，如int, float或varchar或者可以返回一个复合类型的SETOF。

返回SETOF<标量类型>

为了返回一组标量类型，用户需要创建一个实现java.util.Iterator接口的Java方法。这是一个返回一个SETOF的varchar的方法的例子：

```

CREATE FUNCTION javatest.getSystemProperties()
RETURNS SETOF varchar
AS 'foo.fee.Bar.getNames'
IMMUTABLE LANGUAGE java;

```

这个简单的Java方法返回一个迭代器：

```
package foo.fee;
import java.util.Iterator;

public class Bar
{
    public static Iterator getNames()
    {
        ArrayList names = new ArrayList();
        names.add("Lisa");
        names.add("Bob");
        names.add("Bill");
        names.add("Sally");
        return names.iterator();
    }
}
```

返回SETOF<复杂类型>

返回SETOF<复杂类型>的方法必须使用接

口org.postgresql.pljava.ResultSetProvider或org.postgresql.pljava.ResultSetHandle。具有两个接口的原因是它们满足两种不同用例的最佳处理。前者适用于要动态创建要从SETOF函数返回的每一行的情况。在用户要返回执行查询的结果的情况下，后者将生成。

使用ResultSetProvider接口

该接口有两种方法。布尔型assignRowValues(java.sql.ResultSet tupleBuilder, int rowNumber)和void close()方法。Greenplum数据库的查询执行器将重复调用assignRowValues直到它返回假或者直到执行器决定不需要更多行为止。然后它会调用close。

用户可以通过以下方式使用此接口：

```
CREATE FUNCTION javatest.listComplexTests(int, int)
RETURNS SETOF complexTest
AS 'foo.fee.Fum.listComplexTest'
IMMUTABLE LANGUAGE java;
```

该函数映射到一个返回实现ResultSetProvider接口实例的静态java方法。

```
public class Fum implements ResultSetProvider
{
    private final int m_base;
    private final int m_increment;
    public Fum(int base, int increment)
    {
        m_base = base;
        m_increment = increment;
    }
    public boolean assignRowValues(ResultSet receiver, int currentRow)
    throws SQLException
    {
        // Stop when we reach 12 rows.
        //
        if(currentRow >= 12)
            return false;
        receiver.updateInt(1, m_base);
        receiver.updateInt(2, m_base + m_increment * currentRow);
        receiver.updateTimestamp(3, new
        Timestamp(System.currentTimeMillis()));
        return true;
    }
    public void close()
    {
        // Nothing needed in this example
    }
    public static ResultSetProvider listComplexTests(int base,
int increment)
    throws SQLException
    {
        return new Fum(base, increment);
    }
}
```

```

    }
}
```

`listComplextTests`方法被调用一次。如果没有可用结果或`ResultSetProvider`实例，将返回`NULL`。这里的Java类`Fum`实现了这个接口，所以它返回一个自己的实例。然后将重复调用`assignRowValues`方法，直到返回`false`。到那时候，将会调用`close`。

使用`ResultSetHandle`接口

该接口类似于`ResultSetProvider`接口因为它也有将在最后调用的`close()`方法。但是，不是让`evaluator`调用一次构建一行的方法，而是返回一个`ResultSet`的方法。查询`evaluator`将遍历该集合，并将`RestulSet`内容一次一个元组传递给调用者，直到对`next()`的调用返回`false`或者`evaluator`决定不需要更多行。

这是一个使用默认连接获取的语句执行查询的示例。适用于部署描述符的SQL看起来像这样：

```

CREATE FUNCTION javatest.listSupers()
RETURNS SETOF pg_user
AS 'org.postgresql.pljava.example.Users.listSupers'
LANGUAGE java;
CREATE FUNCTION javatest.listNonSupers()
RETURNS SETOF pg_user
AS 'org.postgresql.pljava.example.Users.listNonSupers'
LANGUAGE java;
```

并且在Java包`org.postgresql.pljava.example`中加入了一个`Users`类：

```

public class Users implements ResultSetHandle
{
    private final String m_filter;
    private Statement m_statement;
    public Users(String filter)
    {
        m_filter = filter;
    }
    public ResultSet getResultSet()
    throws SQLException
    {
        m_statement =
            DriverManager.getConnection("jdbc:default:connection").createStatement();
        return m_statement.executeQuery("SELECT * FROM pg_user
                                       WHERE " + m_filter);
    }

    public void close()
    throws SQLException
    {
        m_statement.close();
    }

    public static ResultSetHandle listSupers()
    {
        return new Users("usesuper = true");
    }

    public static ResultSetHandle listNonSupers()
    {
        return new Users("usesuper = false");
    }
}
```

使用JDBC

PL/Java包含映射到PostgreSQL SPI函数的JDBC驱动程序。可以使用以下语句获取映射到当前事务的连接：

```

Connection conn =
    DriverManager.getConnection("jdbc:default:connection");
```

获取连接后，可以准备和执行类似于其他JDBC连接的语句。这些是PL/Java JDBC驱动程序的限制：

- 事务无法以任何方式进行管理。因此，连接后用户不能用如下方法：
 - `commit()`
 - `rollback()`
 - `setAutoCommit()`
 - `setTransactionIsolation()`
- 在保存点上也有一些限制。保存点不能超过其设置的功能，并且必须由同一功能回滚或释放。
- 从`executeQuery()`返回的结果集始终为`FETCH_FORWARD`和`CONCUR_READ_ONLY`。
- 元数据仅在PL/Java 1.1或更高版本中可用。
- `CallableStatement`（用于存储过程）没有实现。
- `Clob`和`Blob`类型未完全实现，需要更多工作。`byte[]`和`String`可分别用于`bytea`和`text`。

异常处理

您可以像其他任何异常一样捕获并处理Greenplum数据库后端中的异常。后端的`ErrorData`结构作为一个名为`org.postgresql.pljava.ServerException`（从`java.sql.SQLException`中派生）的类中的属性公开，并且Java try/catch机制与后端机制同步。

Important: 在函数返回之前，用户将无法继续执行后端函数，并且在后端生成异常时传播错误，除非用户使用了保存点。当回滚保存点时，异常条件被重置，用户可以继续执行。

保存点

Greenplum数据库保存点使用`java.sql.Connection`接口公开。有两个限制。

- 必须在设置的函数中回滚或释放保存点。
- 保存点不能超过其设置的功能。

日志

PL/Java使用标准的Java Logger。因此，用户可以按如下写：

```
Logger.getAnonymousLogger().info( "Time is " + new
Date(System.currentTimeMillis()));
```

目前，记录器使用一个处理器来映射Greenplum数据库配置设置的当前状态`log_min_messages`到有效的Logger级别，并使用Greenplum数据库后端功能输出所有消息`elog()`。

Note: 第一次执行会话中的PL/Java函数时，将从数据库中读取`log_min_messages`设置。在Java端，在特定会话中执行第一个PL/Java函数之后，设置不会更改，直到重新启动使用PL/Java的Greenplum数据库会话。

Logger级别和Greenplum数据库后端级别之间适用以下映射。

Table 2. PL/Java日志级别

<code>java.util.logging.Level</code>	Greenplum数据库级别
SEVERE	ERROR
WARNING	WARNING
CONFIG	LOG
INFO	INFO
FINE	DEBUG1
FINEST	DEBUG2
FINER	DEBUG3

安全

- [安装](#)
- [可信语言](#)

安装

只有数据库超级用户才可以安装PL/Java。 使用SECURITY DEFINER安装PL/Java实用程序函数，以便它们执行以授予函数创建者的访问权限。

可信语言

PL/Java是一种可信语言。 可信的PL/Java语言无法访问PostgreSQL定义可信语言所规定的文件系统。 任何数据库用户都可以创建和访问受信任的语言的函数。

PL/Java还为语言javau安装语言处理程序。 此版本不受信任，只有超级用户可以创建使用它的新函数。 任何用户都可以调用这些函数。

一些PL/Java问题和解决方案

当编写PL/Java时，将JVM映射到与Greenplum数据库后端代码相同的进程空间中，对于多个线程，异常处理和内存管理，已经出现了一些问题。 这里是简要说明如何解决这些问题。

- [多线程](#)
- [异常处理](#)
- [Java垃圾收集器与palloc\(\)和堆栈分配](#)

多线程

Java本身就是多线程的。 Greenplum数据库后端不是。 没有什么可以阻止开发人员在Java代码中使用多个Threads类。 调用后端的终结器可能是从背景垃圾回收线程中产生的。 可能使用的几个第三方Java包使用多个线程。 该模式在同一过程中如何与Greenplum数据库后端共存？

解决方案

解决方案很简单。 PL/Java定义了一个特殊对象Backend.THREADLOCK。 当初始化PL/Java时，后端立即抓取该对象监视器（即它将在此对象上同步）。 当后端调用Java函数时，监视器将被释放，然后在调用返回时立即恢复。 来自Java的所有调用到后端代码都在同一个锁上同步。 这确保一次只能有一个线程可以从Java调用后端，并且只能在后端正在等待返回Java函数调用的时候调用。

异常处理

Java经常使用try/catch/finally块。 Greenplum数据库有时会使用一个异常机制来调用longjmp来将控件转移到已知状态。 这样的跳转通常会有效地绕过JVM。

解决方案

后端现在允许使用宏PG_TRY/PG_CATCH/PG_END_TRY捕获错误，并且在catch块中，可以使⽤ErrorData结构检查错误。 PL/Java实现了一个名为org.postgresql.pljava.ServerException的java.sql.SQLException子类。 可以从该异常中检索和检查ErrorData。 允许捕获处理程序发送回滚到保存点。 回滚成功后，执行可以继续。

Java垃圾收集器与palloc()和堆栈分配

原始类型始终按值传递。包括String类型(这是必需的，因为Java使用双字节字符)。复杂类型通常包含在Java对象中并通过引用传递。例如，Java对象可以包含指向palloc'ed或stack分配的内存的指针，并使用原生的JNI调用来提取和操作数据。一旦调用结束，这些数据将变得陈旧。进一步尝试访问这些数据最多只会产生非常不可预知的结果，但更有可能导致内存错误和崩溃。

解决方案

PL/Java包含的代码可以确保当MemoryContext或堆栈分配超出范围时，陈旧的指针被清除。Java包装器对象可能会生效，但是使用它们的任何尝试将导致陈旧的原生处理异常。

示例

以下简单的Java示例创建一个包含单个方法并运行该方法的JAR文件。

Note: 该示例需要Java SDK来编译Java文件。

以下方法返回一个子字符串。

```
{
public static String substring(String text, int beginIndex,
    int endIndex)
{
    return text.substring(beginIndex, endIndex);
}
}
```

在文本文件example.class中输入这些Java代码。

manifest.txt文件的内容：

```
Manifest-Version: 1.0
Main-Class: Example
Specification-Title: "Example"
Specification-Version: "1.0"
Created-By: 1.6.0_35-b10-428-11M3811
Build-Date: 01/20/2013 10:09 AM
```

编译java代码：

```
javac *.java
```

创建名为analytics.jar的JAR存档，其中包含JAR中的类文件和清单文件MANIFEST文件。

```
jar cfm analytics.jar manifest.txt *.class
```

将jar文件上传到Greenplum master主机。

运行gpscp实用程序将jar文件复制到Greenplum Java目录。使用-f选项指定包含master节点和segment节点主机列表的文件。

```
gpscp -f gphosts_file analytics.jar
:::/usr/local/greenplum-db/lib/postgresql/java/
```

使用gpconfig程序设置Greenplum pljava_classpath服务器配置参数。该参数列出已安装的jar文件。

```
gpconfig -c pljava_classpath -v 'analytics.jar'
```

运行gpstop实用程序-u选项重新加载配置文件。

```
gpstop -u
```

来自psql命令行，运行以下命令显示已安装的jar文件。

```
show pljava_classpath
```

以下SQL命令创建一个表并定义一个Java函数来测试jar文件中的方法：

```
create table temp (a varchar) distributed randomly;
insert into temp values ('my string');
--Example function
create or replace function java_substring(varchar, int, int)
returns varchar as 'Example.substring' language java;
--Example execution
select java_substring(a, 1, 5) from temp;
```

用户可以将内容放在一个文件mysample.sql中，并从psql命令行运行该命令：

```
> \i mysample.sql
```

输出类似于：

```
java_substring
-----
y st
(1 row)
```

参考

The PL/Java Github wiki page - <https://github.com/tada/pljava/wiki>.

PL/Java 1.5.0 release - https://github.com/tada/pljava/tree/REL1_5_STABLE.

展

Greenplum数据库® 6.0文档

- 参考指南
 - SQL Command Reference
 - SQL 2008可选特性兼容性
 - Greenplum环境变量
 - 保留标识符和SQL关键字
- 系统目录参考
 - gp_toolkit管理模式
 - gpperfmon 数据库
 - Greenplum 数据库数据类型
 - 字符集支持
 - 服务器配置参数
- 内置函数摘要
- Greenplum MapReduce规范
- Greenplum PL/pgSQL过程语言
- Greenplum PL/R 语言扩展
- Greenplum PL/Python语言扩展
- Greenplum PL/Container语言扩展
- Greenplum的PL/Java语言扩展

本章包含了如下信息：

- [关于Greenplum的PL/Perl](#)
- [Greenplum数据库中PL/Perl限制](#)
- [可信/不可信语言](#)
- [用PL/Perl开发函数](#)

Parent topic: [Greenplum数据库参考指南](#)

关于Greenplum的PL/Perl

通过Greenplum数据库的PL/Perl扩展，用户可以用Perl写用户定义的函数，可以利用Perl先进的字符串处理操作和函数。 PL/Perl同时提供了可信和不可信的语言变体。

PL/Perl内嵌在用户的Greenplum数据库发布中。 Greenplum数据库的PL/Perl要求系统中每台数据库主机都要事先装好Perl。

参考[PostgreSQL PL/Perl文档](#) □ 获取额外信息。

Greenplum数据库中PL/Perl限制

Greenplum数据库PL/Perl的限制包括：

- Greenplum数据库不支持PL/Perl触发器。
- PL/Perl函数不能直接相互调用。
- SPI还没有完全的实现。
- 如果用户使用`spi_exec_query()`获取非常大的数据集，用户应该要意识到它们都将进入内存中。 用户可以通过使用`spi_query()` / `spi_fetchrow()`来防止这个问题的发生。 一个相似的问题也可能发生，当集合返回函数通过一个`return`语句传递一个大量行的数据集到Greenplum数据库中。 对每一行使用`return_next`能够避免该问题的出现。
- 当一个会话正常结束而不是由于一个致命错误结束时，PL/Perl会执行用户定义好的任何END块。 当前没有其它的动作执行（文件处理

不会被自动地刷写并且对象也不会被自动销毁）。

可信/不可信语言

PL/Perl 包含了可信和不可信两种语言的变体。

PL/Perl可信语言被命名为plperl。 可信PL/Perl语言限制文件系统操作， 和require、use以及其它可能同操作系统或者数据库服务进程进行交互的语句。 有了这些限制，任何Greenplum用户能够创建和执行可信的plperl语言的函数。

PL/Perl不可信语言被命名为plperlu。 您不能使用plperlu不受信任的语言限制您创建的函数的操作。 只有数据库的超级用户才有权限创建有不可信PL/Perl语言的用户定义函数。 同时只有数据库的超级用户以及其他被显式授予特权的用户能够执行不可信PL/Perl的用户定义函数。

在解释器以及运行在单个进程中的多个解释器之间的通信方面，PL/Perl有限制。 请参考 PostgreSQL的[可信与不可信PL/Perl](#) PL/Perl文档获取更多的信息。

启用和移除PL/Perl支持

在用户能在数据库中创建以及执行一个PL/Perl的用户定义函数前，必须要在该数据库上注册PL/Perl语言。 要实现移除PL/Perl，用户必须显式的从每一个它注册的数据库中移除扩展。 您必须是数据库超级用户或所有者才能在Greenplum数据库中注册或删除受信任的语言。

Note: 只用数据库超级用户可能注册或者移除对不可信PL/Perl语言plperlu的支持。

在用户在数据库中开启或者移除PL/Perl支持之前，确保：

- 用户的Greenplum数据库正在运行。
- 用户已经执行（source）了greenplum_path.sh。
- 用户已经设置了\$MASTER_DATA_DIRECTORY和\$GPHOME环境变量。

开启PL/Perl支持

对于要在其中启用PL/Perl的每个数据库，请使用SQL [CREATE EXTENSION](#)

[EXTENSION](#)命令注册该语言。例如，以gpadmin用户身份运行以下命令，为名为testdb的数据库注册可信PL/Perl语言：

```
$ psql -d testdb -c 'CREATE EXTENSION plperl;'
```

移除PL/Perl

要从数据库中删除对PL/Perl的支持，请运行SQL [DROP EXTENSION](#)命令。例如，以gpadmin用户身份运行以下命令，从名为testdb的数据库中删除对可信PL/Perl语言的支持：

```
$ psql -d testdb -c 'DROP EXTENSION plperl;'
```

如果任何现有对象（如函数）依赖于语言，则默认命令将失败。指定CASCADE选项也可以删除所有依赖对象，包括使用PL/Perl创建的函数。

用PL/Perl开发函数

您可以使用标准SQL [CREATE FUNCTION](#)语法定义PL/Perl函数。PL/Perl用户定义函数的主体是普通的Perl代码。PL/Perl解释器将此代码包装在Perl子例程中。

您还可以使用PL/Perl创建匿名代码块。使用SQL [DO](#)命令调用的匿名代码块不接收任何参数，并且丢弃它可能返回的任何值。否则，PL/Perl匿名代码块的行为就像一个函数。只有数据库超级用户使用不受信任的plperlu语言创建匿名代码块。

[CREATE FUNCTION](#)命令的语法要求您将PL/Perl函数体写为字符串常量。虽然使用美元引用更方便，但您可以选择使用转义字符串语法（`E''`），前提是您将函数体中使用的任何单引号和反斜杠加倍。

PL/Perl参数和结果在Perl中处理。传递给PL/Perl函数的参数可通过@_数组访问。使用return语句返回结果值，或者作为函数中计算的最后一个表达式返回结果值。PL/Perl函数不能直接返回非标量类型，因为您在标量上下文中调用它。您可以通过返回引用来返回PL/Perl函数中的非标量类型（如数组，记录和集）。

PL/Perl将null参数值视为“未定义”。将STRICT关键字添加到LANGUAGE子句指示Greenplum数据库在任何输入参数为null时立即

返回null。当创建为STRICT时，函数本身不需要执行null检查。

以下PL/Perl函数使用STRICT关键字返回两个整数中的较大者，如果任何输入为null，则返回null：

```
CREATE FUNCTION perl_max (integer, integer) RETURNS integer
AS $$
    if ($_[0] > $_[1]) { return $_[0]; }
    return $_[1];
$$ LANGUAGE plperl STRICT;

SELECT perl_max( 1, 3 );
perl_max
-----
3
(1 row)

SELECT perl_max( 1, null );
perl_max
-----

```

(1 row)

PL/Perl认为函数参数中的任何内容都不是对字符串的引用，即标准的Greenplum数据库外部文本表示。提供给PL/Perl函数的参数值只是转换为文本形式的输入参数（就像它们已经被SELECT语句显示一样）。如果函数参数不是普通的数字或文本类型，则必须将Greenplum数据库类型转换为Perl更易使用的形式。相反，`return`和`return_next`语句接受任何字符串，该字符串是函数声明的返回类型的可接受输入格式。

有关其他信息，请参阅PostgreSQL [PL/Perl函数和参数](#) 文档，包括复合类型和结果集操作。

内置PL/Perl函数

PL/Perl包含用于访问数据库的内置函数，包括用于准备和执行查询以及操作查询结果的函数。该语言还包括用于错误记录和字符串操作的实用程序函数。

以下示例创建一个包含整数和文本列的简单表。它创建一个PL/Perl用户定义函数，该函数接受输入字符串参数并调用`spi_exec_query()`内置函数来选择表的所有列和行。该函数返回查询结果中的所有行，其中v列包含函数输入字符串。

```
CREATE TABLE test (
```

```

        i int,
        v varchar
    );
INSERT INTO test (i, v) VALUES (1, 'first line');
INSERT INTO test (i, v) VALUES (2, 'line2');
INSERT INTO test (i, v) VALUES (3, '3rd line');
INSERT INTO test (i, v) VALUES (4, 'different');

CREATE OR REPLACE FUNCTION return_match(varchar) RETURNS
SETOF test AS $$

    # store the input argument
    $ss = $_[0];

    # run the query
    my $rv = spi_exec_query('select i, v from test;');

    # retrieve the query status
    my $status = $rv->{status};

    # retrieve the number of rows returned in the query
    my $nrows = $rv->{processed};

    # loop through all rows, comparing column v value with
    input argument
    foreach my $rn (0 .. $nrows - 1) {
        my $row = $rv->{rows}[$rn];
        my $textstr = $row->{v};
        if( index($textstr, $ss) != -1 ) {
            # match!  return the row.
            return_next($row);
        }
    }
    return undef;
$$ LANGUAGE plperl EXECUTE ON MASTER ;

SELECT return_match( 'iff' );
return_match
-----
(4,different)
(1 row)

```

有关可用函数的详细讨论，请参阅PostgreSQL PL/Perl[内置函数文档](#)

。

PL/Perl中的全局值

您可以使用全局哈希映射`%_SHARED`在当前会话的生命周期内，在PL/Perl函数调用之间共享数据，包括代码引用。

以下示例使用`%_SHARED`在用户定义的`set_var()`和`get_var()`PL/Perl函数之间共享数据：

```

CREATE OR REPLACE FUNCTION set_var(name text, val text)
RETURNS text AS $$ 
    if ($_SHARED{$_[0]} = $_[1]) {
        return 'ok';
    } else {
        return "cannot set shared variable $_[0] to $_[1]";
    }
$$ LANGUAGE plperl;

CREATE OR REPLACE FUNCTION get_var(name text) RETURNS text
AS $$ 
    return $_SHARED{$_[0]};
$$ LANGUAGE plperl;

SELECT set_var('key1', 'value1');
set_var
-----
ok
(1 row)

SELECT get_var('key1');
get_var
-----
value1
(1 row)

```

出于安全原因，PL/Perl为每个角色创建一个单独的Perl解释器。这可以防止一个用户对另一个用户的PL/Perl函数的行为进行意外或恶意干扰。每个这样的解释器都保留其自己的%_SHARED变量值和其他全局状态。当且仅当它们由相同的SQL角色执行时，两个PL/Perl函数共享相同的%_SHARED值。

在某些情况下，您必须采取明确的步骤来确保PL/Perl函数可以在%_SHARED中共享数据。例如，如果应用程序在单个会话中在多个SQL角色（通过SECURITY DEFINER函数，使用SET ROLE等）下执行，请确保需要通信的函数由同一用户拥有，并将这些函数标记为SECURITY DEFINER。

备注

在开发PL/Perl函数其它要考虑的：

- PL/Perl内部使用UTF-8编码。它将其他编码中提供的任何参数转换为UTF-8，并将UTF-8的返回值转换回原始编码。
- 嵌套命名的PL/Perl子例程保留了与Perl相同的危险。
- 只有不受信任的PL/Perl语言变体支持模块导入。小心使用plperlu。
- 您在plperlu函数中使用的任何模块都必须在所有Greenplum数据

库主机上的相同位置可用。

MADlib扩展

Greenplum数据库® 6.0文档

本章节包含了以下信息：

- [关于 MADlib](#)
- [示例](#)
- [参考](#)

Parent topic: [Greenplum数据库参考指南](#)

关于 MADlib

MADlib是一个可扩展数据库分析的开源库。通过Greenplum的MADlib扩展，用户可以在Greenplum数据库中使用MADlib功能。

MADlib为结构化数据以及非结构化数据提供了数学、统计学以及机器学习方法的数据并行的实现。它提供了一整套基于SQL的机器学习、数据挖掘以及统计学算法，只需要运行在数据库引擎中，而不需要在Greenplum和其它工具之间进行数据的传递。

MADlib需要m4宏处理器版本1.4.13或更高版本。

MADlib可以与PivotalR一同使用，一个PivotalR包允许用户使用R客户端同Greenplum的数据进行交互。见[关于MADlib、R、PivotalR](#)。

示例

下面是使用GreenplumMADlib扩展的示例：

- [线性回归](#)
- [关联规则](#)
- [朴素贝叶斯分类](#)

见MADlib文档获取更多的示例。



□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

线性回归

该示例是在表regr_example执行一个线性回归。 因变量数据在y列中， 独立变量数据在x1和x2列中。

下面的语句创建regr_example表同时加载样本数据：

```
DROP TABLE IF EXISTS regr_example;
CREATE TABLE regr_example (
    id int,
    y int,
    x1 int,
    x2 int
);
INSERT INTO regr_example VALUES
(1, 5, 2, 3),
(2, 10, 7, 2),
(3, 6, 4, 1),
(4, 8, 3, 4);
```

MADlib的linregr_train()函数产生一个根据一个输入表包含的训练数据产生一个回归模型。 下面的SELECT语句在表regr_example执行一个简单的多元回归同时保存模型在表reg_example_model中。

```
SELECT madlib.linregr_train (
    'regr_example',           -- source table
    'regr_example_model',     -- output model table
    'y',                      -- dependent variable
    'ARRAY[1, x1, x2]'       -- independent variables
);
```

madlib.linregr_train()函数能够添加参数来设置分组的列以及计算模型的异方差性。

Note: 截距通过将一个独立变量设置为常数1来计算，如前一个例子所示。

在表regr_example上执行该查询并创建带有一行数据的regr_example_model表：

```
SELECT * FROM regr_example_model;
-[ RECORD 1 ]-----+
coef | {0.11111111111127,1.14814814814815,1.01851851851852}
r2 | 0.968612680477111
std_err | {1.49587911309236,0.207043331249903,0.346449758034495}
t_stats |
```

```

{0.0742781352708591, 5.54544858420156, 2.93987366103776}
p_values
{0.952799748147436, 0.113579771006374, 0.208730790695278}
condition_no | 22.650203241881
num_rows_processed | 4
num_missing_rows_skipped | 0
variance_covariance |
{{2.23765432098598, -0.257201646090342, -0.437242798353582},
{-0.257201646090342, 0.042866941015057, 0.0342935528120456},
{-0.437242798353582, 0.0342935528120457, 0.12002743484216}}

```

被保存到regr_example_model表中的模型能够同MADlib线性回归预测函数使用，`madlib.linregr_predict()`来查看残差：

```

SELECT regr_example.*,
       madlib.linregr_predict ( ARRAY[1, x1, x2], m.coef )
as predict,
       y - madlib.linregr_predict ( ARRAY[1, x1, x2],
m.coef ) as residual
FROM regr_example, regr_example_model m;
id | y   | x1  | x2  |      predict    |      residual
----+-----+-----+-----+-----+-----+
 1 | 5   | 2   | 3   | 5.46296296296297 | -0.462962962962971
 3 | 6   | 4   | 1   | 5.72222222222224 |  0.2777777777777762
 2 | 10  | 7   | 2   | 10.1851851851852 | -0.185185185185201
 4 | 8   | 3   | 4   | 7.62962962962964 |  0.370370370370364
(4 rows)

```

关联规则

这个例子说明了关联规则的数据挖掘技术在交易数据集。关联规则挖掘是发现大数据集中的变量之间关系的技术。这个例子将考虑那些在商店中通常一起购买的物品。除了购物篮分析，关联规则也应用在生物信息学中，网络分析，和其他领域。

这个例子分析利用MADlib函数`MADlib.assoc_rules`分析存储在表中的关于七个交易的购买信息。函数假定数据存储在两列中，每行有一个物品和交易ID。多个物品的交易，包括多个行，每行一个物品。

这些命令创建表。

```

DROP TABLE IF EXISTS test_data;
CREATE TABLE test_data (
    trans_id INT,
    product text
);

```

该INSERT命令向表中添加数据。

```
INSERT INTO test_data VALUES
(1, 'beer'),
(1, 'diapers'),
(1, 'chips'),
(2, 'beer'),
(2, 'diapers'),
(3, 'beer'),
(3, 'diapers'),
(4, 'beer'),
(4, 'chips'),
(5, 'beer'),
(6, 'beer'),
(6, 'diapers'),
(6, 'chips'),
(7, 'beer'),
(7, 'diapers');
```

MADlib函数`madlib.assoc_rules()`分析数据同时确定具有以下特征的关联规则。

- 一个值至少为.40的支持率。支持率表示包含X的交易与所有交易的比。
- 一个值至少为.75的置信率。置信率表示包含X的交易与包含Y的交易的比。可以将该度量看做给定Y下X的条件概率。

该SELECT命令确定关联规则，创建表`assoc_rules`同时添加统计信息到表中。

```
SELECT * FROM madlib.assoc_rules (
  .40,          -- support
  .75,          -- confidence
  'trans_id',   -- transaction column
  'product',    -- product purchased column
  'test_data',  -- table name
  'public',     -- schema name
  false);       -- display processing details
```

这是SELECT命令的输出。这有两条符合特征的规则。

output_schema	output_table	total_rules	total_time
public	assoc_rules	2	
00:00:01.153283			
(1 row)			

SELECT

为了查看关联规则，用户可以使用该命令。

```
SELECT pre, post, support FROM assoc_rules
ORDER BY support DESC;
```

这是输出。pre和post列分别是关联规则左右两边的项集。

pre	post	support
{diapers}	{beer}	0.714285714285714
{chips}	{beer}	0.428571428571429
(2 rows)		

基于数据，啤酒和尿布经常一起购买。为了增加销售额，用户可以考虑将啤酒和尿布放在一个架子上。

朴素贝叶斯分类

朴素贝叶斯分析，基于一个或多个独立变量或属性，预测一类变量或类结果的可能性。类变量是非数值类型变量，一个变量可以有一个数量有限的值或类别。类变量表示的整数，每个整数表示一个类别。例如，如果类别可以是一个“真”、“假”，或“未知”的值，那么变量可以表示为整数1, 2或3。

属性可以是数值类型、非数值类型以及类类型。训练函数有两个签名 - 一个用于所有属性为数值 另外一个用于混合数值和类类型的情况。后者的附加参数标识那些应该被当做数字值处理的属性。属性以数组的形式提交给训练函数。

MADlib朴素贝叶斯训练函数产生一个特征概率表和一个类的先验表，该表可以同预测函数使用为属性集提供一个类别的概率。

朴素贝叶斯示例 1 - 简单的所有都是数值属性

在第一个示例中，类变量取值为1或者2，同时这里有三个整型属性。

1. 下面的命令创建输入表以及加载样本数据。

```
DROP TABLE IF EXISTS class_example CASCADE;
CREATE TABLE class_example (
    id int, class int, attributes int[]);
INSERT INTO class_example VALUES
(1, 1, '{1, 2, 3}'),
(2, 1, '{1, 4, 3}'),
(3, 2, '{0, 2, 2}'),
(4, 1, '{1, 2, 1}'),
```

```
(5, 2, '{1, 2, 2}'),
(6, 2, '{0, 1, 3}');
```

在生产环境中的实际数据比该示例中的数据量更大，也能获得更好的结果。更大的训练数据集能够显著地提高分类的精确度。

2. 使用create_nb_prepared_data_tables()函数训练模型。

```
SELECT * FROM madlib.create_nb_prepared_data_tables (
    'class_example',           -- name of the training
    table
    'class',                  -- name of the class
    (dependent) column
    'attributes',             -- name of the attributes
    column
    3,                        -- the number of attributes
    'example_feature_probs',  -- name for the feature
    probabilities output table
    'example_priors'          -- name for the class priors
    output table
);
```

3. 为了使用模型进行分类，创建带有数据的表。

```
DROP TABLE IF EXISTS class_example_topredict;
CREATE TABLE class_example_topredict (
    id int, attributes int[]);
INSERT INTO class_example_topredict VALUES
    (1, '{1, 3, 2}'),
    (2, '{4, 2, 2}'),
    (3, '{2, 1, 1});
```

4. 用特征概率、类的先验和class_example_topredict表创建一个分类视图。

```
SELECT madlib.create_nb_probs_view (
    'example_feature_probs',      -- feature probabilities
    output table
    'example_priors',            -- class priors output
    table
    'class_example_topredict',   -- table with data to
    classify
    'id',                       -- name of the key column
    'attributes',                -- name of the attributes
    column
    3,                          -- number of attributes
    'example_classified'        -- name of the view to
    create
);
```

5. 显示分类结果。

```

SELECT * FROM example_classified;
key | class | nb_prob
----+-----+-----
 1 |     1 |      0.4
 1 |     2 |      0.6
 3 |     1 |      0.5
 3 |     2 |      0.5
 2 |     1 |      0.25
 2 |     2 |      0.75
(6 rows)

```

朴素贝叶斯示例 2 – 天气和户外运动

该示例计算在给定的天气条件下，用户要进行户外运动，例如高尔夫、网球等的概率。

表weather_example包含了样本值。

表的标识列是day，整型类型。

play列包含了因变量以及两个类别：

- 0 - No
- 1 - Yes

有四个属性：outlook、temperature、humidity、以及wind。他们是类变量。MADlib create_nb_classify_view()函数希望属性提供的是INTEGER、NUMERIC、或者FLOAT8值类型的数组，所以该示例的属性均用为整型进行编码：

- *outlook* 可能取值为 sunny (1), overcast (2), 或 rain (3).
- *temperature* 可能取值为 hot (1), mild (2), 或 cool (3).
- *humidity* 可能取值为 high (1) 或 normal (2).
- *wind* 可能取值为 strong (1) 或 weak (2).

下表显示了编码变量之前的训练数据。

day	play	outlook	temperature	humidity	wind
2	No	Sunny	Hot	High	Strong
4	Yes	Rain	Mild	High	Weak
6	No	Rain	Cool	Normal	Strong
8	No	Sunny	Mild	High	Weak
10	Yes	Rain	Mild	Normal	Weak
12	Yes	Overcast	Mild	High	Strong
14	No	Rain	Mild	High	Strong
1	No	Sunny	Hot	High	Weak
3	Yes	Overcast	Hot	High	Weak

5	Yes	Rain	Cool	Normal	Weak
7	Yes	Overcast	Cool	Normal	Strong
9	Yes	Sunny	Cool	Normal	Weak
11	Yes	Sunny	Mild	Normal	Strong
13	Yes	Overcast	Hot	Normal	Weak
(14 rows)					

1. 创建一个训练表。

```
DROP TABLE IF EXISTS weather_example;
CREATE TABLE weather_example (
    day int,
    play int,
    attrs int[]
);
INSERT INTO weather_example VALUES
( 2, 0, '{1,1,1,1}' ), -- sunny, hot, high, strong
( 4, 1, '{3,2,1,2}' ), -- rain, mild, high, weak
( 6, 0, '{3,3,2,1}' ), -- rain, cool, normal, strong
( 8, 0, '{1,2,1,2}' ), -- sunny, mild, high, weak
(10, 1, '{3,2,2,2}' ), -- rain, mild, normal, weak
(12, 1, '{2,2,1,1}' ), -- etc.
(14, 0, '{3,2,1,1}' ),
( 1, 0, '{1,1,1,2}' ),
( 3, 1, '{2,1,1,2}' ),
( 5, 1, '{3,3,2,2}' ),
( 7, 1, '{2,3,2,1}' ),
( 9, 1, '{1,3,2,2}' ),
(11, 1, '{1,2,2,1}' ),
(13, 1, '{2,1,2,2}' );
```

2. 根据训练表创建模型。

```
SELECT madlib.create_nb_prepared_data_tables (
    'weather_example', -- training source table
    'play',           -- dependent class column
    'attrs',          -- attributes column
    4,                -- number of attributes
    'weather_probs', -- feature probabilities output
    table
    'weather_priors' -- class priors
);
```

3. 查看特征概率：

class	attr	value	cnt	attr_cnt
1	3	2	6	2
1	1	2	4	3
0	1	1	3	3
0	1	3	2	3
0	3	1	4	2

1	4	1	3	2
1	2	3	3	3
1	2	1	2	3
0	2	2	2	3
0	4	2	2	2
0	3	2	1	2
0	1	2	0	3
1	1	1	2	3
1	1	3	3	3
1	3	1	3	2
0	4	1	3	2
0	2	3	1	3
0	2	1	2	3
1	2	2	4	3
1	4	2	6	2

(20 rows)

- 用模型分类一组记录，首先装载数据到一个表中。在该示例中，表t1有四个行将要分类。

```
DROP TABLE IF EXISTS t1;
CREATE TABLE t1 (
    id integer,
    attributes integer[]);
insert into t1 values
    (1, '{1, 2, 1, 1}'),
    (2, '{3, 3, 2, 1}'),
    (3, '{2, 1, 2, 2}'),
    (4, '{3, 1, 1, 2}');
```

- 使用MADlib create_nb_classify_view()函数对表中的行进行分类。

```
SELECT madlib.create_nb_classify_view (
    'weather_probs',          -- feature probabilities table
    'weather_priors',         -- classPriorsName
    't1',                     -- table containing values to
    classify
    'id',                      -- key column
    'attributes',              -- attributes column
    4,                         -- number of attributes
    't1_out'                   -- output table name
);
```

结果有四行，每行对应表t1中的一条记录。

```
SELECT * FROM t1_out ORDER BY key;
key | nb_classification
-----+
1  | {0}
2  | {1}
3  | {1}
4  | {0}
```

(4 rows)

参考

MADlib网站在<http://madlib.apache.org/>。

MADlib文档在<http://madlib.apache.org/documentation.html>。

PivotalR是第一类能够让用户使用R客户端对Greenplum驻留的数据和MADLib进行交互的R包。

关于MADlib、R、PivotalR

R语言是一门用于统计计算的开源编程语言。 PivotalR 是一个能够让用户通过R客户端与常驻Greenplum数据库的数据进行交互的R语言包。 使用PivotalR要求MADlib已经安装在了Greenplum数据库中。

PivotalR允许R用户不用离开R命令行就能利用数据库内分析的可扩展性和性能。 计算工作在数据库内执行，而终端用户受益于熟悉的R语言接口。 与相应的原生R函数相比，在可扩展性上得到提升的同时在执行时间上有降低。 此外， PivotalR消除了对于非常大的数据集需要花费几个小时完成的数据移动。

PivotalR包的关键特征：

- 以R语法的方式探索和操作数据库内的数据。 SQL翻译由PivotalR来执行。
- 使用熟悉的R语法的预测分析算法，例如线性和逻辑回归。
PivotalR访问MADlib数据库内分析函数调用。
- 对于广泛关于以标准R格式的示例文档包能够通过R客户端来访问。
- PivotalR包也支持MADlib函数的访问。

更多关于PivotalR的信息包括支持的MADlib功能的信息，见
<https://cwiki.apache.org/confluence/display/MADLIB/PivotalR>。

PivotalR的R语言包可以在<https://cran.r-project.org/web/packages/PivotalR/index.html>找到。

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

本节介绍了Greenplum数据库安装中可用的其他模块。这些模块可能来自PostgreSQL或Greenplum。

contrib模块通常打包为扩展。您可以使用`CREATE EXTENSION`命令在数据库中注册模块。您可以使用`DROP EXTENSION`从数据库中删除模块。

安装了以下Greenplum数据库和PostgreSQL contrib模块；有关使用说明，请参阅链接的模块文档。

- [citext](#) - 提供不区分大小写，可识别多字节的文本数据类型。
- [dblink](#) - 提供与其他Greenplum数据库的连接。
- [fuzzystrmatch](#) - 确定字符串之间的相似性和差异。
- [gp_sparse_vector](#) - 实现Greenplum数据库数据类型，该数据类型使用零压缩存储来帮助进行浮点计算。
- [hstore](#) - 提供一种数据类型，用于在单个PostgreSQL值中存储键/值对的集合。
- [orafce](#) - 提供Greenplum数据库特定的Oracle SQL兼容性函数。
- [pageinspect](#) - 低级检查数据库页面内容的函数；仅对超级用户可用。
- [pgcrypto](#) - 为Greenplum数据库提供加密函数。

Parent topic: [Greenplum数据库参考指南](#)



Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

Greenplum的PL/Perl语言扩展

用于分析的Greenplum MADlib扩展

□ 附加提供的模块

Greenplum Partner Connector API

使用Greenplum Partner Connector API (GPPC API) , 您可以使用C和C++编程语言编写可移植的Greenplum数据库用户定义函数 (UDF) 。 使用GPPC API开发的函数不需要重新编译或修改即可使用较旧或较新的Greenplum数据库版本。

可以使用Greenplum数据库中的SQL调用您写入GPPC API的函数。 API提供了一组函数和宏，可用于通过服务器编程接口 (SPI) 发出SQL命令，操作简单和复合数据类型函数参数和返回值，管理内存和处理数据。

您将使用GPPC API开发的C/C++函数编译到共享库中。 在Greenplum数据库群集中安装共享库并将GPPC函数注册为SQL UDF之后，Greenplum数据库用户可以使用GPPC函数。

Note: Greenplum Partner Connector支持Greenplum数据库版本4.3.5.0及更高版本。

本主题包含以下信息：

- [使用GPPC API](#)

- [要求](#)
- [头文件和库文件](#)
- [数据类型](#)
- [函数声明, 参数和结果](#)
- [内存处理](#)
- [使用可变长度文本类型](#)
- [错误报告和记录](#)
- [SPI函数](#)
- [关于元组描述符和元组](#)
- [Set-Returning函数](#)
- [表函数](#)
- [限制](#)
- [样例代码](#)

- [使用PGXS构建GPPC共享库](#)

- [使用Greenplum数据库注册GPPC函数](#)

- [打包和部署注意事项](#)

- [GPPC文本函数示例](#)

- [GPPC Set-Returning函数示例](#)

Parent topic: [Greenplum数据库参考指南](#)



使用GPPC API

GPPC API与PostgreSQL定义的C语言函数共享一些概念。有关开发C语言函数的详细信息，请参阅PostgreSQL文档中的[C语言函数](#)。

GPPC API是一个包装器，它使C/C++函数可以在Greenplum数据库中调用SQL。这个包装器通过API定义的函数和宏对表和数据操作以及SPI操作进行规范化来屏蔽您从Greenplum数据库更改中编写的GPPC函数。

GPPC API包括以下函数和宏：

- 对基础和复合数据类型进行操作。
- 处理函数参数和返回值。
- 分配和释放内存。
- 记录并向客户报告错误。
- 发出SPI查询。
- 返回一个表或一组行。
- 将表作为函数输入参数。

要求

使用GPPC API进行开发时：

- 您必须在具有与Greenplum数据库主机相同的硬件和软件体系结构的系统上开发代码。
- 您必须使用C或C++编程语言编写GPPC函数。
- 函数代码必须使用GPPC API，数据类型和宏。
- 函数代码不得使用PostgreSQL C语言函数API，头文件，函数或宏。
- 函数代码不能`#include postgres.h`头文件或使用`PG_MODULE_MAGIC`。
- 您必须仅使用GPPC包装的内存函数来分配和释放内存。请参阅[内存处理](#)。
- 目标文件中的符号名称不得相互冲突，也不得与Greenplum数据库服务器中定义的符号冲突。如果收到此类错误消息，则必须重命名函数或变量。

头文件和库文件

GPPC头文件和库文件安装在\$GPHOME中：

- \$GPHOME/include/gppc.h - 主要的GPPC头文件
- \$GPHOME/include/gppc_config.h - 定义GPPC版本的头文件
- \$GPHOME/lib/libgppc.[a, so, so.1, so.1.2] - GPPC存档和共享库

数据类型

您创建的GPPC函数将对驻留在Greenplum数据库中的数据进行操作。GPPC API包含等效Greenplum数据库SQL数据类型的数据类型定义。您必须在GPPC函数中使用这些类型。

GPPC API定义了可用于表示任何GPPC类型的通用数据类型。此数据类型名为`GppcDatum`，定义如下：

```
typedef int64_t GppcDatum;
```

下表标识了每个GPPC数据类型以及它映射到的SQL类型。

SQL类型	GPPC类型	GPPC类型的Oid
boolean	GppcBool	GppcOidBool
char (single byte)	GppcChar	GppcOidChar
int2/smallint	GppcInt2	GppcOidInt2
int4/integer	GppcInt4	GppcOidInt4
int8/bigint	GppcInt8	GppcOidInt8
float4/real	GppcFloat4	GppcOidFloat4
float8/double	GppcFloat8	GppcOidFloat8
text	*GppcText	GppcOidText
varchar	*GppcVarChar	GppcOidVarChar
char	*GppcBpChar	GppcOidBpChar
bytea	*GppcBytea	GppcOidBytea
numeric	*GppcNumeric	GppcOidNumeric
date	GppcDate	GppcOidDate
time	GppcTime	GppcOidTime
timetz	*GppcTimeTz	GppcOidTimeTz
timestamp	GppcTimestamp	GppcOidTimestamp
timestamptz	GppcTimestampTz	GppcOidTimestampTz
anytable	GppcAnyTable	GppcOidAnyTable

oid	GppcOid	
-----	---------	--

GPPC API专门处理文本，数字和时间戳数据类型，提供对这些类型进行操作的函数。

示例GPPC基本数据类型声明：

```
GppcText      message;
GppcInt4     arg1;
GppcNumeric   total_sales;
```

GPPC API定义了在通用GppcDatum类型和GPPC特定类型之间进行转换的函数。例如，要从整数转换为datum：

```
GppcInt4 num = 13;
GppcDatum num_dat = GppcInt4GetDatum(num);
```

符合类型

复合数据类型表示行或记录的结构，由字段名称列表及其数据类型组成。该结构信息通常被称为元组描述符。复合类型的实例通常称为元组或行。元组没有固定的布局，可以包含空字段。

GPPC API提供了一个界面，您可以使用该界面定义元组的结构，访问和设置元组。

当GPPC函数将表作为输入参数或返回表或记录类型集时，将使用此接口。本主题后面将介绍使用表中的元组和设置返回函数。

函数声明，参数和结果

GPPC API依赖于宏来声明函数并简化函数参数和结果的传递。这些宏包括：

任务	宏签名	描述
使函数SQL-可调用	GPPC_FUNCTION_INFO(<i>function_name</i>)	粘贴使函数 <i>function_name</i> SQL-可调用。
声明一个函数	GppcDatum <i>function_name</i> (GPPC_FUNCTION_ARGS)	声明名为 <i>function_name</i> 的GPPC函数；每个函数都必须具有相同的签名。
返回参数的数量	GPPC_NARGS()	返回传递给函数的参数数量。
获取参数	GPPC_GETARG_<ARGTYPE>(<i>arg_num</i>)	获取参数编号 <i>arg_num</i> 的值（从0开始），其中<ARGTYPE>标识参数的数据类型。例如，GPPC_GETARG_FLOAT8(0)。
获取并创建文本类型参数的副本	GPPC_GETARG_<ARGTYPE>_COPY(<i>arg_num</i>)	获取并复制参数号 <i>arg_num</i> 的值（从0开始）。<ARGTYPE>标识文本类型(text, varchar, bpchar, bytea)。例如，GPPC_GETARG_BYTEA_COPY(1)。
确定参数是否为NULL	GPPC_ARGISNULL(<i>arg_num</i>)	返回参数编号 <i>arg_num</i> 是否为NULL。
返回结果	GPPC_RETURN_<ARGTYPE>(<i>return_val</i>)	返回值 <i>return_val</i> ，其中<ARGTYPE>标识返回值的数据类型。例如，GPPC_RETURN_INT4(131)。

定义和实现GPPC函数时，必须使用上面标识的两个声明使用GPPC API声明它。例如，要声明名为add_int4s()的GPPC函数：

```
GPPC_FUNCTION_INFO(add_int4s);
GppcDatum add_int4s(GPPC_FUNCTION_ARGS);

GppcDatum
add_int4s(GPPC_FUNCTION_ARGS)
```

```
{
    // code here
}
```

如果add_int4s()函数接受两个int4类型的输入参数，则使用GPPC_GETARG_INT4(arg_num)宏来访问参数值。参数索引从0开始。例如：

```
GppcInt4 first_int = GPPC_GETARG_INT4(0);
GppcInt4 second_int = GPPC_GETARG_INT4(1);
```

如果add_int4s()返回两个输入参数的总和，则使用GPPC_RETURN_INT8(return_val)宏来返回此总和。例如：

```
GppcInt8 sum = first_int + second_int;
GPPC_RETURN_INT8(sum);
```

完整的GPPC函数：

```
GPPC_FUNCTION_INFO(add_int4s);
GppcDatum add_int4s(GPPC_FUNCTION_ARGS);

GppcDatum
add_int4s(GPPC_FUNCTION_ARGS)
{
    // get input arguments
    GppcInt4 first_int = GPPC_GETARG_INT4(0);
    GppcInt4 second_int = GPPC_GETARG_INT4(1);

    // add the arguments
    GppcInt8 sum = first_int + second_int;

    // return the sum
    GPPC_RETURN_INT8(sum);
}
```

内存处理

GPPC API提供用于分配和释放内存的函数，包括文本内存。必须将这些函数用于所有内存操作。

函数名	描述
void *GppcAlloc(size_t num)	分配num个字节的未初始化内存。
void *GppcAlloc0(size_t num)	分配num个字节的初始化为0内存。
void *GppcRealloc(void * ptr, size_t num)	调整预分配的内存大小。
void GppcFree(void * ptr)	释放分配的内存。

分配内存后，可以使用memcpy()等系统函数来设置数据。

以下示例分配GppcDatum数组并将数组设置为函数输入参数的datum版本：

```
GppcDatum *values;
int attnum = GPPC_NARGS();

// allocate memory for attnum values
values = GppcAlloc( sizeof(GppcDatum) * attnum );

// set the values
for( int i=0; i<attnum; i++ ) {
    GppcDatum d = GPPC_GETARG_DATUM(i);
    values[i] = d;
}
```

为GPPC函数分配内存时，可以在当前上下文中分配它。GPPC API包括返回，创建，切换和重置内存上下文的函数。

函数名	描述
GppcMemoryContext	返回当前内存上下文。
GppcGetCurrentMemoryContext(void)	

GppcMemoryContext GppcMemoryContextCreate(GppcMemoryContext <i>parent</i>)	在 <i>parent</i> 下创建新的内存上下文。
GppcMemoryContext GppcMemoryContextSwitchTo(GppcMemoryContext <i>context</i>)	切换到内存上下文 <i>context</i> 。
void GppcMemoryContextReset(GppcMemoryContext <i>context</i>)	在内存上下文 <i>context</i> 中重置（释放）内存。

Greenplum数据库通常在每元组上下文中调用一个SQL调用的函数，它在每次服务器后端处理表行时创建和删除。不要假设在当前内存上下文中分配的内存多个函数调用中可用。

使用可变长度文本类型

GPPC API支持可变长度文本，`varchar`，空白填充和字节数组类型。在对这些数据类型进行操作时，必须使用GPPC API提供的函数。GPPC API中提供的可变文本操作函数包括为其分配内存，确定字符串长度，获取字符串指针以及访问这些类型的函数：

函数名	描述
GppcText GppcAllocText(size_t <i>len</i>) GppcVarChar GppcAllocVarChar(size_t <i>len</i>) GppcBpChar GppcAllocBpChar(size_t <i>len</i>) GppcBytea GppcAllocBytea(size_t <i>len</i>)	为不同长度类型分配 <i>len</i> 个字节的内存。
size_t GppcGetTextLength(GppcText <i>s</i>) size_t GppcGetVarCharLength(GppcVarChar <i>s</i>) size_t GppcGetBpCharLength(GppcBpChar <i>s</i>) size_t GppcGetByteaLength(GppcBytea <i>b</i>)	返回内存块中的字节数。
char *GppcGetTextPointer(GppcText <i>s</i>) char *GppcGetVarCharPointer(GppcVarChar <i>s</i>) char *GppcGetBpCharPointer(GppcBpChar <i>s</i>) char *GppcGetByteaPointer(GppcBytea <i>b</i>)	返回一个指向内存块头部的字符串指针。该字符串不以空值终止。
char *GppcTextGetString(GppcText <i>s</i>) char *GppcVarCharGetString(GppcVarChar <i>s</i>) char *GppcBpCharGetString(GppcBpChar <i>s</i>)	返回一个指向内存块头部的字符串指针。该字符串以空值终止。
GppcText *GppcCStringGetText(const char * <i>s</i>) GppcVarChar *GppcCStringGetVarChar(const char * <i>s</i>) GppcBpChar *GppcCStringGetBpChar(const char * <i>s</i>)	从字符串构建变长类型。

`GppcGet<VLEN_ARGLTYPE>Pointer()`函数返回的内存可能指向实际的数据库内容。请勿修改内存内容。GPPC API提供了在需要时为这些类型分配内存的函数。分配内存后，可以使用`memcpy()`等系统函数来设置数据。

以下示例处理文本输入参数，并为文本字符串连接操作分配和设置结果内存：

```
GppcText first_textstr = GPPC_GETARG_TEXT(0);
GppcText second_textstr = GPPC_GETARG_TEXT(1);
```

```

// determine the size of the concatenated string and allocate
// text memory of this size
size_t arg0_len = GppcGetTextLength(first_textstr);
size_t arg1_len = GppcGetTextLength(second_textstr);
GppcText retstring = GppcAllocText(arg0_len + arg1_len);

// construct the concatenated return string; copying each string
// individually
memcpy(GppcGetTextPointer(retstring), GppcGetTextPointer(first_textstr),
arg0_len);
memcpy(GppcGetTextPointer(retstring) + arg0_len,
GppcGetTextPointer(second_textstr), arg1_len);

```

错误报告和记录

GPPC API提供错误报告和日志记录函数。 API定义的报告级别与Greenplum数据库中的级别相同：

```

typedef enum GppcReportLevel
{
    GPPC_DEBUG1                      = 10,
    GPPC_DEBUG2                      = 11,
    GPPC_DEBUG3                      = 12,
    GPPC_DEBUG4                      = 13,
    GPPC_DEBUG                         = 14,
    GPPC_LOG                          = 15,
    GPPC_INFO                         = 17,
    GPPC_NOTICE                       = 18,
    GPPC_WARNING                      = 19,
    GPPC_ERROR                        = 20,
} GppcReportLevel;

```

(Greenplum数据库[client_min_messages](#)服务器配置参数控制当前客户端日志记录级别。
[log_min_messages](#)配置参数控制当前日志到日志文件级别。)

GPPC报告包括报告级别，报告消息和可选的报告回调函数。

GPPC API提供的报告和处理函数包括：

函数名	描述
GppcReport()	格式化并打印/记录指定报告级别的字符串。
GppcInstallReportCallback()	注册/安装报告回调函数。
GppcUninstallReportCallback()	卸载报告回调函数。
GppcGetReportLevel()	从错误报告中检索级别。
GppcGetReportMessage()	从错误报告中检索消息。
GppcCheckForInterrupts()	如果中断挂起则出错。

GppcReport()函数签名是：

```
void GppcReport(GppcReportLevel elevel, const char *fmt, ...);
```

GppcReport()采用类似于printf()的格式化字符串输入参数。以下示例生成格式化GPPC文本参数的错误级别报告消息：

```
GppcText uname = GPPC_GETARG_TEXT(1);
GppcReport(GPPC_ERROR, "Unknown user name: %s", GppcTextGetCString(uname));
```

有关示例报告回调处理程序，请参阅[GPPC示例代码](#)。

SPI函数

Greenplum数据库服务器编程接口 (SPI) 为C/C++函数的编写者提供了在GPPC函数中运行SQL命令的能力。有关SPI函数的其他信息，请参阅PostgreSQL文档中的[服务器编程接口](#)。

。

GPPC API公开了PostgreSQL SPI函数的子集。通过该子集，您可以在GPPC函数中发出SPI查询并检索SPI结果值。GPPC SPI包装器函数是：

SPI函数名	GPPC函数名	描述
SPI_connect()	GppcSPIConnect()	连接到Greenplum数据库服务器编程接口。
SPI_finish()	GppcSPIFinish()	断开与Greenplum数据库服务器编程接口的连接。
SPI_exec()	GppcSPIExec()	执行SQL语句，返回行数。
SPI_getvalue()	GppcSPIGetValue()	从SQL结果中按编号检索特定属性的值作为字符串。
	GppcSPIGetDatum()	从SQL结果中按编号检索特定属性的值作为GppcDatum。
	GppcSPIGetValueByName()	按名称从SQL结果中检索特定属性的值作为字符串。
	GppcSPIGetDatumByName()	按名称从SQL结果中检索特定属性的值作为GppcDatum。

创建访问服务器编程接口的GPPC函数时，您的函数应符合以下流程：

```
GppcSPIConnect();
GppcSPIExec(...)

// process the results - GppcSPIGetValue(...), GppcSPIGetDatum(...)

GppcSPIFinish()
```

您可以使用GppcSPIExec()在GPPC函数中执行SQL语句。调用此函数时，还可以标识要返回的最大行数。GppcSPIExec()的函数签名是：

```
GppcSPIResult GppcSPIExec(const char *sql_statement, long rcount);
```

GppcSPIExec()返回GppcSPIResult结构。该结构表示SPI结果数据。它包括指向数据的指针，有关处理的行数的信息，计数器和结果代码。GPPC API定义此结构如下：

```
typedef struct GppcSPIResultData
{
    struct GppcSPITupleTableData    *tuptable;
    uint32_t                      processed;
    uint32_t                      current;
    int                           rescode;
} GppcSPIResultData;
typedef GppcSPIResultData *GppcSPIResult;
```

您可以设置和使用GppcSPIResult结构中的current字段来检查tuptable结果数据的每一行。

以下代码摘录使用GPPC API连接到SPI，执行简单查询，循环查询结果并完成处理：

```
GppcSPIResult    result;
char            *attnname = "id";
char            *query = "SELECT i, 'foo' || i AS val FROM generate_series(1,
10) i ORDER BY 1";
bool           isnull = true;

// connect to SPI
if( GppcSPIConnect() < 0 ) {
    GppcReport(GPPC_ERROR, "cannot connect to SPI");
}

// execute the query, returning all rows
result = GppcSPIExec(query, 0);

// process result
while( result->current < result->processed ) {
    // get the value of attnname column as a datum, making a copy
    datum = GppcSPIGetDatumByName(result, attnname, &isnull, true);

    // do something with value

    // move on to next row
}
```

```

        result->current++;
}

// complete processing
GppcSPIFinish();

```

关于元组描述符和元组

表或一组记录包含一个或多个元组（行）。元组的每个属性的结构由元组描述符定义。元组描述符为元组中的每个属性定义以下内容：

- 属性名称
- 属性数据类型的对象标识符
- 属性数据类型的字节长度
- 属性修饰符的对象标识符

GPPC API定义了一个抽象类型GppcTupleDesc来表示元组/行描述符。API还提供了可用于创建，访问和设置元组描述符的函数：

函数名称	描述
GppcCreateTemplateTupleDesc()	创建具有指定数量的属性的空元组描述符。
GppcTupleDescInitEntry()	在指定位置向元组描述符添加属性。
GppcTupleDescNatts()	获取元组描述符中的属性数。
GppcTupleDescAttrName()	获取元组描述符中特定位置（从0开始）的属性名称。
GppcTupleDescAttrType()	获取元组描述符中特定位置（从0开始）的属性的类型对象标识符。
GppcTupleDescAttrLen()	获取元组描述符中特定位置（从0开始）的属性的类型长度。
GppcTupleDescAttrTypmod()	获取元组描述符中特定位置（从0开始）的属性的类型修饰符对象标识符。

要构造元组描述符，首先要创建一个模板，然后为每个属性填写描述符字段。这些函数的签名是：

```

GppcTupleDesc GppcCreateTemplateTupleDesc(int natts);
void GppcTupleDescInitEntry(GppcTupleDesc desc, uint16_t attno,
                           const char *attname, GppcOid typid, int32_t typmod);

```

在某些情况下，您可能希望从现有元组中的属性定义初始化元组描述符条目。以下函数获取元组描述符中的属性数，以及描述符中特定属性（按编号）的定义：

```

int GppcTupleDescNatts(GppcTupleDesc tupdesc);
const char *GppcTupleDescAttrName(GppcTupleDesc tupdesc, int16_t attno);
GppcOid GppcTupleDescAttrType(GppcTupleDesc tupdesc, int16_t attno);
int16_t GppcTupleDescAttrLen(GppcTupleDesc tupdesc, int16_t attno);
int32_t GppcTupleDescAttrTypmod(GppcTupleDesc tupdesc, int16_t attno);

```

以下示例初始化两个属性元组描述符。第一个属性使用来自不同描述符的属性定义进行初始化，第二个属性初始化为布尔类型属性：

```

GppcTupleDesc      tdesc;
GppcTupleDesc      indesc = some_input_descriptor;

// initialize the tuple descriptor with 2 attributes
tdesc = GppcCreateTemplateTupleDesc(2);

// use third attribute from the input descriptor
GppcTupleDescInitEntry(tdesc, 1,
                      GppcTupleDescAttrName(indesc, 2),
                      GppcTupleDescAttrType(indesc, 2),
                      GppcTupleDescAttrTypmod(indesc, 2));

// create the boolean attribute
GppcTupleDescInitEntry(tdesc, 2, "is_active", GppcOidBool, 0);

```

GPPC API定义了一个抽象类型GppcHeapTuple来表示元组/记录/行。元组由其元组描述符定

义，每个元组属性的值以及每个值是否为NULL的指示符。

GPPC API提供了可用于设置和访问元组及其属性的函数：

函数名称	描述
GppcHeapFormTuple()	从GppcDatum数组中形成一个元组。
GppcBuildHeapTupleDatum()	从GppcDatum数组中形成一个GppcDatum元组。
GppcGetAttributeByName()	按名称从元组中获取属性。
GppcGetAttributeByNum()	从数字中获取元组的属性（从1开始）。

构建元组GPPC函数的签名是：

```
GppcHeapTuple GppcHeapFormTuple(GppcTupleDesc tupdesc, GppcDatum *values, bool
*nulls);
GppcDatum    GppcBuildHeapTupleDatum(GppcTupleDesc tupdesc, GppcDatum *values,
bool *nnulls);
```

以下代码摘录从上面的代码示例中的元组描述符构造GppcDatum元组，并从函数的整数和布尔输入参数构造：

```
GppcDatum intarg = GPPC_GETARG_INT4(0);
GppcDatum boolarg = GPPC_GETARG_BOOL(1);
GppcDatum result, values[2];
bool nulls[2] = { false, false };

// construct the values array
values[0] = intarg;
values[1] = boolarg;
result = GppcBuildHeapTupleDatum( tdesc, values, nulls );
```

Set-Returning函数

其签名包括RETURNS SETOF RECORD或RETURNS TABLE(...)的Greenplum数据库UDF是set-returning函数。

GPPC API为GPPC函数返回集（例如，多行/元组）提供支持。Greenplum数据库为每个行或项目调用一次set-returning函数（SRF）。该函数必须保存足够的状态以记住它正在做什么并返回每次调用的下一行。您在SRF上下文中分配的内存必须在多个函数调用中存活。

GPPC API提供宏和函数，以帮助跟踪和设置此上下文，并分配SRF内存。他们包括：

函数/宏名称	描述
GPPC_SRF_RESULT_DESC()	获取此SRF的输出行元组描述符。结果元组描述符由输出表定义或DESCRIBE函数确定。
GPPC_SRF_IS_FIRSTCALL()	确定这是否是对SRF的第一次调用。
GPPC_SRF_FIRSTCALL_INIT()	初始化SRF上下文。
GPPC_SRF_PERCALL_SETUP()	在每次调用SRF时恢复上下文。
GPPC_SRF_RETURN_NEXT()	从SRF返回值并继续处理。
GPPC_SRF_RETURN_DONE()	SRF处理完成的信号。
GppSRFAcc()	在此SRF上下文中分配内存。
GppSRFAcc0()	在此SRF上下文中分配内存并将其初始化为零。
GppSRFSave()	在此SRF上下文中保存用户状态。
GppSRFRestore()	在此SRF上下文中还原用户状态。

GppcFuncCallContext结构提供SRF的上下文。您在第一次调用SRF时创建此上下文。您的set-returning GPPC函数必须在每次调用时检索函数上下文。例如：

```
// set function context
GppcFuncCallContext fctx;
if (GPPC_SRF_IS_FIRSTCALL()) {
    fctx = GPPC_SRF_FIRSTCALL_INIT();
```

```

    }
fctx = GPPC_SRF_PERCALL_SETUP();
// process the tuple

```

GPPC函数必须在返回元组结果时提供上下文或指示处理已完成。例如：

```

GPPC_SRF_RETURN_NEXT(fctx, result_tuple);
// or
GPPC_SRF_RETURN_DONE(fctx);

```

使用DESCRIBE函数定义使用RETURNS SETOF RECORD子句的函数的输出元组描述符。 使用GPPC_SRF_RESULT_DESC()宏获取使用RETURNS TABLE(...)子句的函数的输出元组描述符。

有关set-returning函数代码和部署示例，请参阅[GPPC Set-Returning函数示例](#)。

表函数

GPPC API提供GppcAnyTable类型以将表作为输入参数传递给函数，或者将表作为函数结果返回。

GPPC API中提供的与表相关的函数和宏包括：

函数/宏名称	描述
GPPC_GETARG_ANYTABLE()	获取任何表函数参数。
GPPC_RETURN_ANYTABLE()	返回表。
GppcAnyTableGetTupleDesc()	获取表的元组描述符。
GppcAnyTableGetNextTuple()	获取表中的下一行。

您可以使用GPPC_GETARG_ANYTABLE()宏来检索表输入参数。当您有权访问该表时，可以使用GppcAnyTableGetTupleDesc()函数检查该表的元组描述符。该函数的签名是：

```
GppcTupleDesc GppcAnyTableGetTupleDesc(GppcAnyTable t);
```

例如，要检索作为函数的第一个输入参数的表的元组描述符：

```

GppcAnyTable      intbl;
GppcTupleDesc     in_desc;

intbl = GPPC_GETARG_ANYTABLE(0);
in_desc = GppcAnyTableGetTupleDesc(intbl);

```

GppcAnyTableGetNextTuple()函数从表中获取下一行。同样，要从上表中检索下一个元组：

```

GppcHeapTuple     ntuple;

ntuple = GppcAnyTableGetNextTuple(intbl);

```

限制

使用Greenplum数据库版本5.0.x的GPPC API不支持以下运算符：

- integer || integer
- integer = text
- text < integer

样例代码

Greenplum数据库github存储库中的[gppc test](#) 目录包含示例GPPC代码：

- `gppc_demo/` - 示例代码，用于执行GPPC SPI函数，错误报告，数据类型参数和返回宏，set-returning函数和编码函数
- `tabfunc_gppc_demo/` - 示例代码执行GPPC表和set-returning函数

使用PGXS构建GPPC共享库

您可以将使用GPPC API编写的函数编译到Greenplum数据库服务器按需加载的一个或多个共享库中。

您可以使用PostgreSQL构建扩展基础结构（PGXS）根据Greenplum数据库安装为您的GPPC函数构建源代码。该框架自动化简单模块的通用构建规则。如果您有一个更复杂的用例，则需要编写自己的构建系统。

要使用PGXS基础结构为使用GPPC API创建的函数生成共享库，请创建一个设置PGXS特定变量的简单Makefile。

Note: 有关PGXS支持的Makefile变量的信息，请参阅PostgreSQL文档中的[扩展的构建基础设施](#)。

例如，以下Makefile从名为src1.c和src2.c的两个C源文件生成名为sharedlib_name.so的共享库：

```
MODULE_big = sharedlib_name
OBJS = src1.o src2.o
PG_CPPFLAGS = -I$(shell $(PG_CONFIG) --includedir)
SHLIB_LINK = -L$(shell $(PG_CONFIG) --libdir) -lgppc

PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
```

MODULE_big标识Makefile生成的共享库的基本名称。

PG_CPPFLAGS将Greenplum数据库安装包含目录添加到编译器头文件搜索路径中。

SHLIB_LINK将Greenplum数据库安装库目录添加到链接器搜索路径。此变量还将GPPC库（-lgppc）添加到link命令。

PG_CONFIG和PGXS变量设置和include语句是必需的，通常位于Makefile的最后三行。

使用Greenplum数据库注册GPPC函数

在用户可以从SQL调用GPPC函数之前，必须使用Greenplum数据库注册该函数。

注册GPPC函数涉及将GPPC函数签名映射到SQL用户定义的函数。您可以使用CREATE FUNCTION ... AS 命令定义此映射，以指定GPPC共享库名称。您可以选择为GPPC和SQL函数使用相同的名称或不同的名称。

示例CREATE FUNCTION ... AS 语法如下：

```
CREATE FUNCTION sql_function_name(arg[, ...]) RETURNS return_type
AS 'shared_library_path'[, 'gppc_function_name']
LANGUAGE C STRICT [WITH (DESCRIBE=describe_function)];
```

指定shared_library_path时，可以省略共享库.so扩展名。

如果GPPC函数在名为gppc_try.so的共享库中编译和链接，则以下命令将本主题前面引用的示例add_int4s()函数注册到名为add_two_int4s_gppc()的SQL UDF：

```
CREATE FUNCTION add_two_int4s_gppc(int4, int4) RETURNS int8
AS 'gppc_try.so', 'add_int4s'
LANGUAGE C STRICT;
```

关于动态加载

您可以在CREATE FUNCTION ... AS命令中指定GPPC共享库的名称，以在Greenplum数据库的共享库中注册GPPC函数。Greenplum数据库动态加载程序在用户第一次调用在该共享库中链接的用户定义函数时将GPPC共享库文件加载到内存中。如果在CREATE FUNCTION ... AS命令中未提供共享库的绝对路径，Greenplum数据库将尝试使用以下有序步骤找到库：

1. 如果共享库文件路径以字符串\$libdir开头，则Greenplum数据库将在PostgreSQL包库目录中查找该文件。运行pg_config --pkglibdir命令以确定此目录的位置。
2. 如果指定了没有目录前缀的共享库文件名，则Greenplum数据库将在dynamic_library_path服务器配置参数值标识的目录中搜索该文件。
3. 当前的工作目录。

打包和部署注意事项

您必须以适合Greenplum集群中Greenplum数据库管理员部署的形式打包GPPC共享库和SQL函数注册脚本。提供GPPC包的特定部署说明。

构建程序包和部署说明时，请考虑以下事项：

- 考虑提供Greenplum数据库管理员运行的shell脚本或程序，以便将共享库安装到所需的文件系统位置并注册GPPC函数。
- 必须将GPPC共享库安装到master主机上的相同文件系统位置以及Greenplum数据库群集中的每个segment主机上。
- gpadmin用户必须具有遍历GPPC共享库文件的完整文件系统路径的权限。
- 安装在Greenplum数据库部署中后，GPPC共享库的文件系统位置决定了在使用CREATE FUNCTION ... AS命令在库中注册函数时如何引用共享库。
- 创建一个.sql脚本文件，为GPPC共享库中的每个GPPC函数注册一个SQL UDF。您在.sql注册脚本中创建的函数必须引用GPPC共享库的部署位置。在GPPC部署包中包含此脚本。
- 记录运行GPPC包部署脚本的说明（如果提供）。
- 如果未在程序包部署脚本中包含此任务，请记录有关安装GPPC共享库的说明。
- 如果未在程序包部署脚本中包含此任务，请记录有关安装和运行函数注册脚本的说明。

GPPC文本函数示例

在此示例中，您将开发、构建和部署GPPC共享库，并注册并运行名为concat_two_strings的GPPC函数。此函数使用GPPC API连接两个字符串参数并返回结果。

您将在Greenplum数据库主控主机上开发GPPC函数。部署您在此示例中创建的GPPC共享库需要对Greenplum数据库集群的管理访问权限。

执行以下过程以运行该示例：

1. 登录Greenplum数据库主控主机并设置您的环境。例如：

```
$ ssh gpadmin@<gpmaster>
gpadmin@gpmaster$ . /usr/local/greenplum-db/greenplum_path.sh
```

2. 创建工作目录并导航到新目录。例如：

```
gpadmin@gpmaster$ mkdir gppc_work
gpadmin@gpmaster$ cd gppc_work
```

3. 通过在您选择的编辑器中打开文件，为GPPC源代码准备文件。例如，要使用vi打开名

为gppc_concat.c的文件：

```
gpadmin@gpmaster$ vi gppc_concat.c
```

4. 将以下代码复制/粘贴到文件中：

```
#include <stdio.h>
#include <string.h>
#include "gppc.h"

// make the function SQL-invokable
GPPC_FUNCTION_INFO(concat_two_strings);

// declare the function
GppcDatum concat_two_strings(GPPC_FUNCTION_ARGS);

GppcDatum
concat_two_strings(GPPC_FUNCTION_ARGS)
{
    // retrieve the text input arguments
    GppcText arg0 = GPPC_GETARG_TEXT(0);
    GppcText arg1 = GPPC_GETARG_TEXT(1);

    // determine the size of the concatenated string and allocate
    // text memory of this size
    size_t arg0_len = GppcGetTextLength(arg0);
    size_t arg1_len = GppcGetTextLength(arg1);
    GppcText retstring = GppcAllocText(arg0_len + arg1_len);

    // construct the concatenated return string
    memcpy(GppcGetTextPointer(retstring), GppcGetTextPointer(arg0), arg0_len);
    memcpy(GppcGetTextPointer(retstring) + arg0_len, GppcGetTextPointer(arg1),
arg1_len);

    GPPC_RETURN_TEXT( retstring );
}
```

代码声明并实现了concat_two_strings()函数。它使用GPPC数据类型，宏和函数来获取函数参数，为连接的字符串分配内存，将参数复制到新字符串中，然后返回结果。

5. 保存文件并退出编辑器。

6. 在您选择的编辑器中打开名为Makefile的文件。将以下文本复制/粘贴到文件中：

```
MODULE_big = gppc_concat
OBJS = gppc_concat.o

PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)

PG_CPPFLAGS = -I$(shell $(PG_CONFIG) --includedir)
SHLIB_LINK = -L$(shell $(PG_CONFIG) --libdir) -lgppc
include $(PGXS)
```

7. 保存文件并退出编辑器。

8. 为concat_two_strings()函数构建GPPC共享库。例如：

```
gpadmin@gpmaster$ make all
```

make命令在当前工作目录中生成名为gppc_concat.so的共享库文件。

9. 将共享库复制到Greenplum数据库安装。您必须具有Greenplum数据库管理权限才能复制该文件。例如：

```
gpadmin@gpmaster$ cp gppc_concat.so /usr/local/greenplum-db/lib/postgresql/
```

10. 将共享库复制到Greenplum数据库安装中的每个主机。例如，如果seghostfile包含Greenplum数据库集群中segment主机的列表，每个主机行：

```
gpadmin@gpmaster$ gpscp -v -f seghostfile /usr/local/greenplum-
db/lib/postgresql/gppc_concat.so =:/usr/local/greenplum-
db/lib/postgresql/gppc_concat.so
```

11. 打开psql会话。例如：

```
gpadmin@gpmaster$ psql -d testdb
```

12. 使用Greenplum数据库注册名为concat_two_strings()的GPPC函数。例如，将Greenplum数据库函数concat_with_gppc()映射到GPPC concat_two_strings()函数：

```
testdb=# CREATE FUNCTION concat_with_gppc(text, text) RETURNS text
AS 'gppc_concat', 'concat_two_strings'
LANGUAGE C STRICT;
```

13. 运行concat_with_gppc()函数。例如：

```
testdb=# SELECT concat_with_gppc( 'happy' , 'monday' );
concat_with_gppc
-----
happymonday
(1 row)
```

GPPC Set-Returning函数示例

在此示例中，您将开发，构建和部署GPPC共享库。您还可以为名为return_tbl()的GPPC函数创建并运行.sql注册脚本。此函数使用GPPC API获取带有整数和文本列的输入表，确定整数列是否大于13，并返回带有输入整数列的结果表和一个标识整数是否为的整数的布尔列return_tbl()使用GPPC API报告和SRF函数和宏。

您将在Greenplum数据库master主机上开发GPPC函数。部署您在此示例中创建的GPPC共享库需要对Greenplum数据库集群的管理访问权限。

执行以下过程以运行该示例：

1. 登录Greenplum数据库master主机并设置您的环境。例如：

```
$ ssh gpadmin@<gpmaster>
gpadmin@gpmaster$ . /usr/local/greenplum-db/greenplum_path.sh
```

2. 创建工作目录并导航到新目录。例如：

```
gpadmin@gpmaster$ mkdir gppc_work
gpadmin@gpmaster$ cd gppc_work
```

3. 通过在您选择的编辑器中打开文件，为GPPC代码准备源文件。例如，要使用vi打开名为gppc_concat.c的文件：

```
gpadmin@gpmaster$ vi gppc_rettbl.c
```

4. 将以下代码复制/粘贴到文件中：

```
#include <stdio.h>
#include <string.h>
#include "gppc.h"

// initialize the logging level
GppcReportLevel level = GPPC_INFO;

// make the function SQL-invokable and declare the function
GPPC_FUNCTION_INFO(return_tbl);
GppcDatum return_tbl(GPPC_FUNCTION_ARGS);

GppcDatum
return_tbl(GPPC_FUNCTION_ARGS)
{
    GppcFuncCallContext fctx;
    GppcAnyTable        inttbl;
```

```

GppcHeapTuple      intuple;
GppcTupleDesc     in_tupdesc, out_tupdesc;
GppcBool          resbool = false;
GppcDatum         result, boolres, values[2];
bool              nulls[2] = {false, false};

// single input argument - the table
intbl = GPPC_GETARG_ANYTABLE(0);

// set the function context
if (GPPC_SRF_IS_FIRSTCALL()) {
    fctx = GPPC_SRF_FIRSTCALL_INIT();
}
fctx = GPPC_SRF_PERCALL_SETUP();

// get the tuple descriptor for the input table
in_tupdesc = GppcAnyTableGetTupleDesc(intbl);

// retrieve the next tuple
intuple = GppcAnyTableGetNextTuple(intbl);
if( intuple == NULL ) {
    // no more tuples, conclude
    GPPC_SRF_RETURN_DONE(fctx);
}

// get the output tuple descriptor and verify that it is
// defined as we expect
out_tupdesc = GPPC_SRF_RESULT_DESC();
if (GppcTupleDescNattrs(out_tupdesc) != 2 || 
    GppcTupleDescAttrType(out_tupdesc, 0) != GppcOidInt4 ||
    GppcTupleDescAttrType(out_tupdesc, 1) != GppcOidBool) {
    GppcReport(GPPC_ERROR, "INVALID out_tupdesc tuple");
}

// log the attribute names of the output tuple descriptor
GppcReport(level, "output tuple descriptor attr0 name: %s",
GppcTupleDescAttrName(out_tupdesc, 0));
GppcReport(level, "output tuple descriptor attr1 name: %s",
GppcTupleDescAttrName(out_tupdesc, 1));

// retrieve the attribute values by name from the tuple
bool text_isnull, int_isnull;
GppcDatum intdat = GppcGetAttributeByName(inttuple, "id", &int_isnull);
GppcDatum textdat = GppcGetAttributeByName(inttuple, "msg", &text_isnull);

// convert datum to specific type
GppcInt4 intarg = GppcDatumGetInt4(intdat);
GppcReport(level, "id: %d", intarg);
GppcReport(level, "msg: %s",
GppcTextGetCString(GppcDatumGetText(textdat)));

// perform the >13 check on the integer
if( !int_isnull && (intarg > 13) ) {
    // greater than 13?
    resbool = true;
    GppcReport(level, "id is greater than 13!");
}

// values are datums; use integer from the tuple and
// construct the datum for the boolean return
values[0] = intdat;
boolres = GppcBoolGetDatum(resbool);
values[1] = boolres;

// build a datum tuple and return
result = GppcBuildHeapTupleDatum(out_tupdesc, values, nulls);
GPPC_SRF_RETURN_NEXT(fctx, result);

}

```

代码声明并实现了return_tbl()函数。它使用GPPC数据类型，宏和函数来获取函数参数，检查元组描述符，构建返回元组，并返回结果。该函数还使用SRF宏来跟踪函数调用之间的元组上下文。

5. 保存文件并退出编辑器。

6. 在您选择的编辑器中打开名为Makefile的文件。将以下文本复制/粘贴到文件中：

```

MODULE_big = gppc_rettbl
OBJS = gppc_rettbl.o

```

```

PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)

PG_CPPFLAGS = -I$(shell $(PG_CONFIG) --includedir)
SHLIB_LINK = -L$(shell $(PG_CONFIG) --libdir) -lgppc
include $(PGXS)

```

7. 保存文件并退出编辑器。

8. 为return_tbl()函数构建GPPC共享库。例如：

```
gpadmin@gpmaster$ make all
```

make命令在当前工作目录中生成名为gppc_rettbl.so的共享库文件。

9. 将共享库复制到Greenplum数据库安装。您必须具有Greenplum数据库管理权限才能复制该文件。例如：

```
gpadmin@gpmaster$ cp gppc_rettbl.so /usr/local/greenplum-db/lib/postgresql/
```

此命令将共享库复制到\$libdir

10. 将共享库复制到Greenplum数据库安装中的每个主机。例如，如果seghostfile包含Greenplum数据库集群中segment主机的列表，每个主机一行：

```

gpadmin@gpmaster$ gpscp -v -f seghostfile /usr/local/greenplum-
db/lib/postgresql/gppc_rettbl.so =:/usr/local/greenplum-
db/lib/postgresql/gppc_rettbl.so

```

11. 创建.sql文件以注册GPPC return_tbl()函数。在您选择的编辑器中打开名为gppc_rettbl_reg.sql的文件。

12. 将以下文本复制/粘贴到文件中：

```

CREATE FUNCTION rettbl_gppc(anytable) RETURNS TABLE(id int4, thirteen bool)
  AS 'gppc_rettbl', 'return_tbl'
LANGUAGE C STRICT;

```

13. 通过运行刚刚创建的脚本来注册GPPC功能。例如，要在名为testdb的数据库中注册该函数：

```
gpadmin@gpmaster$ psql -d testdb -f gppc_rettbl_reg.sql
```

14. 打开psql会话。例如：

```
gpadmin@gpmaster$ psql -d testdb
```

15. 创建包含一些测试数据的表。例如：

```

CREATE TABLE gppc_testtbl( id int, msg text );
INSERT INTO gppc_testtbl VALUES (1, 'f1');
INSERT INTO gppc_testtbl VALUES (7, 'f7');
INSERT INTO gppc_testtbl VALUES (10, 'f10');
INSERT INTO gppc_testtbl VALUES (13, 'f13');
INSERT INTO gppc_testtbl VALUES (15, 'f15');
INSERT INTO gppc_testtbl VALUES (17, 'f17');

```

16. 运行rettbl_gppc()函数。例如：

```

testdb=# SELECT * FROM rettbl_gppc(TABLE(SELECT * FROM gppc_testtbl));
   id | thirteen
-----+
    1 | f
    7 | f
   13 | f
   15 | t
   17 | t
   10 | f
(6 rows)

```


Greenplum数据库® 6.0文档

- 参考指南

- SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

- 系统目录参考

- gp_toolkit管理模式

- gpperfmon 数据库

- Greenplum 数据库数据类型

字符集支持

- 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

可以扩展Greenplum数据库以在单独的进程中运行用户提供的代码。这些进程由postgres启动，停止和监视，这使得它们的生命周期与服务器的状态紧密相关。这些进程可以选择连接到Greenplum数据库的共享内存区域并在内部连接到数据库；它们还可以串行运行多个事务，就像常规的客户端连接服务器进程一样。此外，通过链接到libpq，它们可以连接到服务器，并且行为类似于常规客户端应用程序。

Warning: 使用后台工作进程存在相当大的稳健性和安全性风险，因为使用C语言编写时，它们可以不受限制地访问数据。希望启用包含后台工作进程的模块的管理员应该格外小心。只允许经过仔细审核的模块运行后台工作进程。

通过在`shared_preload_libraries`服务器配置参数中包含模块名称，可以在启动Greenplum数据库时初始化后台工作程序。希望运行后台工作程序的模块可以通过从`_PG_init()`调用`RegisterBackgroundWorker(BackgroundWorker *worker)`来注册它。通过调用函数`RegisterDynamicBackgroundWorker(BackgroundWorker *worker, BackgroundWorkerHandle **handle)`，系统启动并运行后，也可以启动后台工作程序。与只能在`postmaster`中调用的`RegisterBackgroundWorker`不同，必须从常规后端调用`RegisterDynamicBackgroundWorker`。

因此定义了`BackgroundWorker`结构：

```
typedef void (*bgworker_main_type)(Datum main_arg);
typedef struct BackgroundWorker
{
    char          bgw_name[BGW_MAXLEN];
    int           bgw_flags;
    BgWorkerStartTime bgw_start_time;
    int           bgw_restart_time;      /* in seconds, or
BGW_NEVER_RESTART */
    bgworker_main_type bgw_main;
    char          bgw_library_name[BGW_MAXLEN]; /* only if
bgw_main is NULL */
    char          bgw_function_name[BGW_MAXLEN]; /* only if
bgw_main is NULL */
    Datum         bgw_main_arg;
    int           bgw_notify_pid;
} BackgroundWorker;
```

`bgw_name`是一个在日志消息，进程列表和类似上下文中使用的字符串。

`bgw_flags`是一个按位或位掩码，表示模块想要的功能。可能的值是`BGWORKER_SHMEM_ACCESS`（请求共享内存访问）和`BGWORKER_BACKEND_DATABASE_CONNECTION`（请求建立数据库连接）。

的能力，以后可以通过它运行事务和查询）。使用**BGWORKER_BACKEND_DATABASE_CONNECTION**连接到数据库的后台工作进程还必须使用**BGWORKER_SHMEM_ACCESS**附加共享内存，否则工作进程将启动失败。

bgw_start_time是**postgres**应该启动进程的服务器状态；它可以是**BgWorkerStart_PostmasterStart**之一（一旦**postgres**完成自己的初始化就开始；请求它的进程不符合数据库连接的条件），**BgWorkerStart_ConsistentState**（一旦在热备份中达到一致状态就开始，允许进程连接到数据库并运行只读查询）和**BgWorkerStart_RecoveryFinished**（系统进入正常读写状态后立即启动）。请注意，最后两个值在不是热备份的服务器中是等效的。请注意，此设置仅指示何时启动进程；当达到不同的状态时，它们不会停止。

bgw_restart_time是**postgres**在重新启动进程之前应该等待的时间间隔（以秒为单位），以防它崩溃。它可以是任何正值，或**BGW_NEVER_RESTART**，表示在发生崩溃时不重启进程。

bgw_main是指向启动进程时要运行的函数的指针。此函数必须采用**Datum**类型的单个参数并返回**void**。**bgw_main_arg**将作为唯一参数传递给它。请注意，全局变量**MyBgworkerEntry**指向在注册时传递的**BackgroundWorker**结构的副本。**bgw_main**可能为NULL；在这种情况下，将使用**bgw_library_name**和**bgw_function_name**来确定入口点。这对**postmaster**启动后启动的后台工作进程很有用，其中**postmaster**没有加载必需的库。

bgw_library_name是库的名称，在该库中应该寻找后台工作程序的初始入口点。除非**bgw_main**为NULL，否则它将被忽略。但是如果**bgw_main**为NULL，则命名库将由工作进程动态加载，**bgw_function_name**将用于标识要调用的函数。

bgw_function_name是动态加载库中函数的名称，该库应该用作新后台工作程序的初始入口点。除非**bgw_main**为NULL，否则它将被忽略。

bgw_notify_pid是一个Greenplum数据库后端进程的PID，**postmaster**应该在进程启动或退出时向其发送SIGUSR1。对于在**postmaster**启动时注册的工作进程，或者注册工作进程的后端不希望等待工作进程启动时，应该为0。否则，它应该初始化为**MyProcPid**。

一旦运行，该进程可以通过调用**BackgroundWorkerInitializeConnection**(char *dbname, char *username)连接到数据库。这允许进程使用SPI接口运行事务和查询。如果**dbname**为NULL，则会话未连接到任何特定数据库，但可以访问共享目录。如果**username**为NULL，则进程将以initdb期间创建的超级用户身份运行。**BackgroundWorkerInitializeConnection**只能在每个后台进程中调用一次，无法切换数据库。

当控制到达bgw_main函数时，信号最初被阻止，并且必须被它解除阻塞；这是为了允许进程在必要时自定义其信号处理程序。通过调用BackgroundWorkerUnblockSignals可以在新进程中取消阻止信号，并通过调用BackgroundWorkerBlockSignals来阻止信号。

如果后台工作程序的bgw_restart_time配置为BGW_NEVER_RESTART，或者退出时退出代码为0或由TerminateBackgroundWorker终止，则退出时将由postmaster自动取消注册。否则，它将在通过bgw_restart_time配置的时间段之后重新启动，或者如果postmaster由于后端故障而重新初始化群集时立即重新启动。需要暂时暂停执行的后端应该使用可中断的睡眠而不是退出；这可以通过调用WaitLatch()来实现。确保在调用该函数时设置WL_POSTMASTER_DEATH标志，并在postgres本身已终止的紧急情况下验证返回代码以提示退出。

当使用RegisterDynamicBackgroundWorker函数注册后台工作程序时，执行注册的后端可以获得有关工作进程状态的信息。希望这样做的后端应该将BackgroundWorkerHandle *的地址作为RegisterDynamicBackgroundWorker的第二个参数传递。如果工作程序已成功注册，则此指针将使用不透明句柄进行初始化，该句柄随后可以传递给GetBackgroundWorkerPid(BackgroundWorkerHandle *, pid_t *)或TerminateBackgroundWorker(BackgroundWorkerHandle *)。GetBackgroundWorkerPid可用于轮询工作进程的状态：返回值BGWH_NOT_YET_STARTED表示工作进程尚未由postmaster启动；BGWH_STOPPED表示它已经启动但不再运行；和BGWH_STARTED表示它当前正在运行。在最后一种情况下，PID也将通过第二个参数返回。TerminateBackgroundWorker使postmaster在运行时将SIGTERM发送给worker，并在它不运行时立即取消注册。

在某些情况下，注册后台工作进程的进程可能希望等待工作进程启动。这可以通过将bgw_notify_pid初始化为MyProcPid，然后将在注册时获得的BackgroundWorkerHandle *传递给WaitForBackgroundWorkerStartup(BackgroundWorkerHandle *handle, pid_t *)函数来实现。此函数将阻塞，直到postmaster尝试启动后台工作程序，或直到postmaster死亡。如果后台运行器正在运行，则返回值将为BGWH_STARTED，并且PID将被写入提供的地址。否则，返回值将为BGWH_STOPPED或BGWH_POSTMASTER_DIED。

worker_spi模块包含一个工作示例，演示了一些有用的技术。

注册后台工作进程的最大数量受max-worker-processes限制。

Parent topic: [Greenplum数据库参考指南](#)

Greenplum数据库® 6.0文档

- 参考指南
 - SQL Command Reference
 - SQL 2008可选特性兼容性
 - Greenplum环境变量
 - 保留标识符和SQL关键字
 - 系统目录参考
 - gp_toolkit管理模式
 - gpperfmon 数据库
 - Greenplum 数据库数据类型
 - 字符集支持
 - 服务器配置参数
 - 内置函数摘要
 - Greenplum MapReduce规范
 - Greenplum PL/pgSQL过程语言
 - Greenplum PL/R 语言扩展
 - Greenplum PL/Python语言扩展
 - Greenplum PL/Container语言扩展
 - Greenplum的PL/Java语言扩展

该部分提供了Greenplum数据库的系统要求和功能集合的高级概述。它包含以下主题：

- [Greenplum SQL标准一致性](#)
- [Greenplum和PostgreSQL兼容性](#)

Parent topic: [Greenplum数据库参考指南](#)

Greenplum SQL标准一致性

SQL语言于1986年被美国国家标准学会 (ANSI) 第一次作为SQL正式标准化。SQL标准的后续版本已由ANSI和国际标准化组织 (ISO) 标准发布：SQL 1989, SQL 1992, SQL 1999, SQL 2003, SQL 2006, 和最后的SQL 2008，它就是当前的SQL标准。该标准的正式名称为ISO/IEC 9075-14:2008。一般来说，每个更新的版本都增加了更多的内容，虽然偶尔也有一些内容被启用或者删除。

重要的是要注意，没有完全遵从SQL标准的商业数据库系统。Greenplum数据库几乎完全符合SQL 1992的标准，多数功能来源于SQL 1999。几个来源于SQL 2003的功能也被实现了（最著名的是SQL OLAP功能）。

该部分针对Greenplum数据库和SQL标准相关的重要一致性的问题。有关对最新的SQL标准的支持功能列表，请参阅[SQL 2008可选特性兼容性](#)。

核心SQL一致性

在构建并行、无共享架构的数据库系统和查询优化器的过程中，某些常见的SQL结构尚未在Greenplum数据库中实现。不支持以下的SQL结构：

1. 有些设置在EXISTS或NOT EXISTS子句中返回子查询，Greenplum的并行优化器不能将之重写为到连接之中。
2. 向后回滚游标，包括FETCH PRIOR, FETCH FIRST,  ABSOLUTE, 和FETCH RELATIVE操作的使用。
3. 在CREATE TABLE语句上（哈希分布表）：UNIQUE或PRIMARY

Greenplum的PL/Perl语言

KEY子句必须包括分布键列的所有值，或者是其超集。因为这个限制，在CREATE TABLE的语句中，仅允许一个UNIQUE子句或者PRIMARY KEY子句。UNIQUE或PRIMARY KEY子句不允许出现在随机分布的表中。

4. CREATE UNIQUE INDEX语句不包含分布键的列的所有值或者为其超集。CREATE UNIQUE INDEX不允许使用在随机分布的表上。
注意UNIQUE INDEXES（但是不是UNIQUE CONSTRAINTS）在分布表的单个部分上执行，它们保证每个部分或者子部分内的键值的唯一性。
5. VOLATILE或STABLE函数不能在segment上执行，因此通常仅限于传递文字值作为其参数的参数。
6. 触发器是不支持的，因为他么通常依赖于VOLATILE函数的使用。
7. 引用完整性约束（外键）不会再Greenplum数据库中实施。用户可以声明外键，但是这些信息保存在系统catalog中。
8. 序列操纵函数CURRVAL和LASTVAL。

SQL 1992 —致性

以下 SQL 1992 的特性在Greenplum数据库中不支持：

1. NATIONAL CHARACTER (NCHAR) 和NATIONAL CHARACTER VARYING (NVARCHAR)。用户可以声明NCHAR和NVARCHAR类型，但是它们只是Greenplum数据库中CHAR和VARCHAR的同义词。
2. CREATE ASSERTION语句。
3. INTERVAL文字在Greenplum数据库中是支持的，但是不符合标准。
4. GET DIAGNOSTICS语句。
5. GLOBAL TEMPORARY TABLE和LOCAL TEMPORARY TABLE。
Greenplum TEMPORARY TABLE不符合SQL标准，但许多商业数据库系统以相同的方式实现了临时表。Greenplum临时表与Teradata中的VOLATILE TABLE相同。
6. UNIQUE断言。

7. 引用完整性检查的MATCH PARTIAL (很可能不会在Greenplum数据库中实现)。

SQL 1999 —致性

以下SQL 1999的功能在Greenplum数据库中不支持：

1. Large Object数据类型：BLOB, CLOB, NCLOB。但是，Greenplum数据库中该BYTEA和TEXT列可以存储大量的数据（数百兆字节）。
2. MODULE (SQL 客户端模块)。
3. CREATE PROCEDURE (SQL/PSM)。这可通过创建返回值为void的FUNCTION在Greenplum数据库中进行操作，如下调用函数：

```
SELECT myfunc(args);
```
4. 该PostgreSQL/Greenplum函数定义语言 (PL/PGSQL) 是Oracle的PL/SQL的子集，而不是和SQL/PSM函数定义语言的兼容。Greenplum数据库还支持使用Python, Perl, Java和R定义函数。
5. BIT和BIT VARYING数据类型 (故意忽略)。这些在S

[Greenplum数据库® 6.0文档](#)[Greenplum平台扩展框架\(PXF\)](#)[PXF介绍](#)[PXF管理手册](#)[使用PXF访问hadoop](#)[使用PXF访问Azure, Google云端存储, Minio和S3对象存储](#)[使用PXF访问SQL数据库\(JDBC\)](#)[PXF故障排除](#)[PXF实用程序手册](#)[Back to the Administrator Guide](#)[Greenplum database 5管理员指南](#)[Greenplum database 4管理员指南](#)[FAQ](#)

[如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息](#)

In this topic:

- [架构概述](#)
- [连接器\(Connector\),服务器\(Servers\)和配置文件\(Profiles\)](#)
- [创建一个外部表](#)
- [PXF的其他特性](#)

Greenplum Platform Extension Framework(PXF)提供的连接器(connectors)可以用于访问存储在Greenplum数据库外部源中的数据。这些连接器将外部数据源映射到Greenplum数据库的外部表(external table)中。创建Greenplum数据库外部表时，你可以通过在命令中提供服务器名称和配置文件名称来标识外部数据存储和数据格式。

您可以通过Greenplum数据库查询外部表引用的数据,您也可以使用外部表将数据加载到Greenplum数据库中以获得更高的性能。

架构概述

GPDB集群包含一个master节点(master node)和多个segment主机(segment host)。GPDB segment主机上的PXF客户端进程为对外部表进行查询的每个segment instance分配工作线程。多个segment主机的PXF代理与外部数据存储并行通信。

连接器(Connector),服务器(Servers)和配置文件(Profiles)

连接器是一个通用的术语,他封装了读取和写入外部数据存储所需要的实现细节。PXF提供创建了与Hadoop (HDFS,Hive,Hbase),对象存储(Azure,Google Cloud Storage, Minio, S3)和sql数据库(通过jdbc)的连接器。

PXF服务器是连接器的命名配置。服务器定义提供PXF访问外部数据源所需的信息。此配置信息是特定于数据存储的，并且可以包括服务器位置，访问凭据和其他相关属性。

Greenplum数据库管理员将为每个允许Greenplum数据库用户访问的外部数据存储配置至少一个服务器定义，并在适当时发布可用的服务器名称。

默认的PXF服务是default(保留), 在没有配置SERVER=时提供外部数据源的位置和访问信息。创建外部表时, 可以指定

`SERVER=<server_name>` 设置, 以标识从中获取配置的服务器配置和访问外部数据存储的凭据。

GPDB数据库管理员将为每个允许GPDB用户访问的外部数据存储配置至少一个服务定义, 并将根据需求发布可用的服务名。默认的PXF服务器名为 `default` (保留), 在配置后, 如果没有

`SERVER=<server_name>` 设置, 则将提供外部数据源的位置和访问信息。

最后, PXF配置文件是一个命名映射, 用于标识特定外部数据存储支持的特定数据格式或协议。PXF支持text, Avro, JSON, RCFfile, Parquet, SequenceFile和ORC数据格式以及JDBC协议, 并提供了一些内置配置文件, 如以下部分所述。

创建一个外部表

PXF实现了一个叫做pxf的GPDB协议, 你可以使用这个协议去创建一个外部表。指定pxf协议的 `CREATE EXTERNAL TABLE` 命令语法如下:

```
CREATE [WRITABLE] EXTERNAL TABLE <table_name>
      ( <column_name> <data_type> [, ...] | LIKE <other_table>
)
LOCATION('pxf://<path-to-data>?PROFILE=<profile_name>[&SERVER=
<server_name>][&<custom-option>=<value>[...]]')
FORMAT '[TEXT|CSV|CUSTOM]' (<formatting-properties>);
```

在创建语句 `CREATE EXTERNAL TABLE` 中的 `LOCATION` 子句是一个URI。这个URI标识描述外部数据位置的路径和其他信息。例如:如果外部数据存储的是HDFS, 则填写指定HDFS文件的绝对路径。如果外部数据存储的是HIVE, 则需要指定符合模式的HIVE表名称。

使用问号(?)引入的URI的查询部分来标识PXF服务器和配置文件名称。

PXF可能需要额外的信息来读取和写入某些数据格式, 可以使用LOCATION字符串的可选组件 = 来提供配置文件的信息, 并通过字符串的组件提供格式信息。

Table 1. CREATE EXTERNAL TABLE参数值和描述

Keyword	Value and Description
---------	-----------------------

<path-to-data>	目录, 文件名, 通配符模式, 表名等。的语法取决于外部数据源
PROFILE= <profile_name>	PXF用于访问数据的配置文件。PXF支持 Hadoop services , object stores , and other SQL databases .
SERVER= <server_name>	PXF用于访问数据的命名服务器配置。可选的; 如果未指定, PXF将使用 default 服务器。
<custom-option>= <value>	配置文件或服务器支持的其他选项及其值
FORMAT <value>	PXF支持 TEXT , CSV 和 CUSTOM 格式
<formatting-properties>	格式化配置文件支持的属性; 例如, FORMATTER 或 delimiter

Note: 创建PXF外部表时, 不能在格式化程序规范中使用HEADER选项

PXF的其他特性

某些PXF连接器和配置文件支持谓词下推和列投影。有关此支持的详细信息, 请参阅以下主题: - [About PXF Filter Pushdown](#) - [About Column Projection in PXF](#)

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

 PXF介绍 PXF管理手册 配置PXF

PXF安装和配置目录

为PXF安装JAVA环境

初始化PXF

配置PXF服务器

 配置PXF HADOOP连接器 (可选)

配置Minio和S3对象存储的连接器 (可选)

配置Azure和Google云端存储的连接器 (可选)

配置JDBC连接器 (可选)

配置PXF客户端主机和端口(可选)

升级PXF

PXF的启动、停止和重启

授权用户访问PXF

注册PXF的jar依赖

监控PXF

In this topic:

- [PXF安装目录](#)
- [PXF运行目录](#)
- [PXF用户配置目录](#)

在你安装Greenplum数据库时，PXF被安装在master和segment节点上。

PXF安装目录

在安装Greenplum时，以下配置文件和目录将被安装在你的GPDB实例中。这些文件/目录在PXF的安装目录\$GPHOME/pxf:

Directory	Description
apache-tomcat/	PXF的Tomcat目录
bin/	PXF脚本和可执行文件目录
conf/	PXF内部配置目录，这个目录包含 <code>pxf-env-default.sh</code> 和 <code>pxf-profiles-default.xml</code> 配置文件。在初始化pxf后，这个目录也会包括 <code>pxf-private.classpath</code> 文件
lib/	PXF库目录
templates/	PXF的配置模板

PXF运行目录

在初始化和启动过程中，PXF在\$GPHOME/pxf创建了如下内部目录：

Directory	Description
pxf-service/	PXF初始化之后的PXF服务实例目录
run/	启动pxf后，pxf的运行目录。包含PXF catalina进程ID文件

PXF用户配置目录

在pxf初始化过程中，PXF使用以下的子目录和模板文件填充你选择的用户配置目录(`$PXF_CONF`)

Directory	Description
conf/	用户可自定义的PXF配置文件位置： <code>pxf-env.sh</code> , <code>pxf-log4j.properties</code> 和 <code>pxf-profiles.xml</code>
keytabs/	PXF服务Kerberos秘钥文件的默认位置
lib/	默认的PXF用户运行库目录
logs/	PXF运行时间日志文件目录，包括 <code>pxf-service.log</code> 和 <code>Tomcat</code> 相关日志 <code>catalina.out</code> . logs 目录和日志文件都是 <code>gpadmin</code> 用户创建的只读文件
servers/	服务配置目录，每一个子目录标识服务名称。默认服务器名为 <code>default</code> 。数据库管理员也可以配置其他服务
templates/	连接器服务器模板文件的配置目录

参数 [初始化PXF](#) and [启动PXF](#) 了解PXF初始化和启动命令和过程的详细信息

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

[PXF介绍](#)

[PXF管理手册](#)

[使用PXF访问hadoop](#)

[使用PXF访问Azure, Google云端存储, Minio和S3对象存储](#)

使用PXF访问SQL数据库
(JDBC)

PXF故障排除

[PXF实用程序手册](#)

[Back to the Administrator Guide](#)

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

In this topic:

- [架构](#)
- [先决条件](#)
- [HDFS Shell 命令入门](#)
- [连接器,数据格式和配置文件](#)

PXF与Cloudera, Hortonworks Data Platform, MapR和通用Apache Hadoop发行版兼容。PXF安装有HDFS, Hive和HBase连接器。您可以使用这些连接器从以上Hadoop发行版访问各种格式的数据。

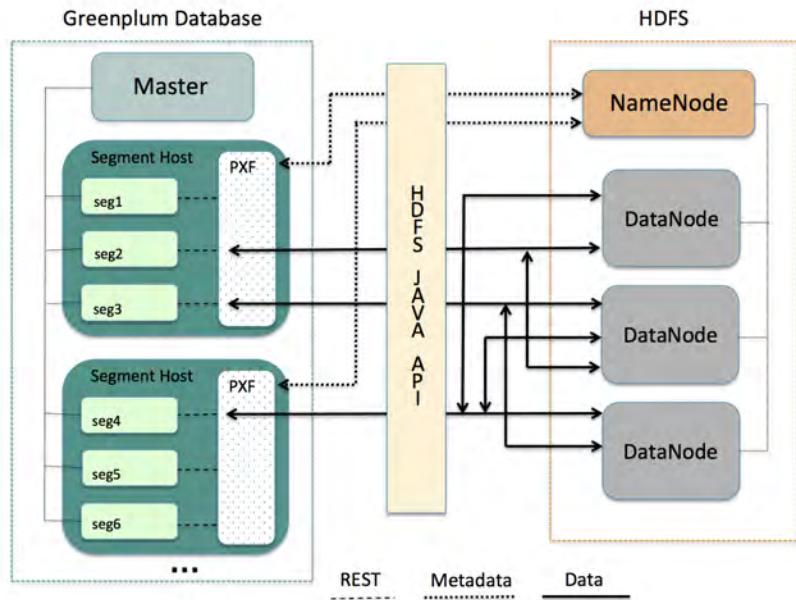
在先前版本的Greenplum数据库中，您可能使用 `gphdfs` 协议的外部表来访问存储在Hadoop中的数据。Greenplum数据库6.0.0版本移除了 `gphdfs` 协议。在Greenplum数据库6.x版本中使用PXF和 `pxf` 协议外部表来访问Hadoop。

架构

HDFS是Apache Hadoop使用的主要分布式存储机制。当用户或应用程序在引用了HDFS文件的PXF外部表上执行查询时，Greenplum数据库master节点将查询分发给所有的segment节点。每个Segment实例请求运行在其主机上的PXF代理。当它从segment实例接受请求时，PXF代理：

1. 分配工作线程以处理来自segment实例的请求。
2. 调用HDFS Java API从HDFS NameNode请求HDFS文件的元数据信息。
3. 提供HDFS NameNode返回的元数据信息给segment实例。

图: PXF-到-Hadoop 架构



Segment实例使用它在Greenplum数据库中的 `gp_segment_id` 和元数据描述的文件块信息将查询所需数据的特定部分分配给自己。然后，segment实例向PXF代理发送请求来读取分配的数据。该数据可以存储在一个或多个HDFS数据节点上。

PXF代理调用HDFS Java API来读取数据并将其传递给segment实例。segment实例将其部分数据传递给Greenplum数据库master节点。此通信跨segment节点和segment实例并行发生。

先决条件

在使用PXF处理Hadoop数据之前，请确保：

- 您已经配置并初始化PXF，并且PXF正在每台segment主机上运行。更多详情，请参阅[配置PXF](#)。
- 您已经配置了计划使用的PXF Hadoop连接器。有关说明，请参阅[配置PXF Hadoop 连接器](#)。如果您计划访问存储在Cloudera Hadoop集群中的JSON格式数据，则PXF需要Cloudera 5.8或者更高的Hadoop发行版。
- 如果开启了用户模拟(默认)，确保您已将Greenplum数据库外部表需要访问到的HDFS文件及目录的读取(并根据需要写入)权限，授予给了每个需要访问这些文件和目录的Greenplum数据库用户/角色的名称。如果未开启用户模拟，您必须要将权限授予给 `gpadmin` 用户。
- Greenplum数据库segment主机和外部Hadoop系统之间的时间是同步的。

HDFS Shell 命令入门

Hadoop包含与HDFS文件系统直接交互的命令行工具。这些工具支持典型的文件系统操作，包括复制和列出文件，更改文件权限等。

HDFS 文件系统命令语法为 `hdfs dfs <选项> [<文件>]`。在没有选项的情况下调用，`hdfs dfs` 将列出该工具支持的文件系统选项。

调用 `hdfs dfs` 命令的用户必须具有HDFS存储数据的读取权限才能列出和查看目录及文件内容，并具有写入权限才能创建目录和文件。

PXF Hadoop主题使用的 `hdfs dfs` 选项包括：

选项	描述
<code>-cat</code>	显示文件内容
<code>-mkdir</code>	在HDFS中创建目录
<code>-put</code>	将文件从本地文件系统复制到HDFS中

例：

在HDFS中创建目录：

```
$ hdfs dfs -mkdir -p /data/exampledир
```

将文本文件从本地文件系统复制到HDFS：

```
$ hdfs dfs -put /tmp/example.txt /data/exampledир/
```

显示位于HDFS中的文本文件的内容：

```
$ hdfs dfs -cat /data/exampledир/example.txt
```

连接器,数据格式和配置文件

PXF Hadoop连接器提供内置配置文件以支持以下数据格式：

- Text
- Avro
- JSON
- ORC
- Parquet

- RCFFile
- SequenceFile
- AvroSequenceFile

PXF Hadoop连接器公开以下配置文件以读取这些支持的数据格式，并在许多情况下写入：

数据源	数据格式	配置文件名称	弃用的配置文件名称
HDFS	单行的分隔文本	hdfs:text	HdfsTextSimple
HDFS	含有被双引号引起的换行符的分隔文本	hdfs:text:multi	HdfsTextMulti
HDFS	Avro	hdfs:avro	Avro
HDFS	JSON	hdfs:json	Json
HDFS	Parquet	hdfs:parquet	Parquet
HDFS	AvroSequenceFile	hdfs:AvroSequenceFile	n/a
HDFS	SequenceFile	hdfs:SequenceFile	SequenceWritable
Hive	TextFile 存储格式	Hive, HiveText	n/a
Hive	SequenceFile 存储格式	Hive	n/a
Hive	RCFile 存储格式	Hive, HiveRC	n/a
Hive	ORC 存储格式	Hive, HiveORC, HiveVectorizedORC	n/a
Hive	Parquet 存储格式	Hive	n/a
HBase	任意存储格式	HBase	n/a

使用 `CREATE EXTERNAL TABLE` 命令指定 `pxf` 协议时，提供配置文件名称，以

创建引用Hadoop文件、目录或表的Greenplum数据库外部表。例如，以下命令创建一个使用默认服务器的外部表，并指定名为 `hdfs:text` 的配置文件：

```
CREATE EXTERNAL TABLE pxf_hdfs_text(location text, month text,
num_orders int, total_sales float8)
LOCATION ('pxf://data/pxf_examples/pxf_hdfs_simple.txt?
PROFILE=hdfs:text')
FORMAT 'TEXT' (delimiter=E',');
```

存储

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

[PXF介绍](#)

[PXF管理手册](#)

[使用PXF访问hadoop](#)

[使用PXF访问Azure, Google云端存储, Minio和S3对象存储](#)

[使用PXF访问SQL数据库 \(JDBC\)](#)

[PXF故障排除](#)

[PXF实用程序手册](#)

[Back to the Administrator Guide](#)

[Greenplum database 5管理员指南](#)

[Greenplum database 4管理员指南](#)

[FAQ](#)

如果您发现翻译不妥之处，欢迎点击“反馈”按钮
[提交详细信息](#)

In this topic:

- [先决条件](#)
- [连接器、数据格式和配置文件](#)
- [示例CREATE EXTERNAL TABLE命令](#)

PXF安装了Azure Blob Storage, Azure Data Lake, Google Cloud Storage, Minio和S3对象存储的连接器

先决条件

在使用PXF处理对象存储数据之前，请确保：

- 您已经配置并初始化了PXF，并且PXF正在每台主机上运行。有关其他信息，请参阅[配置PXF](#)
- 您已经配置了计划使用的PXF对象存储连接器。有关说明，请参阅[配置Azure、Google云端存储、Minio和S3对象存储的连接器](#)。
- Greenplum数据库segment主机与外部对象存储系统之间的时间是同步的。

连接器、数据格式和配置文件

PXF对象存储连接器提供内置配置文件支持以下数据格式：

- Text
- Avro
- JSON
- Parquet
- AvroSequenceFile
- SequenceFile

与Azure、Google Cloud Storage、Minio和S3的PXF连接器提供了以下配置文件以读取、写入(在许多情况下)这些受支持的数据格式:

数据格式	Azure Blob Storage	Azure Data Lake	Google Cloud Storage	S3 或 Minio
分隔的单行 纯文本	wasbs:text	adl:text	gs:text	s3:text
分隔的 带引号的换行符文本	wasbs:text:multi	adl:text:multi	gs:text:multi	s3:text:multi
Avro	wasbs:avro	adl:avro	gs:avro	s3:avro
JSON	wasbs:json	adl:json	gs:json	s3:json
Parquet	wasbs:parquet	adl:parquet	gs:parquet	s3:parquet
AvroSequenceFile	wasbs:AvroSequenceFile	adl:AvroSequenceFile	gs:AvroSequenceFile	s3:AvroSequenceFile
SequenceFile	wasbs:SequenceFile	adl:SequenceFile	gs:SequenceFile	s3:SequenceFile

在 `CREATE EXTERNAL TABLE` 命令中指定 `pxf` 协议以创建引用特定对象存储中的文件或目录的Greenplum数据库外部表时，可以提供配置文件名称。

示例CREATE EXTERNAL TABLE命令

以下命令创建了一个引用S3上的文本文件的外部表。指定名为 `s3:text` 的配置文件和名为 `s3srvcfg` 的服务配置：

```
CREATE EXTERNAL TABLE pxf_s3_text(location text, month text, num_orders int, total_sales float8)
  LOCATION ('pxf://S3_BUCKET/pxf_examples/pxf_s3_simple.txt?PROFILE=s3:text&SERVER=s3srvcfg')
  FORMAT 'TEXT' (delimiter=E',');
```

以下命令创建了一个引用Azure Blob Storage上文本文件的外部表。指定名为 `wasbs:text` 的配置文件和名为 `wasbssrvcfg` 的服务配置。您将提供Azure Blob Storage 容器标识和您的 Azure Blob Storage 账户。

```
CREATE EXTERNAL TABLE pxf_wasbs_text(location text, month text, num_orders int, total_sales float8)
  LOCATION
  ('pxf://AZURE_CONTAINER@YOUR_AZURE_BLOB_STORAGE_ACCOUNT_NAME.blob.core.windows.net/path/to/blob/f:');
```

```
FORMAT 'TEXT';
```

以下命令创建了一个引用Azure Data Lake上文本文件的外部表。指定名为 `adl:text` 的配置文件和名为 `adlsrvcfg` 的服务配置。您将提供您的Azure Data Lake 账户。

```
CREATE EXTERNAL TABLE pxf_adl_text(location text, month text, num_orders int, total_sales
float8)
LOCATION
('pxf://YOUR_ADL_ACCOUNT_NAME.azuredatalakestore.net/path/to/file?PROFILE=adl:text&SERVER=adlsrvc:
FORMAT 'TEXT';
```

以下命令创建了一个引用Google Cloud Storage上JSON文件的外部表。指定名为 `gs:json` 的配置文件和名为 `gcssrvcfg` 的服务配置：

```
CREATE EXTERNAL TABLE pxf_gsc_json(location text, month text, num_orders int, total_sales
float8)
LOCATION ('pxf://dir/subdir/file.json?PROFILE=gs:json&SERVER=gcssrvcfg')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

Greenplum数据库® 6.0文档
Greenplum平台扩展框架(PXF)
□ PXF介绍
□ PXF管理手册
□ 使用PXF访问hadoop
□ 使用PXF访问Azure, Google云端存储, Minio和S3对象存储
使用PXF访问SQL数据库 (JDBC)
PXF故障排除
□ PXF实用程序手册
Back to the Administrator Guide
Greenplum database 5管理员指南
Greenplum database 4管理员指南
FAQ

[如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息](#)

In this topic:

- [先决条件](#)
- [数据类型支持](#)
- [访问外部SQL数据库](#)
 - [JDBC自定义选项](#)
 - [示例: 读取和写入PostgreSQL表](#)
- [关于使用命名查询](#)
 - [示例: 读取PostgreSQL查询的结果](#)
- [用DDL覆盖JDBC服务器配置](#)

您的某些数据可能已经存储在外部SQL数据库中。PXF通过PXF JDBC连接器提供对此数据的访问。JDBC连接器是一个JDBC客户端。它可以从SQL数据库(包括MySQL, ORACLE, PostgreSQL, Apache Ignite和Hive)读取或向SQL数据库写入数据。

本节描述如何使用PXF JDBC连接器访问外部SQL数据库中的数据，包括如何创建引用外部数据库表的PXF外部表，向该表查询数据或将数据插入该表中。

写入外部SQL数据库时，JDBC连接器不保证一致性。请注意，如果 `INSERT` 操作失败，部分数据可能会写入外部数据库表中。如果您需要写操作的一致性，请考虑写入到外部数据库的临时表，并仅在验证写操作后才加载到目标表。

先决条件

在您使用PXF JDBC连接器访问外部数据库前，请确保：

- 您已配置并初始化PXF，并且PXF正在每台segment主机上运行。更多详情，请参阅[配置PXF](#)。
- 您可以确定PXF用户配置目录(`$PXF_CONF`)。
- 所有Greenplum数据库segment主机和外部SQL数据库库之间都可以连接。
- 您已经配置了外部SQL数据库，以便从所有Greenplum数据库segment主机进行访问。
- 您已经注册了所有JDBC驱动程序的JAR依赖。
- (推荐)您已经按照[配置PXF JDBC连接器](#)中的描述创建了一个或多

个命名的PXF JDBC连接服务配置。

数据类型支持

PXF JDBC 连接器支持以下数据类型:

- INTEGER, BIGINT, SMALLINT
- REAL, FLOAT8
- NUMERIC
- BOOLEAN
- VARCHAR, BPCHAR, TEXT
- DATE
- TIMESTAMP
- BYTEA

PXF JDBC 连接器不支持上面未列出的任何数据类型。

注意: JDBC连接器不支持读取或写入以字节数组(`byte[]`)存储的Hive数据。

访问外部SQL数据库

PXF JDBC连接器支持一个名为 `Jdbc` 的配置文件。您可以使用此概要文件从外部SQL数据库表读取数据或将数据写入外部SQL数据库表。您还可以使用连接器在外部SQL数据库中运行静态的命名查询并读取结果。

使用以下语法创建引用外部SQL数据库表的Greenplum数据库外部表，并使用JDBC连接器读取或写入数据：要访问远程SQL数据库中的数据，您可以创建一个引用该远程数据库表的可读或可写的Greenplum数据库外部表。Greenplum数据库外部表和远程数据库表或查询结果元组必须具有相同的定义；列名称和类型必须匹配。

使用以下语法创建引用远程SQL数据库表或来自远程数据库的查询结果的Greenplum数据库外部表：

```
CREATE [READABLE | WRITABLE] EXTERNAL TABLE <table_name>
  ( <column_name> <data_type> [, ...] | LIKE
  <other_table> )
LOCATION ('pxf://<external-table-name>|query:'
```

```
<query_name>?PROFILE=Jdbc [&SERVER=<server_name>] [&<custom-option>=<value>[...]]')
FORMAT 'CUSTOM'
(FORMATTER='pxfwritable_import' | 'pxfwritable_export');
```

CREATE EXTERNAL TABLE 命令中使用的特定关键字和值见下表中描述。

关键字	值
<external-table-name>	外部表的全名。 取决于外部SQL数据库，可能包括模式名称和表名称。
query:<query_name>	要在远程SQL数据库中执行的查询的名称。
PROFILE	PROFILE 关键字必须指定为 Jdbc .
SERVER=<server_name>	PXF用于访问数据的命名服务器配置。可选的; 如果未指定，PXF将使用 default 服务器。
<custom-option>=<value>	<custom-option> 是特定于配置文件的。 Jdbc 配置文件的选项将在下一部分讨论。
FORMAT 'CUSTOM'	JDBC CUSTOM FORMAT 支持用于读取操作的内置 'pxfwritable_import' FORMATTER 函数和用于写入操作的内置 'pxfwritable_export' 函数

注意: 在创建PXF外部表时, 不能在 FORMAT 规范中使用 HEADER 选项。

JDBC自定义选项

您可以在 LOCATION URI 中包含 JDBC 连接器自定义选项，并在每个选项前加上 & 符号。 Jdbc 概要文件支持的 CREATE EXTERNAL TABLE <custom-option> 包括：

选项名称	操作	描述
BATCH_SIZE	Write	整数, 标识要批处理到外部SQL数据库的 INSERT 操作数量。PXF始终会验证 BATCH_SIZE 选项, 即使是在读取操作中提供。默认开启批处理。默认值是100。
		SQL

FETCH_SIZE	Read	整数，标识从外部读取时要缓冲的行数。读取行批处理默认情况下处于启用状态；默认读取大小为1000。
QUERY_TIMEOUT	Read/Write	整数，用于标识JDBC驱动程序等待语句执行的时间（以秒为单位）。默认等待时间是无限的。
POOL_SIZE	Write	在 <code>INSERT</code> 操作上启动线程池，并标识线程池中的线程数。默认情况下，线程池是禁用的。
PARTITION_BY	Read	启用读取分区。分区列<column-name>:<column-type>。您只能指定一个分区列。JDBC连接器支持 <code>date</code> , <code>int</code> 和 <code>enum</code> <column-type>值。如果您未标识 <code>PARTITION_BY</code> 列，则单个PXF实例将为读取请求提供服务。
RANGE	Read	当指定 <code>PARTITION_BY</code> 时是必需的。查询范围；用作提示以帮助创建分区。 <code>RANGE</code> 格式取决于分区列的数据类型。当分区列为 <code>enum</code> 类型时， <code>RANGE</code> 必须指定值列表，即<value>:<value>[:<value> [...]]，每种形成它自己的片段。如果分区列是 <code>int</code> 或 <code>date</code> 类型，则 <code>RANGE</code> 必须指定<start-value>:<end-value>，并表示从<start-value>到<end-value>（含）。如果分区列是 <code>date</code> 类型，请使用 <code>yyyy-MM-dd</code> 日期格式。
		如果指定了 <code>PARTITION_BY</code> 且类型为 <code>int</code> 或 <code>date</code> ，则为必填项。一个片段的间隔[:<interval-unit>]。与 <code>RANGE</code> 一

INTERVAL	Read	起使用, 以提示创建分区。在<interval-value>中指定片段的大小。如果分区列是 <code>date</code> 类型, 请使用<interval-unit>指定 <code>year</code> , <code>month</code> 或 <code>day</code> 。当 <code>PARTITION_BY</code> 列为 <code>enum</code> 类型时, PXF会忽略 <code>INTERVAL</code> 。
QUOTE_COLUMNS	Read	控制在构造外部数据库的SQL查询是 PXF 是否应引用列名。指定为 <code>true</code> 强制PXF引用所有列名称; 如果指定任何其他值, PXF不会引用列名。如果未指定 <code>QUOTE_COLUMNS</code> (默认), 当查询中任一字段满足以下条件, PXF自动引用所有列名: - 包含特殊字符, 或 - 混合大小写, 并且外部数据库不支持未引用的混合大小写标识符。

批量写入操作(写)

当外部SQL数据库的JDBC驱动程序支持它时, 批量 `INSERT` 操作可能会大大提升性能。

默认情况下启用批量写, 默认批处理大小为100。要禁用批处理或修改批处理大小的值, 请使用 `BATCH_SIZE` 设置创建PXF外部表:

- `BATCH_SIZE=0` 或 `BATCH_SIZE=1` - 关闭批处理
- `BATCH_SIZE=(n>1)` - 将 `BATCH_SIZE` 设置为 `n`

当外部数据库的JDBC驱动程序不支持批处理时, PXF JDBC 连接器的行为取决于 `BATCH_SIZE` 设置, 如下所述:

- `BATCH_SIZE` 省略 - JDBC 连接器插入时不使用批处理。
- `BATCH_SIZE=(n>1)` - `INSERT` 操作失败并且连接器返回错误。

批量读取操作

默认情况下，PXF JDBC连接器自动批处理从外部数据库表中获取的行。默认的行获取大小为1000。要修改默认的获取大小值，请在创建PXF外部表时指定 `FETCH_SIZE`。例如：

```
FETCH_SIZE=5000
```

如果外部数据库JDBC驱动程序不支持读取时批处理，则必须通过设置 `FETCH_SIZE=0` 来显式禁用读取行批处理。

线程池(写)

当外部数据库的JDBC驱动程序支持线程化时，PXF JDBC连接器可以通过在多个线程中处理 `INSERT` 操作来进一步提升性能。

考虑将批处理和线程池一起使用。当一起使用时，每个线程将接收并处理一批完整的数据。如果您使用线程池而不使用批处理，则线程池中的每个线程都恰好接收一个元组。

当线程池中的任一线程失败时，JDBC连接器返回一个错误。请注意 `INSERT` 操作失败，部分数据可能会写入外部数据库表中。

要禁用或启动线程池并设置线程池大小，请使用 `POOL_SIZE` 设置创建PXF外部表，如下所述：

- `POOL_SIZE=(n<1)` - 线程池大小是系统中的CPU数量
- `POOL_SIZE=1` - 关闭线程池
- `POOL_SIZE=(n>1)` - 将 `POOL_SIZE` 设为 `n`

分区(读)

PXF JDBC连接器支持运行在多个segment主机上的PXF实例同时对外部SQL表的读取访问。此功能被称为分区。默认情况下，未启用读取分区。要启用读取分区，请在创建PXF外部表时设置 `PARTITION_BY`，`RANGE` 和 `INTERVAL` 自定义选项。

PXF使用您指定的 `RANGE` 和 `INTERVAL` 值以及 `PARTITION_BY` 列将外部表中的特定数据行分配给在Greenplum数据库segment主机上运行的PXF实例。此列选择特定于PXF处理，与您可能已为外部SQL数据

库中的表指定的分区列没有关系。

标识分区参数的示例 JDBC <custom-option> 子字符串：

```
&PARTITION_BY=id:int&RANGE=1:100&INTERVAL=5
&PARTITION_BY=year:int&RANGE=2011:2013&INTERVAL=1
&PARTITION_BY=createdate:date&RANGE=2013-01-01:2016-01-
01&INTERVAL=1:month
&PARTITION_BY=color:enum&RANGE=red:yellow:blue
```

启用分区时，PXF JDBC连接器将 `SELECT` 查询拆分为多个子查询，这些子查询检索数据的子集，每个子集称为一个片段。JDBC连接器会自动向每个片段添加额外的查询约束（`WHERE` 表达式），以确保从外部数据库中检索每个元组的数据都恰好一次。

例如，当用户查询使用指定

`&PARTITION_BY=id:int&RANGE=1:5&INTERVAL=2` 的 `LOCATION` 子句创建的PXF外部表时，PXF会生成5个片段：根据分区设置两个，和最多三个隐式片段生成的碎片。与每个片段相关的约束如下：

- 片段 1: `WHERE (id < 1)` - 隐式生成的片段，用于RANGE起始区间
- 片段 2: `WHERE (id >= 1) AND (id < 3)` - 分区设置指定的片段
- 片段 3: `WHERE (id >= 3) AND (id < 5)` - 分区设置指定的片段
- 片段 4: `WHERE (id >= 5)` - 隐式生成的片段，用于RANGE结束区间
- 片段 5: `WHERE (id IS NULL)` - 隐式生成的片段

PXF在Greenplum数据库segment之间分配片段。在segment主机上运行的PXF实例为服务片段的主机上的每个segment生成一个线程。如果片段的数量小于或等于在片段主机上配置的Greenplum segment的数量，则单个PXF实例可以为所有片段提供服务。每个PXF实例将其结果发送回Greenplum数据库，在此收集它们并将其返回给用户。

当您指定 `PARTITION_BY` 选项时，根据与目标数据库的最佳JDBC连接数以及跨Greenplum数据库segment的最佳外部数据分配，调整 `INTERVAL` 值和单位。`INTERVAL` 低边界由Greenplum数据库segment的数量驱动，而高边界由与目标数据库的可接受的JDBC连接数量驱动。`INTERVAL` 设置会影响片段的数量，理想情况下不应设置得太高或太低。使用多个值进行测试可以帮助您选择最佳设置。

示例: 读取和写入PostgreSQL表

在本例中，您将：

- 创建一个PostgreSQL数据库和表，并将数据写入表中
- 创建一个PostgreSQL用户并将表上的所有权限都赋予该用户
- 配置PXF JDBC 连接器以访问PostgreSQL数据库
- 创建一个引用PostgreSQL表的PXF可读外部表
- 读取PostgreSQL表中的数据
- 创建一个引用PostgreSQL表的PXF可写外部表
- 将数据写入PostgreSQL表
- 再次读取PostgreSQL表中的数据

创建一个PostgreSQL表

执行以下步骤在名为 `pgtestdb` 的数据库的 `public` 模式下创建一个名为 `forpxf_table1` 的PostgreSQL表，并向名为 `pxfuser1` 的用户赋予该表的所有权限：

1. 确定PostgreSQL服务器的主机名和端口。
2. 以 `postgres` 用户连接默认PostgreSQL数据库。例如，假设您的PostgreSQL服务以默认端口运行在 `pserver` 主机上：

```
$ psql -U postgres -h pserver
```

3. 创建一个名为 `pgtestdb` 的PostgreSQL数据库并连接到这个数据库：

```
=# CREATE DATABASE pgtestdb;
=# \connect pgtestdb;
```

4. 创建一个名为 `forpxf_table1` 的表并并向表中写入一些数据：

```
=# CREATE TABLE forpxf_table1(id int);
=# INSERT INTO forpxf_table1 VALUES (1);
=# INSERT INTO forpxf_table1 VALUES (2);
=# INSERT INTO forpxf_table1 VALUES (3);
```

5. 创建一个名为 `pxfuser1` 的PostgreSQL用户：

```
=# CREATE USER pxfuser1 WITH PASSWORD 'changeme';
```

6. 为用户 `pxfuser1` 分配 `forpxf_table1` 表的所有权限，并退出 `psql` 子系统：

```
=# GRANT ALL ON forpxf_table1 TO pxfuser1;
=# \q
```

有了这些权限，`pxfuser1` 可以读取和写入 `forpxf_table1` 表。

7. 更新 PostgreSQL 配置以允许用户 `pxfuser1` 从每个Greenplum数据库segment主机访问 `pgtestdb`。此配置特定于您的PostgreSQL环境。您将更新 `/var/lib/pgsql/pg_hba.conf` 文件，然后重启PostgreSQL服务。

配置JDBC连接器

您必须为PostgreSQL创建JDBC服务配置，将PostgreSQL驱动程序JAR文件下载到您的系统，将该JAR文件复制到PXF用户配置目录，同步PXF配置，然后重启PXF。

此过程通常由Greenplum数据库管理员执行。

1. 登录到Greenplum数据库master节点：

```
$ ssh gpadmin@<gpmaster>
```

2. 如[示例配置步骤](#)中所述为PostgreSQL创建JDBC服务配置，命名服务目录为 `pgsrvcfg`。`jdbc-site.xml` 文件的内容应类似于以下内容(将PostgreSQL主机系统替换为 `pgserverhost`)：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<property>
  <name>jdbc.driver</name>
  <value>org.postgresql.Driver</value>
</property>
<property>
  <name>jdbc.url</name>
  <value>jdbc:postgresql://pgserverhost:5432/pgtestdb</value>
</property>
<property>
  <name>jdbc.user</name>
  <value>pxfuser1</value>
</property>
<property>
  <name>jdbc.password</name>
```

```
<value>changeme</value>
</property>
</configuration>
```

3. 同步PXF 配置到Greenplum数据库集群。例如：

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster sync
```

从PostgreSQL表中读取

执行以下步骤创建一个PXF外部表，该表引用您在上一节创建的 `forpxf_table1` PostgreSQL表，并读取该表中的数据：

1. 指定 `Jdbc` 配置文件创建PXF外部表。例如：

```
gpadmin=# CREATE EXTERNAL TABLE pxf_tblfrompg(id int)
           LOCATION ('pxf://public.forpxf_table1?
PROFILE=Jdbc&SERVER=pgsrvcfg')
           FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

2. 显示 `pxf_tblfrompg` 表的所有行：

```
gpadmin=# SELECT * FROM pxf_tblfrompg;
 id
-----
 1
 2
 3
(3 rows)
```

写入PostgreSQL表

执行以下步骤，将一些数据写入Postgres `forpxf_table1` 表，然后从该表中读取它们。您必须为写操作创建一个新的外部表。

1. 指定 `Jdbc` 配置文件创建一个可写的PXF外部表。例如：

```
gpadmin=# CREATE WRITABLE EXTERNAL TABLE
pxf_writeto_postgres(id int)
           LOCATION ('pxf://public.forpxf_table1?
PROFILE=Jdbc&SERVER=pgsrvcfg')
           FORMAT 'CUSTOM' (FORMATTER='pxfwritable_export');
```

2. 将一些数据写入 `pxf_writeto_postgres` 表。例如：

```
=# INSERT INTO pxf_writeto_postgres VALUES (111);
=# INSERT INTO pxf_writeto_postgres VALUES (222);
=# INSERT INTO pxf_writeto_postgres VALUES (333);
```

3. 使用您在上一节创建的可读外部表 `pxf_tblfrompg` 来查看PostgreSQL `forpxf_table1` 表中的数据：

```
gpadmin=# SELECT * FROM pxf_tblfrompg ORDER BY id DESC;
 id
-----
 333
 222
 111
   3
   2
   1
(6 rows)
```

关于使用命名查询

PXF JDBC连接器允许您指定静态定义的查询以对远程SQL数据库运行。考虑在以下情况下使用命名查询：

- 您需要联接所有都驻留在同一外部数据库中的几个表。
- 您想在数据源附近执行复杂的聚合。
- 您将在外部数据库中使用但不允许创建 `VIEW`。
- 您宁愿消耗外部系统中的计算资源，以最大程度地减少Greenplum数据库资源的利用率。
- 您要运行HIVE查询并通过YARN控制资源利用率。

Greenplum数据库管理员定义了一个查询，并为您提供了创建外部表时要使用的查询名称。在表 `CREATE EXTERNAL TABLE` `LOCATION` 子句中指定 `query:<query_name>` 代替表名，以指示PXF JDBC连接器在远程SQL数据库中运行名为 `<query_name>` 的静态查询。

PXF仅支持具有可读外部表的命名查询。您必须为要运行的每个查询创建一个唯一的Greenplum数据库可读外部表。

外部表列的名称和类型必须与查询结果返回的列的名称，类型和顺序完全匹配。如果查询返回聚合或其他函数的结果，请确保使用 `AS` 限

定符指定特定的列名。

例如，假设您正在使用具有以下定义的PostgreSQL表：

```
CREATE TABLE customers(id int, name text, city text, state text);
CREATE TABLE orders(customer_id int, amount int, month int, year int);
```

这个PostgreSQL查询中，管理员将其命名为 `order_rpt`：

```
SELECT c.name, sum(o.amount) AS total, o.month
  FROM customers c JOIN orders o ON c.id = o.customer_id
 WHERE c.state = 'CO'
GROUP BY c.name, o.month
```

该查询返回类型为 `(name text, total int, month int)` 的元组。如果为名为 `pgserver` 的PXF JDBC服务器定义了 `order_rpt` 查询，则可以创建Greenplum数据库外部表来读取这些查询结果，如下所示：

```
CREATE EXTERNAL TABLE orderrpt_frompg(name text, total int, month int)
  LOCATION ('pxf://query:order_rpt?
PROFILE=Jdbc&SERVER=pgserver&PARTITION_BY=month:int&RANGE=1:13&INTERRUPT_FORMAT='CUSTOM' (FORMATTER='pxfwritable_import'));
```

该命令引用在 `pgserver` 服务器配置中定义的名为 `order_rpt` 的查询。它还指定了JDBC读取分区选项，这些选项为PXF提供了用于在其服务器segment之间划分查询结果数据的信息。

PXF JDBC连接器自动将列投影和过滤器下推应用于引用命名查询的外部表。

示例：读取PostgreSQL查询的结果

在此示例中，您：

- 使用在[示例：读取和写入PostgreSQL数据库](#)中创建的PostgreSQL数据库 `pgtestdb`，用户 `pxfuser1` 和PXF JDBC连接器服务器配置 `pgsrvcfg`。
- 创建两个PostgreSQL表并将数据插入表中。
- 将表上的所有特权分配给 `pxfuser1`。
- 定义一个在两个PostgreSQL表上执行复杂SQL语句的命名查询，并

将该查询添加到 `pgsrvcfg` JDBC服务器配置中。

- 创建一个与查询结果元组匹配的PXF可读外部表定义，并指定读取分区选项。
- 使用PXF列投影和过滤器下推读取查询结果。

创建PostgreSQL表并分配权限

执行以下过程，以在名为 `pgtestdb` 的数据库的 `public` 模式中创建名为 `customers` 和 `orders` 的PostgreSQL表，并向用户 `pxfuser1` 授予这些表的所有特权：

- 确定PostgreSQL服务器的主机名和端口。
- 以postgres用户身份连接到 `pgtestdb` PostgreSQL数据库。例如，如果您的PostgreSQL服务器正在名为 `pserver` 的主机的默认端口上运行：

```
$ psql -U postgres -h pserver -d pgtestdb
```

- 创建一个名为 `customers` 的表，并将一些数据插入该表：

```
CREATE TABLE customers(id int, name text, city text, state text);
INSERT INTO customers VALUES (111, 'Bill', 'Helena', 'MT');
INSERT INTO customers VALUES (222, 'Mary', 'Athens', 'OH');
INSERT INTO customers VALUES (333, 'Tom', 'Denver', 'CO');
INSERT INTO customers VALUES (444, 'Kate', 'Helena', 'MT');
INSERT INTO customers VALUES (555, 'Harry', 'Columbus', 'OH');
INSERT INTO customers VALUES (666, 'Kim', 'Denver', 'CO');
INSERT INTO customers VALUES (777, 'Erik', 'Missoula', 'MT');
INSERT INTO customers VALUES (888, 'Laura', 'Athens', 'OH');
INSERT INTO customers VALUES (999, 'Matt', 'Aurora', 'CO');
```

- 创建一个名为 `orders` 的表，并将一些数据插入该表：

```
CREATE TABLE orders(customer_id int, amount int, month int, year int);
INSERT INTO orders VALUES (111, 12, 12, 2018);
INSERT INTO orders VALUES (222, 234, 11, 2018);
INSERT INTO orders VALUES (333, 34, 7, 2018);
INSERT INTO orders VALUES (444, 456, 111, 2018);
INSERT INTO orders VALUES (555, 56, 11, 2018);
INSERT INTO orders VALUES (666, 678, 12, 2018);
INSERT INTO orders VALUES (777, 12, 9, 2018);
```

```
INSERT INTO orders VALUES (888, 120, 10, 2018);
INSERT INTO orders VALUES (999, 120, 11, 2018);
```

- 为用户 `pxfuser1` 分配表 `customers` 和 `orders` 的所有特权，然后退出 `psql` 子系统：

```
GRANT ALL ON customers TO pxfuser1;
GRANT ALL ON orders TO pxfuser1;
\q
```

配置命名查询

在此过程中，您将创建一个命名查询文本文件，将其添加到 `pgsrvcfg` JDBC服务器配置中，并将PXF配置同步到Greenplum数据库集群。

此过程通常由Greenplum数据库管理员执行。

- 登录到Greenplum数据库主节点：

```
$ ssh gpadmin@<gpmaster>
```

- 导航到JDBC服务器配置目录 `pgsrvcfg`。例如：

```
gpadmin@gpmaster$ cd $PXF_CONF/servers/pgsrvcfg
```

- 在文本编辑器中打开名为 `pg_order_report.sql` 的查询文本文档，然后将以下查询复制/粘贴到该文件中：

```
SELECT c.name, c.city, sum(o.amount) AS total, o.month
  FROM customers c JOIN orders o ON c.id = o.customer_id
 WHERE c.state = 'CO'
 GROUP BY c.name, c.city, o.month
```

- 保存文件并退出编辑器。
- 将对PXF配置的这些更改同步到Greenplum数据库集群。例如：

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster sync
```

阅读查询结果

在Greenplum数据库群集上执行以下过程，以创建一个PXF外部表，该表引用您在上一节中创建的查询文件，然后读取查询结果数据：

1. 创建指定 `Jdbc` 配置文件的PXF外部表。例如：

```
CREATE EXTERNAL TABLE pxf_queryres_frompg(name text, city text,
month int)
LOCATION ('pxf://query:pg_order_report?
PROFILE=Jdbc&SERVER=pgsrcfg&PARTITION_BY=month:int&RANGE=1:13&
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import'));
```

通过这种分区方案，PXF将向远程SQL数据库发出4个查询，每季度一个查询。每个查询将返回目标季度每个月的客户名称以及给定月份中每个客户每月的总订单总额。然后，当您查询外部表时，Greenplum数据库将为您将数据合并到单个结果集中。

2. 显示查询结果的所有行：

```
SELECT * FROM pxf_queryres_frompg ORDER BY city, total;

name | city | total | month
-----+-----+-----+
Matt | Aurora | 120 | 11
Tom | Denver | 34 | 7
Kim | Denver | 678 | 12
(3 rows)
```

3. 使用列投影来显示每个城市的订单总数：

```
SELECT city, sum(total) FROM pxf_queryres_frompg GROUP BY
city;

city | sum
-----+-----
Aurora | 120
Denver | 712
(2 rows)
```

当您执行此查询时，PXF仅请求和检索 `city` 和 `total` 列的查询结果，从而减少了发送回Greenplum数据库的数据量。

4. 提供其他过滤器和聚合以过滤PostgreSQL中的 `total`：

```
SELECT city, sum(total) FROM pxf_queryres_frompg
WHERE total > 100
GROUP BY city;
```

city	sum
Denver	678
Aurora	120
(2 rows)	

在此示例中，PXF将向子查询添加 WHERE 过滤器。该过滤器被推送到远程数据库系统并在其上执行，从而减少了PXF发送回Greenplum数据库的数据量。但是， GROUP BY 聚合不会推送到远程，而是由Greenplum执行。

用DDL覆盖JDBC服务器配置

您可以通过直接在 CREATE EXTERNAL TABLE LOCATION 子句中指定自定义选项，来为特定的外部数据库表覆盖JDBC服务器配置中的某些属性：

自定义选项名称	jdbc-site.xml属性名
JDBC_DRIVER	jdbc.driver
DB_URL	jdbc.url
USER	jdbc.user
PASS	jdbc.password
BATCH_SIZE	jdbc.statement.batchSize
FETCH_SIZE	jdbc.statement.fetchSize
QUERY_TIMEOUT	jdbc.statement.queryTimeout

通过自定义选项指定的示例JDBC连接字符串：

```
&JDBC_DRIVER=org.postgresql.Driver&DB_URL=jdbc:postgresql://pgserver:5432/testdb
&JDBC_DRIVER=com.mysql.jdbc.Driver&DB_URL=jdbc:mysql://mysqlhost:3306/testdb
```

例如：

```
CREATE EXTERNAL TABLE pxf_pgtbl(name text, orders int)
  LOCATION ('pxf://public.forpxf_table1?
PROFILE=Jdbc&JDBC_DRIVER=org.postgresql.Driver&DB_URL=jdbc:postgresql://pgserver:5432/testdb
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_export'));
```

您以这种方式提供的凭据在外部表定义中可见。不要在生产环境中使

用这种传递凭据的方法。

有关PXF用于获取Greenplum数据库用户的配置属性设置的优先级规则的详细信息，请参考[配置属性优先级](#)。

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ PXF介绍

□ PXF管理手册

□ 使用PXF访问hadoop

□ 使用PXF访问Azure, Google云端存储, Minio和S3对象存储

使用PXF访问SQL数据库
(JDBC)

PXF故障排除

In this topic:

- [PXF错误](#)
- [PXF日志](#)
 - 服务级别日志记录
 - 客户端级别日志记录
- [解决PXF内存问题](#)
 - 配置内存不足条件操作
 - 为PXF增加JVM内存
 - 资源受限的PXF segment主机的另一种选择
- [解决PXF JDBC连接器时区错误](#)
- [PXF片段元数据缓存](#)

□ PXF实用程序手册

Back to the Administrator Guide

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

PXF错误

下表描述了使用PXF时可能会遇到的一些错误：

Error Message	Discussion
Protocol “pxf” does not exist	Cause: pxf扩展名未注册。 Solution: 按照PXF启用过程中的说明为 Enable Procedure 为数据库创建(启用)PXF扩展
Invalid URI pxf://<path-to-data>: missing options section	Cause: <code>LOCATION</code> URI配置项不包括配置或其他需要的信息。 Solution: 在URI中提供配置和必需的选项。
org.apache.hadoop.mapred.InvalidInputException: Input path does not exist: hdfs://<namenode>:8020/<path-to-file>	Cause: 在<path-to-file>中指定的HDFS文件不存在。 Solution: 指定现有HDFS文件的路径
	Cause: <schema><hivetable>

PXF日志

启用更详细的消息日志记录可能有助于PXF故障排除工作。PXF提供了两类消息日志记录：服务级别和客户端级别。

服务级别日志记录

PXF利用 `log4j` 进行服务级别的日志记录。PXF-service-related日志消息捕获由 `$PXF_CONF/conf/pxf-log4j.properties` 文件中的 `log4j` 控制。默认的PXF日志记录配置会将INFO和更严格的日志记录写入 `$PXF_CONF/logs/pxf-service.log`。您可以配置日志记录级别和日志文件位置。

启用 `DEBUG` 级别时，PXF 提供更详细日志记录。要配置 PXF `DEBUG` 日志记

录并检查输出，请执行以下操作：

1. 登录gpdb集群的master主机

```
$ ssh gpadmin@<gpmaster>
```

2. 使用编辑器打开 `$PXF_CONF/conf/pxf-log4j.properties`，取消以下行的注释，保存文件，然后退出编辑器：

```
#Log4j.Logger.org.greenplum.pxf=DEBUG
```

3. 使用 `pxf cluster sync` 命令拷贝更新的文件 `pxf-log4j.properties` 到Greenplum数据库集群。例如：

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster sync
```

4. 依照[Restarting PXF](#) 章节描述，重启gpdb集群的每个segment节点的pxf

5. 现在启用了 `DEBUG` 级别的日志记录，您可以执行PXF操作。确保记下时间；这会将信息定向输出到 `$PXF_CONF/logs/pxf-service.log` 中的相关日志。

```
$ date
Wed Oct 4 09:30:06 MDT 2017
$ psql -d <dbname>
```

6. 创建和查询外部表。例如：

```
dbname=> CREATE EXTERNAL TABLE hdfstest(id int, newid int)
          LOCATION ('pxf://data/dir/hdfsfile?PROFILE=hdfs:text')
          FORMAT 'TEXT' (delimiter='E',');
dbname=> SELECT * FROM hdfstest;
<select output>
```

7. 最后，从 `pxf-service.log` 检查/收集日志消息。

Note: `DEBUG` 记录非常冗长，并且会对性能产生影响。在收集了所需的信息之后，请关闭PXF服务的 `DEBUG` 日志记录。

客户端级别日志记录

数据库级客户端日志记录可以提供对内部PXF服务操作的了解。

在psql会话中将 `client_min_messages` 服务器配置参数设置为 `DEBUG2`，可以在对PXF外部表进行操作期间启用Greenplum数据库及PXF调试消息日志记

录。

```
$ psql -d <dbname>
```

```
dbname=# SET client_min_messages=DEBUG2;
dbname=# SELECT * FROM hdfstest;
...
DEBUG2: churl http header: cell #19: X-GP-URL-HOST: seghost1 (seg0
slice1 127.0.0.1:40000 pid=3981)
CONTEXT: External table hdfstest
DEBUG2: churl http header: cell #20: X-GP-URL-PORT: 5888 (seg0 slice1
127.0.0.1:40000 pid=3981)
CONTEXT: External table hdfstest
DEBUG2: churl http header: cell #21: X-GP-DATA-DIR: data/dir/hdfsfile
(seg0 slice1 127.0.0.1:40000 pid=3981)
CONTEXT: External table hdfstest
DEBUG2: churl http header: cell #22: X-GP-OPTIONS-PROFILE: hdfs:text
(seg0 slice1 127.0.0.1:40000 pid=3981)
CONTEXT: External table hdfstest
...
...
```

检查/收集来自stdout的日志消息

Note: `DEBUG2` 数据库会话日志记录会影响性能。记住，在收集了所需的信息之后，请关闭 `DEBUG2` 日志记录。

```
dbname=# SET client_min_messages=NOTICE;
```

解决PXF内存问题

因为单个PXF客户端服务(JVM)为segment主机上的多个segment实例提供服务，所以PXF堆大小可能是限制运行的瓶颈。在并发工作负载时针对大文件的查询，影响更加明显。您可能会遇到由于内存不足或Java垃圾收集器影响响应时间而导致查询挂起或失败的情况。要避免或纠正这些情况，请首先尝试增加Java最大堆大小或减少Tomcat最大线程数，这取决于最适合您系统配置的方式。您还可以选择将PXF配置为在检测到内存不足情况时执行特定的操作。

Note: 本主题中描述的配置更改需要在Greenplum数据库集群的每个节点上修改配置文件。在主服务器上执行更新后，请确保将PXF配置同步到Greenplum数据库集群。

配置内存不足条件操作

在内存不足 (OOM) 的情况下，PXF返回以下错误以响应查询：

```
java.lang.OutOfMemoryError: Java heap space
```

您可以将PXF JVM配置为在检测到OOM条件时启用/禁用以下操作：

- 自动杀死PXF服务器（默认情况下启用）。
- 转储Java堆（默认情况下禁用）。

自动杀死PXF服务器

默认情况下，对PXF进行了配置，以便当PXF JVM在segment主机上检测到内存不足情况时，它将自动运行一个脚本，该脚本将杀死主机上运行的PXF服务器。`PXF_OOM_KILL` 配置属性控制此自动终止行为。

启用自动终止功能后，PXF JVM检测到OOM条件并终止segment主机上的PXF服务器：

- PXF将以下消息记录到segment主机上的`$PXF_CONF/logs/catalina.out`中：

```
===== <date> PXF Out of memory detected <=====
===== <date> PXF shutdown scheduled <=====
```

- 在PXF外部表上运行的任何查询都将失败，并显示以下错误，直到您在segment主机上重新启动PXF服务器为止：

```
... Failed to connect to <host> port 5888: Connection refused
```

当以这种方式关闭segment主机上的PXF服务器时，必须显式重新启动主机上的PXF服务器。有关`pxf start`命令的更多信息，请参见[pxf参考页](#)。

有关禁用/启用此PXF配置属性的说明，请参考下面的配置[procedure](#)。

转储Java堆

在内存不足的情况下，捕获Java堆转储以帮助确定导致资源耗尽的因素可能很有用。您可以使用`PXF_OOM_DUMP_PATH` 属性来配置PXF在检测到OOM条件时将堆转储写入文件。默认情况下，PXF不会在OOM上转储Java堆。

如果选择在OOM上启用堆转储，则必须将`PXF_OOM_DUMP_PATH` 设置为文件或目录的绝对路径：

- 如果指定目录，则PXF JVM将堆转储写入文件`<directory>/java_pid<pid>.hprof`，其中`<pid>` 标识PXF服务器实例的进程ID。每次JVM进行OOM操作时，PXF JVM都会将新文件写入目录。
- 如果指定文件，但该文件不存在，则PXF JVM在检测到OOM时会将堆转储

写入文件。如果文件已经存在，则JVM将不会转储堆。

确保 `gpadmin` 用户对转储文件或目录具有写权限。

Note: 堆转储文件通常很大。如果在OOM上为PXF启用堆转储，并为 `PXF_OOM_DUMP_PATH` 指定一个目录，则多个OOM将在该目录中生成多个文件，并可能消耗大量磁盘空间。如果为 `PXF_OOM_DUMP_PATH` 指定文件，则在文件名不变的情况下磁盘使用率是恒定的。您必须重命名转储文件或配置其他 `PXF_OOM_DUMP_PATH` 才能生成后续的堆转储。

有关启用/禁用此PXF配置属性的说明，请参阅下面的配置[步骤](#)。

步骤

默认情况下，将启用OOM上PXF服务器的自动终止功能。默认情况下，在OOM上禁用堆转储生成。要配置这些属性之一或全部，请执行以下步骤：

1. 登录到您的Greenplum数据库主节点：

```
$ ssh gpadmin@<gpmaster>
```

2. 编辑 `$PXF_CONF/conf/pxf-env.sh` 文件。例如：

```
gpadmin@gpmaster$ vi $PXF_CONF/conf/pxf-env.sh
```

3. 如果您想在OOM上配置（即关闭或重新打开）PXF服务器自动关闭功能，请在 `pxf-env.sh` 文件中找到 `PXF_OOM_KILL` 属性。如果该设置已被注释掉，请取消注释它，然后更新该值。例如，要关闭此行为，请将值设置为 `false`：

```
export PXF_OOM_KILL=false
```

4. 如果您要在PXF服务器达到OOM条件时配置（即打开或关闭）自动堆转储，请在 `pxf-env.sh` 文件中找到 `PXF_OOM_DUMP_PATH` 设置。

1. 要打开此行为，请将 `PXF_OOM_DUMP_PATH` 属性值设置为您希望PXF JVM将Java堆转储到的文件系统位置。例如，要转储到名为 `/home/gpadmin/pxfoom_segh1` 的文件：

```
export PXF_OOM_DUMP_PATH=/home/pxfoom_segh1
```

2. 要在打开堆转储后将其关闭，请注释掉 `PXF_OOM_DUMP_PATH` 属性设置：

```
#export PXF_OOM_DUMP_PATH=/home/pxfoom_segh1
```

5. 保存 `pxf-env.sh` 文件并退出编辑器。

6. 使用 `pxf cluster sync` 命令将更新的 `pxf-env.sh` 文件复制到Greenplum数据库集群。 例如：

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster sync
```

7. 如[重新启动PXF](#)中所述，在每个Greenplum数据库segment主机上重新启动PXF。

为PXF增加JVM内存

在segment主机上运行的每个PXF代理都配置有默认的最大Java堆大小2GB和初始堆大小1GB。如果Greenplum数据库群集中的segment主机具有足够的内存，请尝试将最大堆大小增加到3-4GB之间的值。如果可以，将初始堆大小和最大堆大小设置为相同的值最佳。

执行以下过程来增加在Greenplum数据库集群中每个segment主机上运行的PXF代理服务的堆大小。

1. 登录gpdb集群的master主机

```
$ ssh gpadmin@<gpmaster>
```

2. 编辑 `$PXF_CONF/conf/pxf-env.sh` 文件。 例如：

```
gpadmin@gpmaster$ vi $PXF_CONF/conf/pxf-env.sh
```

3. 在 `pxf-env.sh` 文件中找到 `PXF_JVM_OPTS` 设置，然后将 `-Xmx` 和/或 `-Xms` 选项更新为所需的值。 例如：

```
PXF_JVM_OPTS="-Xmx3g -Xms3g"
```

4. 保存文件并退出编辑器。

5. 使用 `pxf cluster sync` 命令将更新的 `pxf-env.sh` 文件复制到Greenplum数据库集群。 例如：

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster sync
```

6. 依照[Restarting PXF](#) 描述，重启每个segment主机上的pxf服务。

资源受限的PXF segment主机的另一种选择

如果增加最大堆大小不适合您的Greenplum数据库部署，请尝试减少PXF的并发工作线程数。正在运行的线程数量的减少将防止任何PXF节点耗尽其内存，同时确保当前查询运行完毕(尽管速度稍慢)。Tomcat的默认行为是将请求排队，直到线程空闲或队列耗尽为止。

PXF的Tomcat线程的默认最大数量为200。`PXF_MAX_THREADS` 配置属性控制此设置。

PXF线程容量由配置文件以及是否压缩数据确定。如果计划在外部Hive数据存储中的大量文件上运行大型工作负载，或者正在读取压缩的ORC或Parquet数据，请考虑指定较低的 `PXF_MAX_THREADS` 值。

Note: 请记住，线程数用完后，线程数的增加与内存消耗的增加相关。

执行以下过程，以设置在Greenplum数据库部署中每个segment主机上运行的PXF代理的Tomcat线程的最大数量。

1. 登录到gpdb集群的master节点

```
$ ssh gpadmin@<gpmaster>
```

2. 编辑 `$PXF_CONF/conf/pxf-env.sh`。例如：

```
gpadmin@gpmaster$ vi $PXF_CONF/conf/pxf-env.sh
```

3. 在 `pxf-env.sh` 文件中找到 `PXF_MAX_THREADS` 设置。取消注释设置，并将其更新为所需的值。例如，要将Tomcat线程的最大数量设置为100：

```
export PXF_MAX_THREADS=100
```

4. 保存文件并退出编辑器。

5. 使用 `pxf cluster sync` 命令将更新的 `pxf-env.sh` 文件复制到Greenplum数据库集群。例如：

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster sync
```

6. 依照[Restarting PXF](#) 描述重启PXF服务

解决PXF JDBC连接器时区错误

您可以使用PXF JDBC连接器访问存储在外部SQL数据库中的数据。如果PXF SQL

为服务器设置的默认时区与为外部数据库设置的时区不匹配，则取决于JDBC驱动程序，该程序可能会返回错误。

例如，如果使用PXF JDBC连接器访问时区冲突的Oracle数据库，则PXF会记录类似于以下内容的错误：

```
SEVERE: Servlet.service() for servlet [PXF REST Service] in context with
path [/pxf] threw exception
java.io.IOException: ORA-00604: error occurred at recursive SQL level 1
ORA-01882: timezone region not found
```

如果遇到此错误，可以在`$PXF_CONF/conf/pxf-env.sh`配置文件中的`PXF_JVM_OPTS`属性设置中为PXF服务器设置默认时区选项。例如，要设置时区：

```
export PXF_JVM_OPTS=<current_settings> -Duser.timezone=America/Chicago"
```

您也可以使用`PXF_JVM_OPTS`属性来设置其他Java选项。

如前几节所述，您必须将更新的PXF配置同步到Greenplum数据库集群，然后在每个segment主机上重新启动PXF服务器。

PXF片段元数据缓存

PXF连接器*Fragmenter*使用来自外部数据源的元数据将数据拆分为可并行读取的片段列表（块，文件等）。PXF在每个查询的基础上缓存片段元数据：访问片段元数据的第一个线程将信息存储在缓存中，其他线程重用此缓存的元数据。这种性质的缓存减少了具有大量片段的外部数据源对查询内存的需求。

默认情况下，PXF片段元数据缓存处于启用状态。要关闭片段元数据缓存，或在关闭后重新启用片段元数据缓存，请执行以下过程：

1. 登录到您的Greenplum数据库主节点：

```
$ ssh gpadmin@<gpmaster>
```

2. 编辑`$PXF_CONF/conf/pxf-env.sh`文件。例如：

```
gpadmin@gpmaster$ vi $PXF_CONF/conf/pxf-env.sh
```

3. 在`pxf-env.sh`文件中找到`PXF_FRAGMENTER_CACHE`设置。如果该设置已被注释掉，请取消注释它，然后更新该值。例如，要关闭片段元数据缓存，请将值设置为`false`：

```
export PXF_FRAGMENTER_CACHE=false
```

4. 保存文件并退出编辑器。
5. 使用 `pxf cluster sync` 命令将更新的 `pxf-env.sh` 文件复制到Greenplum数据库集群。 例如：

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster sync
```

6. 如[重新启动PXF](#)中所述，在每个Greenplum数据库segment主机上重新启动PXF。

Greenplum平台扩展框架(PXF)包括以下实用程序参考页：

Greenplum数据库® 6.0文档

- [pxf cluster](#)

Greenplum平台扩展框架(PXF)

- [pxf](#)

□ PXF介绍

□ PXF管理手册

□ 使用PXF访问hadoop

□ 使用PXF访问Azure, Google云端存储, Minio和S3对象存储

使用PXF访问SQL数据库
(JDBC)

PXF故障排除

□ **PXF实用程序手册**

[Back to the Administrator Guide](#)

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

[如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息](#)

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 定义外部表

使用PXF访问外部数据

□ 使用外部表访问外部数据

使用Greenplum的并行文件
服务器 (gpfldist)

□ 装载和卸载数据

□ 性能管理

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

您组织管理的数据可能已存在于外部源中，例如Hadoop，对象存储库和其他SQL数据库。Greenplum平台扩展框架 (PXF) 通过内置连接器提供对此外部数据的访问，该连接器将外部数据源映射到Greenplum数据库表定义。

PXF随Hadoop和对象存储连接器一起安装。这些连接器使您能够读取以文本, Avro, JSON, RCFFile, Parquet, SequenceFile和ORC格式存储的外部数据。您可以使用JDBC连接器访问外部SQL数据库。

Note: 在以前版本的Greenplum Database中，您可能已使用 gphdfs 外部表协议来访问存储在Hadoop中的数据。Greenplum 数据库6.0.0版删除了gphdfs 协议。使用PXF和pxf 外部表协议访问Greenplum数据库版本6.x中的Hadoop。

Greenplum平台扩展框架包括协议C库和Java服务。配置并初始化PXF后，在每个Greenplum数据库段主机上启动单个PXF JVM进程。这个长时间运行的进程同时提供多个查询请求。

有关PXF的体系结构和使用PXF的详细信息，请参阅[Greenplum Platform Extension Framework \(PXF\)](#)文档。

Parent topic: [使用外部数据](#)

Parent topic: [装载和卸载数据](#)



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

关于Greenplum的架构

关于管理和监控工具

□ 关于Greenplum数据库中的并发控制

关于并行数据装载

关于Greenplum数据库中的冗余和故障切换

关于Greenplum数据库中的数据库统计信息

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

□ 性能管理

Greenplum database 5管理员指南

Greenplum database 4管理员

Greenplum数据库是一种大规模并行处理 (MPP) 数据库服务器, 其架构特别针对管理大规模分析型数据仓库以及商业智能工作负载而设计。

MPP (也被称为*shared nothing*架构) 指有两个或者更多个处理器协同执行一个操作的系统, 每一个处理器都有其自己的内存、操作系统和磁盘。Greenplum使用这种高性能系统架构来分布数T字节数据仓库的负载并且能够使用系统的所有资源并行处理一个查询。

Greenplum数据库是基于PostgreSQL开源技术的。它本质上是多个PostgreSQL面向磁盘的数据库实例一起工作形成的一个紧密结合的数据库管理系统 (DBMS)。它基于PostgreSQL 9.4开发, 其SQL支持、特性、配置选项和最终用户功能在大部分情况下和PostgreSQL非常相似。与Greenplum数据库交互的数据库用户会感觉在使用一个常规的PostgreSQL DBMS。

Greenplum数据库可以使用追加优化 (append-optimized, AO) 的存储格式来批量装载和读取数据, 并且能提供HEAP表上的性能优势。追加优化的存储为数据保护、压缩和行/列方向提供了校验和。行式或者列式追加优化的表都可以被压缩。

Greenplum数据库和PostgreSQL的主要区别在于:

- 在基于Postgres查询规划器的常规查询规划器之外, 可以利用GPORCA进行查询规划。
- Greenplum数据库可以使用追加优化的存储。
- Greenplum数据库可以选用列式存储, 数据在逻辑上还是组织成一个表, 但其中的行和列在物理上是存储在一种面向列的格式中, 而不是存储成行。列式存储只能和追加优化表一起使用。列式存储是可压缩的。当用户只需要返回感兴趣的列时, 列式存储可以提供更好的性能。所有的压缩算法都可以用在行式或者列式存储的表上, 但是行程编码 (RLE) 压缩只能用于列式存储的表。Greenplum数据库在所有使用列式存储的追加优化表上都提供了压缩。

为了支持Greenplum数据库的并行结构, PostgreSQL的内部已经被修改或者增补。例如, 系统目录、优化器、查询执行器以及事务管理器组件都已经被修改或者增强, 以便能够在所有的并行PostgreSQL数据库实例之上同时执行查询。Greenplum的*interconnect* (网络层) 允许在不同的PostgreSQL实例之间通讯, 让系统表现为一个逻辑数据库。

Greenplum数据库也可以使用声明式分区和子分区来隐式成分区约束。

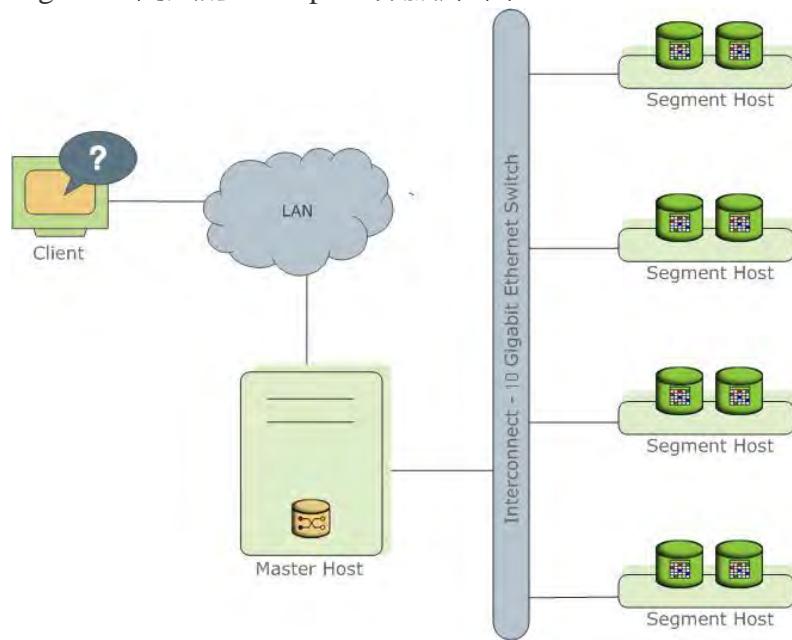
[指南](#)

Greenplum数据库也包括为针对商业智能 (BI) 负载优化PostgreSQL而设计的特性。例如，Greenplum增加了并行数据装载（外部表）、资源管理、查询优化以及存储增强，这些在PostgreSQL中都是无法找到的。很多Greenplum开发的特性和优化都在PostgreSQL社区中找到了一席之地。例如，表分区最初是由Greenplum开发的一个特性，现在已经出现在了标准的PostgreSQL中。

Greenplum数据库的查询使用一种火山式查询引擎模型，其中的执行引擎拿到一个执行计划并且用它产生一棵物理操作符树，然后通过物理操作符计算表，最后返回结果作为查询响应。

Greenplum数据库通过将数据和处理负载分布在多个服务器或者主机上来存储和处理大量的数据。Greenplum数据库是一个由基于PostgreSQL 8.3的数据库组成的阵列，阵列中的数据库工作在一起呈现了一个单一数据库的景象。Master是Greenplum数据库系统的入口。客户端会连接到这个数据库实例并且提交SQL语句。Master会协调与系统中其他称为Segment的数据库实例一起工作，Segment负责存储和处理数据。

Figure 1. 高层的Greenplum数据库架构



下面的主题描述了组成一个Greenplum数据库系统的组件以及它们如何一起工作。

- [关于Greenplum的Master](#)
- [关于Greenplum的Segment](#)
- [关于Greenplum的Interconnect](#)

Parent topic: [Greenplum数据库概念](#)

关于Greenplum的Master

Greenplum数据库的Master是整个Greenplum数据库系统的入口，它接受连接和SQL查询并且把工作分布到Segment实例上。

Greenplum数据库的最终用户与Greenplum数据库（通过Master）交互时，会觉得他们是在与一个典型的PostgreSQL数据库交互。他们使用诸如psql之类的客户端或者JDBC、ODBC、[libpq](#) (PostgreSQL的C语言API) 等应用编程接口（API）连接到数据库。

Master是全局系统目录的所在地。全局系统目录是一组包含了有关Greenplum数据库系统本身的元数据的系统表。Master上不包含任何用户数据，数据只存在于Segment之上。Master会认证客户端连接、处理到来的SQL命令、在Segment之间分布工作负载、协调每一个Segment返回的结果以及把最终结果呈现给客户端程序。

Greenplum数据库使用预写式日志（WAL）来实现主/备镜像。在基于WAL的日志中，所有的修改都会在应用之前被写入日志，以确保对于任何正在处理的操作的数据完整性。

Note: Segment镜像还不能使用WAL日志。

关于Greenplum的Segment

Greenplum数据库的Segment实例是独立的PostgreSQL数据库，每一个都存储了数据的一部分并且执行查询处理的主要部分。

当一个用户通过Greenplum的Master连接到数据库并且发出一个查询时，在每一个Segment数据库上都会创建一些进程来处理该查询的工作。更多有关查询处理的内容，请见[关于Greenplum的查询处理](#)。

用户定义的表及其索引会分布在Greenplum数据库系统中可用的Segment上，每一个Segment都包含数据的不同部分。服务于Segment数据的数据库服务器进程运行在相应的Segment实例之下。用户通过Master与一个Greenplum数据库系统中的Segment交互。

Segment运行在被称作Segment主机的服务器上。一台Segment主机通常运行2至8个Greenplum的Segment，这取决于CPU核数、RAM、存储、网络接口和工作负载。Segment主机预期都以相同的方式配置。从Greenplum数据库获得最佳性能的关键在于在大量能力相同的Segment之间平均地分布数据和工作负载，这样所有的Segment可以同时开始为一个任务工作并且同时完成它们的工作。

关于Greenplum的Interconnect

Interconnect是Greenplum数据库架构中的网络层。

Interconnect指的是Segment之间的进程间通信以及这种通信所依赖的网络基础设施。 Greenplum的Interconnect采用了一种标准的以太交换网络。出于性能原因，推荐使用万兆网或者更快的系统。

默认情况下，Interconnect使用带流控制的用户数据包协议 (UDPIFC) 在网络上发送消息。Greenplum软件在UDP之上执行包验证。这意味着其可靠性等效于传输控制协议 (TCP) 且性能和可扩展性要超过TCP。如果Interconnect被改为TCP，Greenplum数据库会有1000个Segment实例的可扩展性限制。对于Interconnect的默认协议UDPIFC则不存在这种限制。

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

关于Greenplum的架构

关于管理和监控工具

□ 关于Greenplum数据库中的并发控制

关于并行数据装载

关于Greenplum数据库中的冗余和故障切换

关于Greenplum数据库中的数据库统计信息

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

□ 性能管理

Greenplum database 5管理员
指南

Greenplum database 4管理员

Greenplum数据库提供了标准的命令行工具来执行通常的监控和管理任务。

Greenplum的命令行工具位于 `$GPHOME/bin` 目录中并且在Master主机上执行。Greenplum为下列管理任务提供了实用工具：

- 在一个阵列上安装Greenplum数据库
- 初始化一个Greenplum数据库系统
- 开始和停止Greenplum数据库
- 增加或者移除一个主机
- 扩展阵列并且在新的Segment上重新分布表
- 恢复失效的Segment实例
- 管理失效Master实例的故障切换和恢复
- 备份和恢复一个数据库（并行）
- 并行装载数据
- 在Greenplum数据库之间转移数据
- 系统状态报告

Greenplum数据库包括一个包含查询状态和系统指标的可选的性能管理数据库。`gpperfmon_install` 管理工具创建名为`gpperfmon` 的数据库，并启用运行在Greenplum数据库Master和Segment节点上的数据收集代理。运行在节点上的数据收集代理会从节点上收集查询状态，还包括诸如CPU和内存使用量等系统指标。Master节点上的代理周期性的（通常15秒）从节点代理上收集数据并更新`gpperfmon` 数据库。用户可以查询`gpperfmon` 数据库来查看查询和系统指标。

Parent topic: [Greenplum数据库概念](#)



并发控制

Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 关于Greenplum的架构
 - 关于管理和监控工具
 - 关于Greenplum数据库中的并发控制**
 - 关于并行数据装载
 - 关于Greenplum数据库中的冗余和故障切换
 - 关于Greenplum数据库中的数据库统计信息
- 管理一个Greenplum系统
 - 管理Greenplum数据库访问
 - 定义数据库对象
 - 分布与倾斜
 - 插入, 更新, 和删除数据
 - 查询数据
 - 使用外部数据
 - 装载和卸载数据
 - 性能管理
- Greenplum database 5管理员指南
- Greenplum database 4管理员

Greenplum数据库使用了PostgreSQL的多版本并发控制 (MVCC) 模型来管理对于堆表的并发事务。

数据库管理系统中的并发控制允许在确保数据库的完整性的前提下，并发查询能够完成并且得到正确的结果。传统的数据库使用两阶段锁协议来阻止一个事务修改已经被另一个并发事务读取的数据并且阻止任何并发事务读取或者写入另一个事务已经更新的数据。协调事务所需的锁增加了数据库中的竞争，降低了总体事务吞吐量。

Greenplum数据库使用PostgreSQL的多版本并发控制 (MVCC) 模型来管理堆表的并发。通过MVCC，每一个查询都在它开始时的一个数据库快照上操作。在执行时，一个查询不能看到其他并发事务所作出的更改。这确保了一个查询看到的是数据库的一个一致的视图。读取行的查询不会被写入行的事务所阻塞。反过来，写入行的查询也不会被读取行的事务所阻塞。这使得Greenplum可以达到比使用锁来协调读写事务的传统数据库系统更高的并发度。

Note: 追加优化表使用一种不同于MVCC的并发控制模型管理。它们是为了“一次写，多次读”的应用而设计，这些应用从不或者很少会进行行级更新。

快照

MVCC模型依赖于系统能够管理数据行的多个版本的能力。一个查询其实是在该查询开始时的数据库快照上操作。快照就在一个语句或者事务开始时可见的行的集合。快照保证查询在其执行期间看到的是数据库的一个一致且合法的视图。

每一个事务都会被分配一个唯一的事务ID (XID)，它是一个增量式的32位值。当一个新事务开始时，它被分配下一个XID。没有被包裹在一个事务中的一个SQL语句会被当做一个单语句事务，即会给它隐式地加上BEGIN和COMMIT。这和一些数据库系统中的自动提交概念类似。

Note: Greenplum数据库只为涉及DDL或者DML操作的事务分配XID值，它们通常是唯一需要XID的事务。

当一个事务插入一行时，其XID会被保存在该行的 xmin 系统列中。当一个事务删除一行时，其XID会被保存在xmax 系统列中。更新一行被

指南

视为一次删除加上一次插入，因此XID会被保存在当前行的xmax中以及新插入行的xmin中。 xmin和 xmax列再加上事务完成状态就指定了一个事务的范围，行的这个版本对于其中的事务可见。一个事务可以看到所有小于xmin的事务的效果，这些事务确保已经被提交，但它无法看到任何大于等于xmax的事务的效果。

多语句事务还必须记录一个事务中哪个命令插入了一行 (cmin) 或者删除了一行 (cmax)，这样事务能够看到事务中先前的命令所作的更改。命令序列只在事务期间有意义，因此在一个事务开始时该序列被重置为0。

XID是数据库的一个性质。每一个Segment数据库都有其自己的XID序列，因此不能拿它和其他Segment数据库的XID进行比较。Master会使用一个集群范围的会话ID号来与Segment协调分布式事务，会话ID号被称为gp_session_id。Segment会维护一个分布式事务ID到其本地XID的映射。Master用两阶段提交协议在所有Segment之间协调分布式事务。如果一个事务在任一个Segment上失败，它将会在所有Segment上回滚。

用户可以用一个SELECT语句查看任意行的xmin、xmax、cmin和cmax列：

```
SELECT xmin, xmax, cmin, cmax, * FROM tablename;
```

因为用户是在Master上运行该SELECT命令，看到的XID都是分布式事务ID。如果用户能在一个Segment数据库上执行该命令，xmin和xmax值将是该Segment的本地XID。

Note: Greenplum数据库会将复制表的每行分发到所有节点上，所以每一行在所有节点上都是重复的。每个segment节点维护自己的xmin, xmax, cmin和cmax, 和gp_segment_id与ctid系统列一样。Greenplum数据库不允许用户访问复制表的这些列，因为他们在查询中没有单一、明确的值。

事务ID

MVCC模型使用事务ID (XID) 来判断哪些行在一个查询或者事务开始时是可见的。XID是一个32位值，因此在该值溢出并且回卷到零之前，一个数据库理论上可以执行超过四十亿个事务。不过，Greenplum数据库使用模 2³²的计算方式来使用XID，这允许事务ID回卷，就像时钟会在十二点回卷一样。对于任何给定的XID，有大约二十亿个过去的XID和二十亿个未来的XID。直到一行的一个版本存在大约二十亿个事务之前，这一套机制都有效，当这种情况发生

时那个版本就会突然变成一个新行。为了阻止这种情况的发生，Greenplum有一个被称为 FrozenXID的特殊XID，当把它和任何其他常规XID比较时它都是较老的哪一个。如果行中的xmin位于那二十亿个事务之中，就必须被替换为FrozenXID，这也是VACUUM命令执行的功能之一。

至少在每二十亿个事务时清理数据库可以阻止XID回卷。Greenplum数据库会监控事务ID并且在需要一次VACUUM操作时做出告警。

当不再可用的事务ID达到可观的比例并且事务ID回卷还没发生时，将会发出一个警告：

```
WARNING: database "database_name" must be vacuumed within  
number_of_transactions transactions
```

当该警告被发出时，就需要一次VACUUM操作。如果没有执行所需的VACUUM操作，当Greenplum数据库达到事务ID回卷发生之前的一个限制点时，它将会停止创建新事务来避免可能的数据丢失并且发出这样的错误：

```
FATAL: database is not accepting commands to avoid  
wraparound data loss in database "database_name"
```

关于从这种错误恢复的过程请见[从一次事务ID限制错误中恢复](#)。

服务器配置参数xid_warn_limit和 xid_stop_limit控制何时显示这些警告和错误。xid_warn_limit参数指定在xid_stop_limit之前多少个事务ID时发出警告。xid_stop_limit参数指定在回卷发生之前多少个事务ID时发出错误并且不允许创建新的事务。

事务隔离模式

SQL标准描述了数据库事务并发运行时可能发生的三种现象：

- 脏读 – 一个事务可能读到来自另一个并发事务的未提交数据。
- 不可重复读 – 在一个事务中两次读取同一行得到不同的结果，因为另一个并发事务在这个事务开始后提交了更改。
- 幻读 – 在同一个事务中两次执行同一个查询可能返回不同的行集合，因为另一个并发事务增加了行。

SQL标准定义了数据库系统需要支持的四种事务隔离模式，以及在事务并发执行时可以出现的现象。

Table 1. SQL事务隔离模式

级别	脏读	不可重复读	幻读
未提交读	可能	可能	可能
已提交读	不可能	可能	可能
可重复读	不可能	不可能	可能
可串行化	不可能	不可能	不可能

Greenplum数据库中未提交读和已提交读行为与SQL标准的已提交读一致。 Greenplum数据库中可串行化与可重复读隔离级别除了避免了幻读外，行为与SQL标准的已提交读一样。

已提交读和可重复读的区别是：在已提交读模式下，每个语句可以看到该语句执行前已提交的行；在可重复读模式下，语句只能看到该事务启动前提交的行。

在已提交读模式下，如果另一个并发执行的事务修改了行的值并提交，该行的值读两遍结果可能不同。已提交读允许幻读，一个查询执行两次拿到的结果集可能不同。

可重复读模式避免了非可重复读和幻读，虽然后者并不是标准中所必须的。一个尝试着修改其他并发事务修改过的数据的事务将被回滚。在可重复读隔离级别下执行事务的应用必须做好处理因为可串行化错误失败的事务。如果可重复读对于应用并不是必须的，建议使用已提交读模式。

可串行化模式，Greenplum数据库并不完全支持，可以保证一组事务在并行执行时得到的结果与串行执行的结果相同。在Greenplum数据库里指定可串行化的方式会退化到可重复读模式。MVCC快照隔离(SI)模式在没有昂贵的锁开销的前提下避免了脏读、不可重复读和幻读，但是仍然会存在在Greenplum数据库可串行化模式下的事务，无法做到真正的串行化。这些异常通常归咎于Greenplum数据库没有执行谓词锁定，即一个事务里的写会影响另一个并行事务里之前读的结果。

Note: PostgreSQL 9.1 可串行化隔离级别引入了一种新的可串行化快照隔离(SSI)模式，可以与SQL标准定义的可串行化隔离级别完全兼容。这个模式在Greenplum数据库中不可用。SSI监视可能导致序列化异常的条件的并发事务。当发现了潜在的串行化错误，一个事务被提交，其余的被回滚，并且必须重试。

Greenplum数据库并行运行的事务需要检查并识别可能并行更新相同数据的交互。通过使用显式表锁或要求冲突事务更新以表示冲突的虚拟行，可以防止发现的问题。

SQL语句SET TRANSACTION ISOLATION LEVEL可以设置当前事务的隔离级别。必须要在执行SELECT, INSERT, DELETE, UPDATE, or COPY语句前设置：

```
BEGIN;  
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;  
...  
COMMIT;
```

隔离模式也可以在BEGIN语句里指定：

```
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

可以通过设置*default_transaction_isolation*配置项来改变会话的默认隔离级别。

从表中移除过期的行

更新或者删除一行会在表中留下该行的一个过期版本。当一个过期的行不在被任何活跃事务引用时，它可以被移除从而腾出其所占用的空间进行重用。VACUUM命令会标记过期行所使用的空间可以被重用。

当表中的过期行累积后，为了容纳新的行就必须扩展磁盘文件。这样执行查询所需的磁盘I/O就会增加，从而性能受到影响。这种情况被称为膨胀，并且应该通过定期清理表来解决。

VACUUM命令（不带FULL）可以与其他查询并行运行。它会标记之前被过期行所占用的空间为空闲可用。如果剩余的空闲空间数量可观，它会把该页面加到该表的空闲空间映射中。当Greenplum数据库之后需要空间分配给新行时，它首先会参考该表的空闲空间映射以寻找有可用空间的页面。如果没有找到这样的页面，它会为该文件追加新的页面。

VACUUM（不带FULL）不会合并页面或者减小表在磁盘上的尺寸。它回收的空间只是放在空闲空间映射中表示可用。为了阻止磁盘文件大小增长，重要的是足够频繁地运行VACUUM。运行VACUUM的频率取决于表中更新和删除（插入只会增加新行）的频率。重度更新的表可能每天需要运行几次VACUUM来确保通过空闲空间映射能找到可用的空闲空间。在运行了一个更新或者删除大量行的事务之后运行VACUUM也非常重要。

VACUUM FULL命令会把表重写为没有过期行，并且将表减小到其最小尺寸。表中的每一页都会被检查，其中的可见行被移动到前面还没

有完全填满的页面中。空页面会被丢弃。该表会被一直锁住直到VACUUM FULL完成。相对于常规的VACUUM命令来说，它是一种非常昂贵的操作，可以用定期的清理来避免或者推迟这种操作。最好是在一个维护期来运行VACUUM FULL。VACUUM FULL的一种替代方案是用一个CREATE TABLE AS语句重新创建该表并且删除掉旧表。

用户可以运行VACUUM VERBOSE tablename来得到一份Segment上已移除的死亡行数量、受影响页面数以及有可用空闲空间页面数的报告。

查询pg_class系统表可以找出一个表在所有Segment上使用了多少页面。注意首先对该表执行ANALYZE确保得到的是准确的数据。

```
SELECT relname, relpages, reltuples FROM pg_class WHERE  
relname='tablename';
```

另一个有用的工具是gp_toolkit方案中的gp_bloat_diag视图，它通过比较一个表使用的实际页数和预期的页数来确定表膨胀。更多有关gp_bloat_diag的内容请见Greenplum数据库参考指南中的“gp_toolkit管理方案”。

- [管理事务ID的例子](#)

Parent topic: [Greenplum数据库概念](#)

Greenplum数据库® 6.0文档

- 管理员指南
- Greenplum数据库概念
- 关于Greenplum的架构
- 关于管理和监控工具
- 关于Greenplum数据库中的并发控制
- 关于并行数据装载
- 关于Greenplum数据库中的冗余和故障切换
- 关于Greenplum数据库中的数据库统计信息
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象
- 分布与倾斜
- 插入, 更新, 和删除数据
- 查询数据
- 使用外部数据
- 装载和卸载数据
- 性能管理

Greenplum database 5管理员
指南

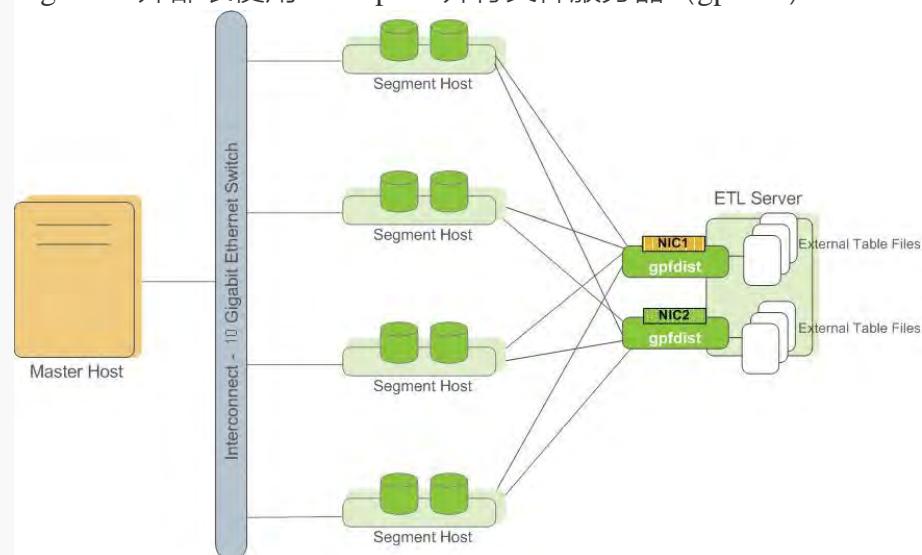
Greenplum database 4管理员

这一主题给出了对于Greenplum数据库数据装载特性的简短介绍。

在一个大型的、具有数T字节的数据仓库中，必须在一个相对较小的维护窗口内完成对大量数据的装载。Greenplum支持使用其外部表特性的快速、并行数据装载。管理员也可以用单行错误隔离模式装载外部表，这样可以把有问题的行过滤到一个单独的错误表而继续装载正确格式的行。对于一次装载操作，管理员可以指定一个错误阈值以控制Greenplum在碰到多少不正确的行后中止装载操作。

通过结合外部表以及Greenplum数据库的并行文件服务器（gpfldist），管理员可以在他们的Greenplum数据库系统上达到最大的并行度和装载带宽。

Figure 1. 外部表使用Greenplum并行文件服务器（gpfldist）



另一个Greenplum工具gupload能运行定义在一个YAML格式控制文件中的装载任务。在控制文件中描述了源数据位置、格式、转换要求、参与的主机、数据库目标以及其他说明，而gupload会执行装载。这允许用户描述一个复杂的任务，并且以可控、可重复的方式来执行它。

Parent topic: [Greenplum数据库概念](#)



冗余和故障切换

Greenplum数据库® 6.0文档

- 管理员指南
- Greenplum数据库概念
- 关于Greenplum的架构
- 关于管理和监控工具
- 关于Greenplum数据库中的并发控制
- 关于并行数据装载

关于Greenplum数据库中的冗余和故障切换

- 关于Greenplum数据库中的数据库统计信息
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象
- 分布与倾斜
- 插入, 更新, 和删除数据
- 查询数据
- 使用外部数据
- 装载和卸载数据
- 性能管理

Greenplum database 5管理员
指南

Greenplum database 4管理员

这一主题给出了Greenplum数据库高可用性特性的一个高层次的概述。

用户可以利用镜像组件部署一个不会出现单点故障的Greenplum数据库。下面的小节描述了镜像Greenplum系统主要组件的策略。更多关于Greenplum高可用特性的详细介绍, 请见 [Greenplum数据库高可用性概述](#)。

Important: 当Pivotal Greenplum数据库群集无法接受数据丢失时, 必须启用Master节点和Primary节点的Mirro, 以便集群可以被Pivotal所支持。如果没有镜像, 系统和数据可用性无法保证, 在这种情况下, Pivotal将尽最大努力恢复群集。

关于Segment镜像

部署Greenplum数据库系统时, 可以配置*Mirror*实例。Mirror节点可以使当Primary节点宕机的时候, 将数据库查询转移到备份节点上。Mirror节点通过将数据从主节点同步到从节点的事务日志复制进程保持同步。Mirror机制在生产环境中强烈建议开启, 并且是Pivotal支持所必须的。

作为最佳实践, 从节点实例必须和主节点部署到不同的机器以防止单机故障带来的影响。在虚拟化环境下, 二级(镜像)Segment必须总是位于一个与主要Segment不同的存储系统上。在一个为最大化可用性或者主机或者多个主要Segment失效时最小化性能衰退而设计的配置中, 镜像Segment可以被分布在集群中剩下的主机上。

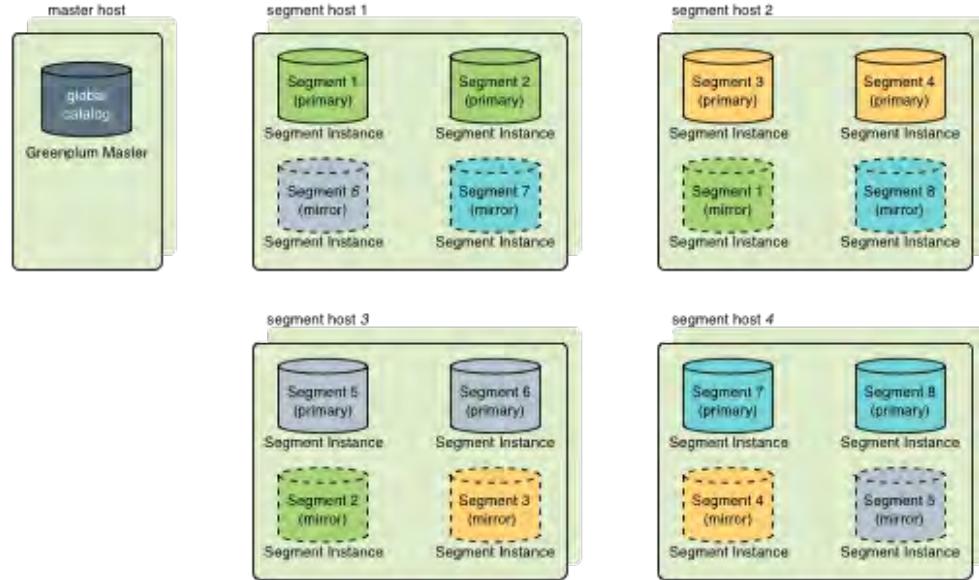
当用户在初始化或者扩展Greenplum系统时, 可使用两种标准的镜像配置。默认的配置被称为组镜像, 这种配置会把一个主机上的主要Segment的所有镜像都放置在集群中的其他一台主机上。还可以用一个命令行选项选择另一种标准配置散布镜像, 散布镜像会把每台主机的镜像散布到集群中剩余的主机上并且要求集群中的主机数量比每个主机上的主要Segment数量更多。

[Figure 1](#) 展示了当配置了散布镜像后, 表数据如何分布在Segment之间。

Figure 1. Greenplum数据库中的散布镜像



指南



Segment故障切换和恢复

当Greenplum数据库系统中启用了镜像时，系统会在主要Segment变成不可用后自动切换到镜像Segment。在一个Segment实例或者主机出问题的情况下，只要在剩余的活动Segment上的所有数据可用，Greenplum数据库系统就能保持可操作。

如果Master无法连接到一个Segment实例，它会在Greenplum数据库系统目录中标记该Segment实例为宕机并且把镜像Segment提升起来作为替代。失效的Segment实例将保持无法操作的状态直到管理员采取措施将它重新恢复上线。管理员可以在系统运行时恢复失效的Segment。恢复进程只会复制该Segment无法操作期间错过的更改。

如果用户没有启用镜像，当一个Segment实例变得无效时整个系统将会自动关闭。在继续操作之前，用户必须恢复所有失效的Segment。

关于Master镜像

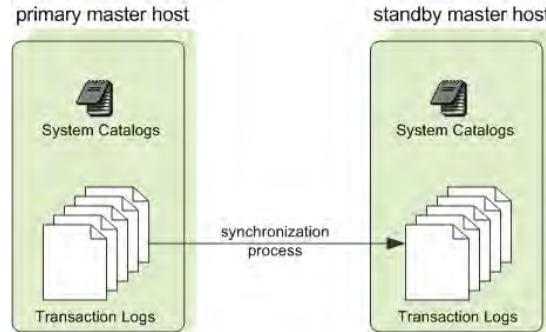
用户也可以选择性地在一台不同于Master节点的主机上部署一个Master实例的备份或者镜像。在主Master主机变得不可用时，一个备份Master主机将发挥温备的作用。后备Master利用事务日志复制进程保持与主Master同步，复制进程运行在后备Master上并且负责在主备Master主机之间同步数据。

如果主Master失效，日志复制进程会停止，并且后备Master会被激活以替代它的位置。这种切换不会自动发生，而是必须由外部触发。在激活后备Master时，将用已复制的日志来重构Master主机最后一次成

功提交时的状态。被激活的后备Master实际会变成Greenplum数据库的Master，它会在主端口（在Master主机和备份Master主机上必须被设置为相同）上接受客户端连接。

由于Master不包含任何用户数据，只有系统目录表需要在主节点和备份节点之间同步。当这些表被更新时，更改会被自动地复制到后备Master以确保与主Master的同步。

Figure 2. Greenplum数据库中的Master镜像



关于Interconnect冗余

Interconnect指的是Segment之间的进程间通信以及这种通信所依赖的网络设施。通过在用户的网络上部署双千兆以太网交换机以及到Greenplum数据库主机（Master和Segment）服务器的冗余千兆连接，用户可以得到非常可靠的Interconnect。出于性能的原因，我们推荐10-Gb以太网或者更快的网络。

Parent topic: [Greenplum数据库概念](#)

数据库统计信息

Greenplum数据库® 6.0文档

□ 管理员指南

Greenplum数据库中通过[ANALYZE](#)命令收集的统计信息的概述。

统计信息是描述数据库中数据的元数据。查询优化器需要最新的统计信息以便为查询选择最好的执行计划。例如，如果一个查询连接两个表并且其中的一个必须被广播到所有的Segment，优化器可以选择两个表中较小的那个来最小化网络流量。

优化器使用的统计信息由ANALYZE命令计算并且保存在系统目录中。有三种方式开始一个分析操作：

- 用户可以直接运行ANALYZE命令。
- 用户可以在数据库外部从命令行运行analyzedb管理工具。
- 当在没有统计信息的表上执行DML操作或者一个DML操作修改的行数超过指定阈值时，会触发一次自动的分析操作。

这些方法会在下面的小节中描述。VACUUM ANALYZE命令是另一种启动分析操作的方式，但是不鼓励使用它，因为清理和分析是两种不同目的的不同操作。

计算统计信息需要消耗时间和资源，因此Greenplum数据库通过在大型表的采样上计算统计信息来得到估计值。在大部分情况下，默认的设置提供了足以为查询生成正确执行计划的信息。如果产生的统计信息没有产生最优的查询执行计划，管理员可以调节配置参数通过提高样本尺寸或者系统目录中保存的统计信息粒度来产生更准确的统计信息。产生更准确的统计信息需要CPU和存储代价并且不一定能产生更好的计划，因此重点是查看解释计划并且测试查询性能来确保额外的统计信息代价能导致更好的查询性能。

Parent topic: [Greenplum数据库概念](#)

系统统计信息

表尺寸

查询规划器想要最小化执行查询所需的磁盘I/O和网络流量，它会使用必须被处理的行数以及查询必须访问的磁盘页面数的估计值。用于生成这些估计值的数据是pg_class系统表列reltuples和relpages，它们分别
VACUUM ANALYZE

包含上一次 或 命令运行时的行数和页面数。随着行被增加或删除，这些数字变得越来越不准确。不过，总是可以从操作系统拿到准确的磁盘页面计数，因此只要reltuples与relpages的比例不发生显著变化，优化器就能够产生对选择正确的查询执行计划足够准确的行数估计。

当reltuples列与SELECT COUNT(*)返回的行计数显著不同时，应该执行一次分析来更新统计信息。

当一个REINDEX命令完成了重建一个索引时，relpages和reltuples列被设置为零。应该在基表上运行ANALYZE命令以更新这些列。

pg_statistic系统表和pg_stats视图

pg_statistic系统表保持在每个数据库表上最后一次ANALYZE操作的结果。其中为每一个表的每一列都有一行。行有下面的列：

starelid

该列所属的表或索引的对象ID。

staattnum

所描述列的编号，从1开始。

stainherit

如果值是true，统计信息不仅包括指定的relation，还包括继承的子列。

stanullfrac

列中为空的项的分数值。

stawidth

非空项的平均存储宽度，单位是字节。

stadistinct

一个正数，它是列中可区分值的数量估计。这个数字预计并不会随着行数变化。一个负值是可区分值数量除以行数，也就是该列中有可区分值的行的比例取负值。当可区分值数量随行数增加时使用这种形式。例如，一个唯一列的n_distinct值为-1.0。平均宽度超过1024的列被认为是唯一的。

stakind *N*

一个代码数字，它表示存储在pg_statistic行第*N*个槽中的统计信息类型。

staop *N*

用来得到第*N*个槽中统计信息的操作符。例如，一个直方图槽会显示<操作符，它定义数据的排序顺序。

stanumbers *N*

float4数组，包含第*N*个槽的合适类型的数字统计信息，如果槽类型不涉及数字值则为NULL。

stavalues *N*

第N个槽的合适类型的列数据值，如果该槽类型不存储任何数据值则为NULL。每一个数组的元素值实际是指定列的数据类型，因此没有办法比使用anyarray更具体地定义这些列的类型。

为一个列收集的统计信息随着不同数据类型变化，因此pg_statistic表中存储适合于四个槽中数据类型的统计信息，每个槽由四个列组成。例如，第一个槽通常包含一列的最常见值，它由列stakind1、staop1、stanumbers1和stavalue1组成。

stakindN列每个都包含一个数字代码，用于描述存储在其插槽中的统计信息的类型。从1到99的stakind代码编号保留给核心PostgreSQL数据类型。Greenplum数据库使用代码编号1, 2, 3, 4, 5, 和99。0的意思是槽未使用。下面的表格描述了为三种代码存储的统计信息类型。

Table 1. pg_statistic中"槽"的内容

stakind代码	描述
1	<p>最常见值 (MCV) 槽</p> <ul style="list-style-type: none"> staop包含"="操作符的对象ID，它被用来决定值是否相同。 stavalue包含一个数组，其中是该列中出现的K个最常见非空值。 stanumbers包含stavalue数组中值的频度 (总行数的分数)。 <p>值按照频度降序排序。因为数组是可变尺寸的，K可以由统计收集器选择。要被加入到stavalue数组中，值必须出现超过一次。唯一列没有MCV槽。</p>
2	<p>直方图槽 – 描述标量数据的分布。</p> <ul style="list-style-type: none"> staop是"<"操作符的对象ID，它描述排序顺序。 stavalue包含M (其中M>=2) 个非空值，它们将非空的列数据值划分成M-1个群体数量大致相等的箱子。第一个stavalue项是最小值而最后最后一个是最大小值。 stanumbers没有被使用且应该为NULL。 <p>如果也提供了一个最常见值的槽，那么该直方图描述的是将MCV数组中列出的值移除后的数据分布 (在技术上的说法是一个压缩直方图)。这允许更精确地表示一个有一些非常常见值的列的分布。在一个只有一些可区分值的列中，有可能MCV列表就描述了整个数据群体，在这种情况下直方图缩小为空并且应该被省略。</p>

3	<p>相关关系槽 – 描述表元组物理顺序和列数据值顺序之间的相关关系。</p> <ul style="list-style-type: none"> staop是"<"操作符的对象ID。与直方图一样，理论上可能会出现多于一项。 stavalue未被使用并且应该为空。 stanumbers包含一个单项，它是数据值的序列和它们实际元组位置序列之间的相关系数。系数范围从+1到-1。
4	<p>最常见的元素插槽 - 类似于最常见值 (MCV) 插槽，除了它存储列值的最常见非空元素。当列数据类型是数组或具有可识别元素的其他类型（例如，tsvector）时，这非常有用。</p> <ul style="list-style-type: none"> staop包含适合于元素类型的相等运算符。 stavalue包含最常见的元素值。 stanumbers包含共同的元素频率。 <p>频率测量为元素值出现的非空行的分数，而不是所有行的频率。此外，值将按元素类型的默认顺序排序（以支持特定值的二分查找）。由于这会将最小和最大频率放在不稳定点上，因此有两个额外的stanumbers成员可以保存最小和最大频率的副本。可选地，可以存在保持空元素的频率的第三额外成员（频率以相同的术语表示：包含至少一个空元素的非空行的分数）。如果省略此成员，则假定该列不包含NULL元素。</p> <p>Note: 注意：对于tsvector列，stavalue元素的类型为text，即使它们在tsvector中的表示不完全是文本。</p>
5	<p>不同的元素计数直方图槽 - 描述了数组类型列的每一行中存在的不同元素值的数量的分布。仅考虑非空行，并且仅考虑非空元素。</p> <ul style="list-style-type: none"> staop包含适合于元素类型的相等运算符。 <p><</p>

本号

Greenplum数据库® 6.0文档

- 管理员指南
- Greenplum数据库概念
- 管理一个Greenplum系统

关于Greenplum数据库 发布版本号

启动和停
止Greenplum数据库

- 访问数据库
- 配置Greenplum数据库
系统
- 启用压缩
- 启用高可用和数据持久
化特征
- 备份和恢复数据库
- 扩容Greenplum系统

监控Greenplum系统

日常系统维护任务

Recommended
Monitoring and
Maintenance Tasks

- 管理Greenplum数据库访问
- 定义数据库对象
- 分布与倾斜

插入, 更新, 和删除数据

Greenplum的版本号和其组织方式的改变，界定了从一个版本到下一个版本做了那些修改。

Greenplum数据库的发布版本号采用格式x.y.z, 含义分别为:

- x表明主版本号
- y表明小版本号
- z表明补丁版本号

具有相同主版本号的Greenplum数据库保证在该版本下具有向后兼容性。当Greenplum数据库的元数据目录修改或不匹配的特性改变出现或新特性被引入时，才会增加主版本号。之前版本中被注释掉的功能在新的主版本发布时可能被移除。

当向后兼容的新特性被引入或曾经的特性被注释时，会在某一主版本的基础上增加小版本号。 (之前注释的功能不会在该小版本中被移除)

当出现向后兼容的问题修复时，Greenplum数据库会在某一小版本号的基础上增加补丁版本号。

Parent topic: [管理一个Greenplum系统](#)



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库
发布版本号启动和停
止Greenplum数据库

□ 访问数据库

□ 配置Greenplum数据库
系统

启用压缩

□ 启用高可用和数据持久
化特征

□ 备份和恢复数据库

□ 扩容Greenplum系统

监控Greenplum系统

日常系统维护任务

Recommended
Monitoring and
Maintenance Tasks

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

在一个Greenplum数据库管理系统中, 所有主机上的数据库实例 (Master和所有的Segment) 一起被启动或者停止, 启停操作统一由Master实例发起, 它们步调一致, 在外界看来是一个完整的数据库管理系统。

由于一个Greenplum数据库系统分布在很多机器上, 启动和停止一个Greenplum数据库系统的过程与普通PostgreSQL数据库的启动和停止过程不同。

分别使用`gpstart` 和 `gpstop` 工具来启动和停止Greenplum数据库。这些工具位于Greenplum数据库Master主机的 `$GPHOME/bin` 目录下。

Important: 不要采用`kill`命令来直接杀掉任何后台Postgres进程。 可以使用数据库内命令`pg_cancel_backend()`来完成此操作。

采用 `kill -9` 或 `kill -11` 可能引发数据库损坏并且妨碍根据目前数据库的表现进行问题根源分析。

有关`gpstart` 和 `gpstop`的详细信息, 请参考文档 *Greenplum数据库工具指南*。

Parent topic: [管理一个Greenplum系统](#)

启动Greenplum数据库

在Master实例上, 通过执行`gpstart`可以启动一个初始化好的Greenplum数据库系统。

可以使用`gpstart`工具来启动一个已经由`gpinitcluster`工具初始化好的Greenplum数据库系统, 前提是该系统已经被`gpstop`工具停止。`gpstart` 通过启动整个Greenplum数据库集群中的所有Postgres数据库实例来启动Greenplum数据库。`gpstart` 会精心安排这一过程并且以并行的方式执行它。

在Master主机上运行`gpstart`以启动一个Greenplum数据库:

```
$ gpstart
```



重启Greenplum数据库

□ 查询数据

停止Greenplum数据库系统然后重新启动它。

执行gpstop工具并带有-r选项时，会停止Greenplum数据库并在数据库完全关闭后重新启动它。

要重启Greenplum数据库，在Master主机上输入下面的命令：

```
$ gpstop -r
```

仅重新载入配置文件更改

重新载入对Greenplum数据库配置文件的更改而不中断系统。

gpstop 工具可以在不中断服务的前提下重新载入对 pg_hba.conf 配置文件和 Master 上 postgresql.conf、pg_hba.conf 文件中的运行参数进行更改。活动会话将会在它们重新连接到数据库时使用这些更新。很多服务器配置参数需要完全重启系统(gpstop -r)才能激活。有关服务器配置参数的信息请见 *Greenplum Database Reference Guide*。

使用gpstop工具重新载入配置文件更改而不关闭系统：

```
$ gpstop -u
```

以维护模式启动Master

只启动Master来执行维护或者管理任务而不影响Segment上的数据。

例如，可以用维护模式连接到一个只在Master实例上的数据库并且编辑系统目录设置。更多有关系统目录表的信息请见 *Greenplum Database Reference Guide*。

1. 使用-m模式运行gpstart：

```
$ gpstart -m
```

2. 以维护模式连接到Master进行目录维护。例如：

```
$ PGOPTIONS=' -c gp_session_role=utility' psql postgres
```

3. 在完成管理任务后，停止处于维护模式的Master。然后以生产模式重启它。

```
$ gpstop -mr
```

Warning:

对维护模式连接的不当使用可能会导致不一致的系统状态。只有技术支持才应该执行这一操作。

停止Greenplum数据库

`gpstop`工具可以停止或者重启Greenplum数据库系统，它总是运行在Master主机上。当被激活时，`gpstop`会停止系统中所有的`postgres`进程，包括Master和所有的Segment实例。`gpstop`工具使用一种默认的最多64个并行工作线程的方式来关闭构成整个Greenplum数据库集群的Postgres实例。系统在关闭之前会等待所有的活动事务完成。要立即停止Greenplum数据库，可以使用快速模式。

- 要停止Greenplum数据库：

```
$ gpstop
```

- 要以快速模式停止Greenplum数据库：

```
$ gpstop -M fast
```

默认情况下，如果有任何客户端连接存在，就不允许关闭Greenplum数据库。使用`-M fast`选项可以在关闭前回滚所有正在进行中的事务并且中断所有连接。

停止客户端进程：

Greenplum数据库会为每一个客户端连接唤起一个后台进程。具有SUPERUSER权限的Greenplum用户可以取消或直接终止这些客户端后台进程。

使用`pg_cancel_backend()`函数可以取消某个活动查询或队列中的后台进程。使用`pg_terminate_backend()`可以直接终端访问Greenplum数据库的客户端连接。

`pg_cancel_backend()`函数有两种表现形式：

- `pg_cancel_backend(pid int4)`

```
pg_cancel_backend( pid int4, msg text )
```

`pg_terminate_backend()`函数也有两种同样地表现形式：

- `pg_terminate_backend(pid int4)`
- `pg_terminate_backend(pid int4, msg text)`

如果提供了`msg`信息, Greenplum数据库会将取消信息的`msg`内容一并返回给客户端。`msg`最大为128字节; 任何超出该长度的信息都会被Greenplum数据库直接截断。

如果`pg_cancel_backend()`和`pg_terminate_backend()`函数执行成功, 会返回状态`true`, 相反的会返回`false`。

要取消或中断后台进程, 必须先查出要处理的后台进程ID。进程ID可以从`pg_stat_activity`视图的`procpid`列得到。例如, 如果要看所有运行中和等待中查询的进程信息, 可以执行:

```
=# SELECT usename, procpid, waiting, current_query, datname
   FROM pg_stat_activity;
```

上例语句部分查询结果如下:

usename	procpid	waiting	current_query	datname
sammy	31861	f	<IDLE> in transaction	testdb
billy	31905	t	SELECT * FROM topten;	testdb

我们可以从查询结果中找到对应查询或客户端连接的进程ID(`procpid`)。

例如, 下面语句会取消上面输出中的等待查询, 并将"Admin canceled long-running query."信息反馈给客户端。

```
=# SELECT pg_cancel_backend(31905 , 'Admin canceled long-
running query.');
ERROR: canceling statement due to user request: "Admin
canceled long-running query."
```

Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统

关于Greenplum数据库
发布版本号

启动和停
止Greenplum数据库

访问数据库

- 配置Greenplum数据库
系统

启用压缩

- 启用高可用和数据持久
化特征
- 备份和恢复数据库
- 扩容Greenplum系统

监控Greenplum系统

日常系统维护任务

Recommended
Monitoring and
Maintenance Tasks

- 管理Greenplum数据库访问
- 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

这个主题描述了几种可用来连接Greenplum数据库的客户端工具以及如何建立一个数据库会话。

- 建立一个数据库会话
- 支持的客户端应用
- [Greenplum数据库客户端应用](#)
- [用psql连接](#)
- [使用PgBouncer连接池](#)
- [数据库应用接口](#)
- [连接问题的发现及解决](#)

Parent topic: [管理一个Greenplum系统](#)



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库
发布版本号

启动和停
止Greenplum数据库

□ 访问数据库

□ 配置Greenplum数据库
系统

启用压缩

□ 启用高可用和数据持久
化特征

□ 备份和恢复数据库

□ 扩容Greenplum系统

监控Greenplum系统

日常系统维护任务

Recommended
Monitoring and
Maintenance Tasks

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

服务器配置参数会影响Greenplum数据库的行为。它们是PostgreSQL的“大统一配置”系统的一部分，因此它们有时也被称为“GUC”。大部分的Greenplum数据库服务器配置参数和PostgreSQL的配置参数相同，但是也有一些是 Greenplum所特有的。

- [关于Greenplum数据库的Master参数和本地参数](#)
- [设置配置参数](#)
- [查看服务器配置参数设置](#)
- [配置参数种类](#)

Parent topic: [管理一个Greenplum系统](#)



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库
发布版本号

启动和停
止Greenplum数据库

□ 访问数据库

□ 配置Greenplum数据库
系统

启用压缩

□ 启用高可用和数据持久
化特征

□ 备份和恢复数据库

□ 扩容Greenplum系统

监控Greenplum系统

日常系统维护任务

Recommended
Monitoring and
Maintenance Tasks

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

可以利用Greenplum数据库自身的特性或工具来配置启用数据压缩。
压缩能减少磁盘空间使用并提高系统访问的I/O, 但是在压缩和解压缩
数据时会带来一些额外的负载。

可以采用一下特性或工具来让Greenplum数据库支持数据压缩, 具体见
以下特定文档。

- 追加优化表支持压缩表数据, 请见[CREATE TABLE](#).
- 采用用户自定义数据类型来压缩数据, 请见[CREATE TYPE](#).
- 通过外部表协议[gpfdist](#) ([gpfdists](#)), [s3](#), and [pxf](#) 访问外部数据
时支持压缩。请见[CREATE EXTERNAL TABLE](#).
- Workfiles (查询所用内存超出系统分配内存时临时溢出的文件) 压
缩。请见服务器配置参数[gp_workfile_compression](#).
- Greenplum数据库工具[gpbackup](#), [gprestore](#), [gupload](#), and
[gplogfilter](#) 支持压缩

针对一些压缩算法 (例如zlib), Greenplum要求软件包已经安装在主
机系统中。. 另外一些压缩算法 (例如zstd) 的支持则依赖于编
译Greenplum数据库时是否配置了该选项。

Parent topic: [管理一个Greenplum系统](#)



特征

Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统

- 关于Greenplum数据库发布版本号

- 启动和停止Greenplum数据库

- 访问数据库
- 配置Greenplum数据库系统

- 启用压缩

- 启用高可用和数据持久化特征

- 备份和恢复数据库
- 扩容Greenplum系统

监控Greenplum系统

日常系统维护任务

Recommended Monitoring and Maintenance Tasks

- 管理Greenplum数据库访问
- 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

Greenplum数据库的容错和高可用特征可以通过配置启用

Important: 当Greenplum数据库 集群不允许数据丢失时, master和segment镜像功能必须启用。没有镜像就不能保证系统和数据的 高可用, 此时Pivotal会尽最大的努力恢复故障集群。master和segment镜像详情, 请见[关于冗余和故障转移](#)。

关于启用高可用的工具详情, 请见*Greenplum*数据库工具指南。

- [Greenplum数据库高可用性概述](#)
- [在Greenplum数据库中启用镜像](#)
- [检测故障的Segment](#)
- [恢复故障Segment](#)
- [恢复故障的Master](#)

Parent topic: [管理一个Greenplum系统](#)



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库
发布版本号

启动和停
止Greenplum数据库

□ 访问数据库

□ 配置Greenplum数据库
系统

启用压缩

□ 启用高可用和数据持久
化特征

□ 备份和恢复数据库

□ 扩容Greenplum系统

监控Greenplum系统

日常系统维护任务

Recommended
Monitoring and
Maintenance Tasks

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

这个主题描述如何使用Greenplum的备份和恢复特性。

定期执行备份能确保在数据损坏或者系统失效发生时能恢复数据或者重建Greenplum数据库系统。用户还可以使用备份从一个Greenplum数据库系统迁移数据到另一个。

- [备份和恢复概述](#)
- [使用gpbackup和gprestore并行备份](#)

Parent topic: [管理一个Greenplum系统](#)



Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统
 - 关于Greenplum数据库
发布版本号
 - 启动和停
止Greenplum数据库
 - 访问数据库
 - 配置Greenplum数据库
系统
 - 启用压缩
 - 启用高可用和数据持久
化特征
 - 备份和恢复数据库
 - 扩容Greenplum系统**
 - 监控Greenplum系统
 - 日常系统维护任务
 - Recommended
Monitoring and
Maintenance Tasks
 - 管理Greenplum数据库访问
 - 定义数据库对象
 - 分布与倾斜
 - 插入、更新、和删除数据

为了放大性能和存储能力，通过向集群增加主机来扩容用户的Greenplum系统。

随着额外的数据被收集以及现有数据的保留时间增加，数据仓库会随着时间而长大。有时，有必要增加数据库能力来联合不同的数据仓库到一个数据库中。也可能需要额外的计算能力（CPU）来适应新增加的分析项目。尽管在系统被初始定义时就留出增长的空间是最为明智的，但是通常不太可能在提前太多在资源上投资。因此，用户应该寄希望于定期地执行一次数据库扩容项目。

由于Greenplum的MPP架构，在用户增加资源到系统时，得到的容量和性能就好像该系统一开始就用增加后的资源实现一样。和要求大量停机时间来转储和恢复数据的数据仓库系统不同，扩容一个Greenplum数据库系统是一种最小化停机时间的分阶段处理。在数据被重新分布时，常规和特别负载可以继续并且事务一致性也能被维护。管理员可以安排分布活动以适合正在进行的操作并且可以按需暂停和继续。表可以被排名，这样数据集可以以一种优先序列的方式被重新分布，从而确保关键性的负载能很快从扩容后的能力受益，或者释放所需的磁盘空间来重新分布非常大的表。

扩容处理使用标准的Greenplum数据库操作，因此它是透明的并且管理员易于排查错误。Segment镜像和任何复制机制就地保持活动，因此容错性没有打折扣并且灾难恢复措施也保持有效。

- **系统扩容概述**

用户可以最小化停机时间，通过增加节点实例和节点主机来扩容Greenplum数据库。

- **规划Greenplum系统扩容**

细心的规划将帮助确保一个成功的Greenplum扩容项目。

- **准备并增加节点**

验证用户的新节点已经准备好整合到现有的Greenplum系统中。

- **初始化新节点**

使用`gpexpand`工具创建并初始化新节点实例，并创建扩容schema。

- **重分布表**

重新分布表让现有数据在新扩容后的集群上得以平衡。

- **移除扩容Schema**

要在扩容Greenplum集群后进行清理，需要移除扩容Schema。

Parent topic: [管理一个Greenplum系统](#)

□ 查询数据

Greenplum数据库® 6.0文档

[管理员指南](#)

[Greenplum数据库概念](#)

[管理一个Greenplum系统](#)

关于Greenplum数据库发布版本号

启动和停止Greenplum数据库

[访问数据库](#)

[配置Greenplum数据库系统](#)

启用压缩

[启用高可用和数据持久化特征](#)

[备份和恢复数据库](#)

[扩容Greenplum系统](#)

监控Greenplum系统

日常系统维护任务

Recommended Monitoring and Maintenance Tasks

[管理Greenplum数据库访问](#)

[定义数据库对象](#)

分布与倾斜

插入、更新、和删除数据

[查询数据](#)

[使用外部数据](#)

[装载和卸载数据](#)

可以用系统中包含的各种工具以及附加组件来监控一个Greenplum数据库系统。

观察Greenplum数据库系统的性能有助于管理员理解系统的行为、计划工作流以及故障排查问题。本章讨论用于监控数据库性能和活动的工具。

另外，记得回顾[Recommended Monitoring and Maintenance Tasks](#) 中编写脚本 监控活动来快速检测系统中问题的有关内容。

Parent topic: [管理一个Greenplum系统](#)

监控数据库活动和性能

Greenplum数据库包含一个可选的系统监控和管理数据库`gpperfmon`，管理员可以选择启用它。`gpperfmon_install`命令行工具创建`gpperfmon`数据库并启用数据收集代理来存储查询和系统矩阵信息到该数据库。管理员可以查询`gpperfmon`数据库中的矩阵信息。更多`gpperfmon`的信息，请见[Greenplum Database Reference Guide](#)。

监控系统状态

作为一个Greenplum数据库管理员，必须监控系统的问题事件，例如一个Segment宕机或者一台Segment主机磁盘空间耗尽。下面的主题描述如何监控一个Greenplum数据库系统的健康状况以及检查一个Greenplum数据库系统的特定状态信息。

- [检查系统状态](#)
- [检查磁盘空间使用](#)
- [检查数据分布倾斜](#)
- [查看数据库对象的元数据信息](#)
- [查看会话内存使用信息](#)
- [查看查询工作文件使用信息](#)

检查系统状态

一个Greenplum数据库系统由横跨多台机器的多个PostgreSQL实例（Master和Segment）构成。要监控一个Greenplum数据库系统，需要了解整个系统的信息以及个体实例的状态信息。`gpstate`工具提供有关一个Greenplum数据库系统的状态信息。

查看Master和Segment的状态及配置

默认的`gpstate`行为是检查Segment实例并且显示可用和失效Segment的一个简短状态。例如，要快速查看Greenplum数据库系统的状态：

```
$ gpstate
```

要查看Greenplum数据库阵列配置更详细的信息，使用带有`-s`选项的`gpstate`：

```
$ gpstate -s
```

查看镜像配置和状态

如果在使用镜像作为数据冗余，用户可能想要看看系统中的镜像Segment实例列表、它们当前的同步状态以及镜像和主Segment之间的映射。例如，要查看一个系统中的镜像Segment和它

们的状态: 5

```
$ gpstate -m
```

要查看主Segment到镜像Segment的映射:

```
$ gpstate -c
```

要查看后备Master镜像的状态:

```
$ gpstate -f
```

检查磁盘空间使用

一个数据库管理员最重要的监控任务是确保Master和Segment数据目录所在的文件系统的使用率不会超过 70%的。完全占满的数据磁盘不会导致数据损坏，但是可能会妨碍数据库的正常操作。如果磁盘占用得太满，可能会导致数据库服务器关闭。

可以使用`gp_toolkit`管理模式中的`gp_disk_free`外部表 来检查Segment主机文件系统中的剩余空闲空间 (以KB为计量单位) 。例如:

```
=# SELECT * FROM gp_toolkit.gp_disk_free
 ORDER BY dfsegment;
```

检查分布式数据库和表的大小

`gp_toolkit`管理模式包含几个可以用来判断Greenplum数据库的分布式数据库、 模式、 表或索引磁盘空间使用的视图。

用于检查数据库对象尺寸和磁盘空间的视图列表，请见 *Greenplum Database Reference Guide*.

查看一个数据库的磁盘空间使用情况

要查看一个数据库的总大小 (以字节计)， 使用`gp_toolkit`管理模式中的`gp_size_of_database`视图。例如:

```
=> SELECT * FROM gp_toolkit.gp_size_of_database
 ORDER BY sodddatname;
```

查看一个表的磁盘空间使用情况

`gp_toolkit`管理模式包含几个检查表大小的视图。表大小视图根据对象ID (而不是名称) 列出表。要根据一个表的名称检查其尺寸，必须在`pg_class`表中查找关系名称 (`relname`) 。例如:

```
=> SELECT relname AS name, sotdsiz AS size, sotdtoastsize
 AS toast, sotdadditionals AS other
 FROM gp_toolkit.gp_size_of_table_disk as sotd, pg_class
 WHERE sotd.sotdoid=pg_class.oid ORDER BY relname;
```

可用的表大小视图的列表请见 *Greenplum Database Reference Guide*.

查看索引的磁盘空间使用情况

*gp_toolkit*管理模式包含几个用于检查索引大小的视图。要查看一个表上所有索引的总大小，使用*gp_size_of_all_table_indexes*视图。要查看一个特定索引的大小，使用*gp_size_of_index*视图。该索引大小视图根据对象ID（而不是名称）列出表和索引。要根据一个索引的名称查看其尺寸，必须在*pg_class*表中查找关系名称(*relname*)。例如：

```
=> SELECT soisize, relname as indexname
   FROM pg_class, gp_toolkit.gp_size_of_index
  WHERE pg_class.oid=gp_size_of_index.soioid
    AND pg_class.relkind='i';
```

检查数据分布倾斜

Greenplum数据库中所有的表都是分布式的，意味着它们的数据被按规则划分到系统中的所有Segment上。不均匀分布的数据可能会削弱查询处理性能。一个表的分布策略在表创建时被确定。有关选择表分布策略的信息，请见下列主题：

- [查看一个表的分布键](#)
- [查看数据分布](#)
- [检查查询过程倾斜](#)

*gp_toolkit*管理模式还包含一些用于检查表上数据分布倾斜的视图。有关如何检查非均匀数据分布的信息，请见[Greenplum Database Reference Guide](#).

查看一个表的分布键

要查看一个表中被用作数据分布键的列，可以使用`psql`中的`\d+`元命令来检查表的定义。例如：

```
=# \d+ sales
           Table "retail.sales"
 Column     |      Type       | Modifiers | Description
-----+-----+-----+-----+
 sale_id   | integer        |           |
 amt       | float          |           |
 date      | date           |           |
 Has OIDs: no
 Distributed by: (sale_id)
```

当我们创建复制表时，Greenplum数据库会在每个Segment上都存储一份完整的表数据。复制表没有分布键。`\d+`元命令会展示分布表的分布键，复制表展示状态为Distributed Replicated。

查看数据分布

要查看一个表中行的数据分布（每个Segment上的行数），可以运行一个这样的查询：

```
=# SELECT gp_segment_id, count(*)
   FROM table_name GROUP BY gp_segment_id;
```

如果所有的Segment都有大致相同的行数，一个表就可以被认为是分布均匀的。

Note:

如果在复制表上运行该查询会执行失败，因为Greenplum数据库不允许用户查询复制表的*gp_segment_id*系统列数据。由于每个Segment上都有一份完整的表数据，复制表必然是均匀的。

检查查询过程倾斜

Segment

当一个查询被执行时，所有的应该具有等量的负载来保证最好的性能。如果发现了一个执行性能低下的查询，可能需要使用EXPLAIN命令进行深入研究。有关使用EXPLAIN命令和查询分析的信息，请见[查询分析](#)。

如果表的数据分布策略与查询谓词没有很好地匹配，查询执行负载可能会倾斜。要检查执行倾斜，可以运行一个这样的查询：

```
=# SELECT gp_segment_id, count(*) FROM table_name
   WHERE column='value' GROUP BY gp_segment_id;
```

这将显示对于给定的WHERE谓词，Segment会返回的行数。

如[查看数据分布](#)所说的，该查询在复制表上运行时也会报错，因为在复制表上查询的gp_segment_id列不具有参考价值。

避免极度倾斜警告

当执行一个使用哈希连接操作的查询时，可能会收到下面的警告消息：

```
Extreme skew in the innerside of Hashjoin
```

当一个哈希连接操作符的输入倾斜时，就会发生这种情况。它不会阻碍查询成功完成。可以按照这些步骤来避免计划中的倾斜：

1. 确保所有的事实表都被分析过。
2. 验证该查询用到的任何已填充临时表都被分析过。
3. 查看该查询的EXPLAIN ANALYZE计划，并且在其中查找以下信息：
 - 如果有带多列过滤的扫描产生超过预估的行数，则
将gp_selectivity_damping_factor服务器配置参数的值设置为当前值的2倍以上并且重新测试该查询。
 - 如果在连接一个相对较小（小于5000行）的单一事实表时发生倾斜，
将gp_segments_for_planner服务器配置参数设置为1并且重新测试该查询。
4. 检查应用于该查询的过滤属性是否匹配基表的分布键。如果过滤属性和分布键相同，考虑用不同的分布键重新分布一些基表。
5. 检查连接键的基数。如果基数较低，尝试用不同的连接列或者表上额外的过滤属性来重写该查询以降低行数。这些更改可能会改变查询的语义。

查看数据库对象的元数据信息

Greenplum数据库在其系统目录中跟踪各种有关存储在数据库中对象（例如表、视图、索引等）和全局对象（例如角色和表空间）的元数据信息。

查看最后一个执行的操作

可以使用系统视图pg_stat_operations和pg_stat_partition_operations查看在一个对象（例如一个表）上执行的动作。例如，要查看在一个表上执行的动作，比如它何时被创建以及它上一次是什么时候被清理和分析：

```
=> SELECT schemaname as schema, objname as table,
       username as role, actionname as action,
       subtype as type, statime as time
      FROM pg_stat_operations
     WHERE objname='cust';
schema | table | role | action | type | time
-----+-----+-----+-----+-----+-----
 sales | cust  | main | CREATE | TABLE | 2016-02-09 18:10:07.867977-08
 sales | cust  | main | VACUUM |       | 2016-02-10 13:32:39.068219-08
 sales | cust  | main | ANALYZE|       | 2016-02-25 16:07:01.157168-08
(3 rows)
```

查看一个对象的定义

要查看一个对象（例如表或者视图）的定义，在`psql`中可以使用`\d+`元命令。例如，要查看一个表的定义：

```
=> \d+ mytable
```

查看会话内存使用信息

可以创建并且使用`session_level_memory_consumption`视图来查看正在Greenplum数据库上运行查询的会话的当前内存利用信息。该视图包含会话信息以及该会话连接到的数据库、该会话当前运行的查询和会话处理所消耗的内存等信息。

- [创建`session_level_memory_consumption`视图](#)
- [`session_level_memory_consumption`视图](#)

创建`session_level_memory_consumption`视图

要在Greenplum数据库中创建`session_level_memory_consumption`视图，为每一个数据库运行一次扩展创建语句`CREATE EXTENSION gp_internal_tools;`。例如，要在数据库`testdb`中安装该视图，可使用这个命令：

```
$ psql -d testdb -c "CREATE EXTENSION gp_internal_tools;"
```

`session_level_memory_consumption`视图

`session_level_memory_consumption`视图提供有关正在运行SQL查询的会话的内存消耗以及闲置时间的信息。

当基于资源队列的资源管理方式启用时，在该视图中，列`is_runaway`表示是否Greenplum数据库认为该会话是一个失控会话，这种判断基于该会话的查询的vmem内存消耗来做出。在资源队列管理模式下，当查询消耗过多内存时，Greenplum数据库认为该会话处于失控状态。Greenplum数据库的服务器配置参数`runaway_detector_activation_percent`控制Greenplum数据库什么时候会认为一个会话是失控会话。

在该视图中，当基于组的资源管理方式启用时，列`is_runaway`, `runaway_vmem_mb`和`runaway_command_cnt`功能失效。

Table 1. `session_level_memory_consumption`

列	类型	引用	描述
<code>datname</code>	<code>name</code>		该会话连接到的数据库名。
<code>sess_id</code>	<code>integer</code>		会话ID。
<code>username</code>	<code>name</code>		会话用户的用户名。
<code>current_query</code>	<code>text</code>		该会话正在运行的当前SQL查询。
<code>segid</code>	<code>integer</code>		Segment ID。
<code>vmem_mb</code>	<code>integer</code>		该会话的总vmem内存使用，以MB计。
<code>is_runaway</code>	<code>boolean</code>		会话被标记为在Segment上失控。
<code>qe_count</code>	<code>integer</code>		该会话的查询处理数量。

active_qe_count	integer		该会话的活动查询处理数量。
dirty_qe_count	integer		还没有释放其内存的查询处理的数量。 对于没有运行的会话该值为-1。
runaway_vmem_mb	integer		当会话被标记为失控会话时消耗的vmem内存量。
runaway_command_cnt	integer		当会话被标记为失控会话时的命令计数。
idle_start	timestamptz		这个会话中上一次一个查询处理变成空闲的时间。

查看查询工作文件使用信息

Greenplum数据库管理方案 *gp_toolkit* 包含显示 Greenplum 数据库工作文件信息的视图。如果没有足够的内存让查询完全在内存中执行，Greenplum 数据库会在磁盘上创建工作文件。这些信息可以用来故障排查和查询调优。这些视图中的信息也可以被用来为 Greenplum 数据库配置参数 *gp_workfile_limit_per_query* 和 *gp_workfile_limit_per_segment* 指定值。

在模式 *gp_toolkit* 中有下面这些视图：

- *gp_workfile_entries* 视图为当前在 Segment 上创建了工作文件的每个操作符都包含一行。
- *gp_workfile_usage_per_query* 视图为当前在 Segment 上创建了工作文件的每个查询都包含一行。
- *gp_workfile_usage_per_segment* 视图为每个 Segment 都包含一行。每一行显示了当前在该 Segment 上用于工作文件的总磁盘空间。

有关使用 *gp_toolkit* 的信息请见 [使用 gp_toolkit](#)。

查看数据库服务器日志文件

Greenplum 数据库中的每一个数据库实例（Master 和 Segment）都运行着一个有着自己的服务器日志文件的 PostgreSQL 数据库服务器。日常的日志文件被创建在 Master 和每个 Segment 的数据目录中的 pg_log 目录下。

日志文件格式

服务器日志文件被写成一种逗号分隔值（CSV）格式。一些日志项并不会所有的域都有值。例如，只有与一个查询工作者进程相关的日志项才会有 slice_id 值。可以用查询的会话标识符 (*gp_session_id*) 和命令标识符 (*gp_command_count*) 来确定一个特定查询的相关日志项。

下列域会被写入到日志中：

Table 2. Greenplum 数据库服务器日志格式

#	域名称	数据类型	描述
1	event_time	timestamp with time zone	该日志项被写入到日志中的时间
2	user_name	varchar(100)	数据库用户名
3	database_name	varchar(100)	数据库名

4	process_id	varchar(10)	系统进程ID (以"p"为前缀)
5	thread_id	varchar(50)	线程号 (以"th"为前缀)
6	remote_host	varchar(100)	在Master上, 是客户端机器的主机名/地址。在Segment上, 是Master的主机名/地址。
7	remote_port	varchar(10)	Segment或者Master的端口号
8	session_start_time	timestamp with time zone	会话连接被打开的时间
9	transaction_id	int	Master上的顶层事务ID。这个ID是任何子事务的父亲。
10	gp_session_id	text	会话标识符号 (以"con"为前缀)
11	gp_command_count	text	会话中的命令号 (以"cmd"为前缀)
12	gp_segment	text	Segment内容标识符 (主Segment以"seg"为前缀, 镜像Segment以"mir"为前缀)。Master的内容ID总是为-1。
13	slice_id	text	切片ID (被执行的查询计划的一部分)
14	distr_trnx_id	text	分布式事务ID
15	local_trnx_id	text	本地事务ID
16	sub_trnx_id	text	子事务ID
17	event_severity	varchar(10)	值包括: LOG、ERROR、FATAL、PANIC、DEBUG1、DEBUG2
18	sql_state_code	varchar(10)	与日志消息相关的SQL状态代码
19	event_message	text	日志或者错误消息文本
20	event_detail	text	与一个错误或者警告消息相关的详细消息文本
21	event_hint	text	与一个错误或者警告消息相关的提示消息文本
22	internal_query	text	内部生成的查询文本
23	internal_query_pos	int	内部生成的查询文本的指针式索引
24	event_context	text	这个消息产生的上下文
25	debug_query_string	text	完整的用户提供的查询字符串, 用于调试。内部使用可能会修改这个字符串。
26	error_cursor_pos	int	该查询字符串中的指针式索引
27	func_name	text	这个消息产生的函数
28	file_name	text	产生该消息的内部代码文件
29	file_line	int	产生该消息的内部代码文件的行号
30	stack_trace	text	与这个消息相关的堆栈跟踪

搜索Greenplum服务器日志文件

Greenplum数据库提供一个名为`gplogfilter`的工具, 它可以在一个Greenplum数据库 日志文件中搜索匹配指定条件的项。默认情况下, 这个工具在默认日志位置搜索Greenplum数据库的Master日志。例如, 要显示Master日志文件的最后三行:

```
$ gplogfilter -n 3
```

要同时搜索所有Segment的日志文件, 可以通过`gpssh`工具来运行`gplogfilter`。例如, 要显示每个Segment日志文件的最后三行:

```
$ gpssh -f seg_host_file
```

```
=> source /usr/local/greenplum-db/greenplum_path.sh
=> gplogfilter -n 3 /gpdata/gp*/pg_log/gpdb*.log
```

使用gp_toolkit

使用Greenplum数据库的管理方案`gp_toolkit`来查询系统目录、日志文件和操作系统环境以得到系统状态信息。`gp_toolkit`方案包含一些可以用SQL命令访问的视图。`gp_toolkit`方案对所有数据库用户都可以访问。一些对象要求超级用户权限。用与下面类似的命令把`gp_toolkit`方案增加到用户的方案搜索路径中：

```
=> ALTER ROLE myrole SET search_path TO myschema, gp_toolkit;
```

有关可用的管理方案视图及其用法的描述，请见 *Greenplum Database Reference Guide*.

SQL标准错误代码

下面的表格列出了所有定义好的错误代码。有些没有被用到，但是在SQL标准中有定义。其中也显示了错误分类。对于每一种错误分类都有一个标准错误代码，它的最后三个字符为000。这种代码只用于落入该分类却没有更详细代码的错误情况。

每种错误代码的PL/pgSQL情况名称与该表中显示的短语相同，但是把空格替换成下划线。例如，代码22012 (DIVISION BY ZERO) 的情况名称是DIVISION_BY_ZERO。情况名称不区分大小写。

Note: 与error截然不同，PL/pgSQL不识别warning、condition names，warning、error和condition names的分类分别为00、01和02。

Table 3. SQL代码

错误代码	含义	常量
Class 00 — Successful Completion		
00000	SUCCESSFUL COMPLETION	successful_completion
Class 01 — Warning		
01000	WARNING	warning
0100C	DYNAMIC RESULT SETS RETURNED	dynamic_result_sets_returned
01008	IMPLICIT ZERO BIT PADDING	implicit_zero_bit_padding
01003	NULL VALUE ELIMINATED IN SET FUNCTION	null_value_eliminated_in_set_function
01007	PRIVILEGE NOT GRANTED	privilege_not_granted
01006	PRIVILEGE NOT REVOKED	privilege_not_revoked
01004	STRING DATA RIGHT TRUNCATION	string_data_right_truncation
01P01	DEPRECATED FEATURE	deprecated_feature
Class 02 — No Data (this is also a warning class per the SQL standard)		
02000	NO DATA	no_data
02001	NO ADDITIONAL DYNAMIC RESULT SETS RETURNED	no_additional_dynamic_result_sets_returned
Class 03 — SQL Statement Not Yet Complete		
03000	SQL STATEMENT NOT YET COMPLETE	sql_statement_not_yet_complete
Class 08 — Connection Exception		

08000	CONNECTION EXCEPTION	connection_exception
08003	CONNECTION DOES NOT EXIST	connection_does_not_exist
08006	CONNECTION FAILURE	connection_failure
08001	SQLCLIENT UNABLE TO ESTABLISH SQLCONNECTION	sqlclient_unable_to_establish_sqlconnection
08004	SQLSERVER REJECTED ESTABLISHMENT OF SQLCONNECTION	sqlserver_rejected_establishment_of_sqlconnection
08007	TRANSACTION RESOLUTION UNKNOWN	transaction_resolution_unknown
08P01	PROTOCOL VIOLATION	protocolViolation
Class 09 — Triggered Action Exception		
09000	TRIGGERED ACTION EXCEPTION	triggered_action_exception
Class 0A — Feature Not Supported		
0A000	FEATURE NOT SUPPORTED	feature_not_supported
Class 0B — Invalid Transaction Initiation		
0B000	INVALID TRANSACTION INITIATION	invalid_transaction_initiation
Class 0F — Locator Exception		
0F000	LOCATOR EXCEPTION	locator_exception
0F001	INVALID LOCATOR SPECIFICATION	invalid_locator_specification
Class 0L — Invalid Grantor		
0L000	INVALID GRANTOR	invalid_grantor
0LP01	INVALID GRANT OPERATION	invalid_grant_operation
Class 0P — Invalid Role Specification		
0P000	INVALID ROLE SPECIFICATION	invalid_role_specification
Class 21 — Cardinality Violation		
21000	CARDINALITY VIOLATION	cardinalityViolation
Class 22 — Data Exception		
22000	DATA EXCEPTION	data_exception
2202E	ARRAY SUBSCRIPT ERROR	array_subscript_error
22021	CHARACTER NOT IN REPERTOIRE	character_not_in_repertoire
22008	DATETIME FIELD OVERFLOW	datetime_field_overflow
22012	DIVISION BY ZERO	division_by_zero
22005	ERROR IN ASSIGNMENT	error_in_assignment
2200B	ESCAPE CHARACTER CONFLICT	escape_character_conflict

22022	INDICATOR OVERFLOW	indicator_overflow
22015	INTERVAL FIELD OVERFLOW	interval_field_overflow
2201E	INVALID ARGUMENT FOR LOGARITHM	invalid_argument_for_logarithm
2201F	INVALID ARGUMENT FOR POWER FUNCTION	invalid_argument_for_power_function
2201G	INVALID ARGUMENT FOR WIDTH BUCKET FUNCTION	invalid_argument_for_width_bucket_function
22018	INVALID CHARACTER VALUE FOR CAST	invalid_character_value_for_cast
22007	INVALID DATETIME FORMAT	invalid_datetime_format
22019	INVALID ESCAPE CHARACTER	invalid_escape_character
2200D	INVALID ESCAPE OCTET	invalid_escape_octet
22025	INVALID ESCAPE SEQUENCE	invalid_escape_sequence
22P06	NONSTANDARD USE OF ESCAPE CHARACTER	nonstandard_use_of_escape_character
22010	INVALID INDICATOR PARAMETER VALUE	invalid_indicator_parameter_value
22020	INVALID LIMIT VALUE	invalid_limit_value
22023	INVALID PARAMETER VALUE	invalid_parameter_value
2201B	INVALID REGULAR EXPRESSION	invalid_regular_expression
22009	INVALID TIME ZONE DISPLACEMENT VALUE	invalid_time_zone_displacement_value
2200C	INVALID USE OF ESCAPE CHARACTER	invalid_use_of_escape_character
2200G	MOST SPECIFIC TYPE MISMATCH	most_specific_type_mismatch
22004	NULL VALUE NOT ALLOWED	null_value_not_allowed
22002	NULL VALUE NO INDICATOR PARAMETER	null_value_no_indicator_parameter
22003	NUMERIC VALUE OUT OF RANGE	numeric_value_out_of_range
22026	STRING DATA LENGTH MISMATCH	string_data_length_mismatch
22001	STRING DATA RIGHT TRUNCATION	string_data_right_truncation
22011	SUBSTRING ERROR	substring_error
22027	TRIM ERROR	trim_error
22024	UNTERMINATED C STRING	unterminated_c_string
2200F	ZERO LENGTH	zero_length_character_string

	CHARACTER STRING	
22P01	FLOATING POINT EXCEPTION	floating_point_exception
22P02	INVALID TEXT REPRESENTATION	invalid_text_representation
22P03	INVALID BINARY REPRESENTATION	invalid_binary_representation
22P04	BAD COPY FILE FORMAT	bad_copy_file_format
22P05	UNTRANSLATABLE CHARACTER	untranslatable_character
Class 23 — Integrity Constraint Violation		
23000	INTEGRITY CONSTRAINT VIOLATION	integrity_constraintViolation
23001	RESTRICT VIOLATION	restrict_violation
23502	NOT NULL VIOLATION	not_null_violation
23503	FOREIGN KEY VIOLATION	foreign_key_violation
23505	UNIQUE VIOLATION	unique_violation
23514	CHECK VIOLATION	check_violation
Class 24 — Invalid Cursor State		
24000	INVALID CURSOR STATE	invalid_cursor_state
Class 25 — Invalid Transaction State		
25000	INVALID TRANSACTION STATE	invalid_transaction_state
25001	ACTIVE SQL TRANSACTION	active_sql_transaction
25002	BRANCH TRANSACTION ALREADY ACTIVE	branch_transaction_already_active
25008	HELD CURSOR REQUIRES SAME ISOLATION LEVEL	held_cursor_requires_same_isolation_level
25003	INAPPROPRIATE ACCESS MODE FOR BRANCH TRANSACTION	inappropriate_access_mode_for_branch_transaction
25004	INAPPROPRIATE ISOLATION LEVEL FOR BRANCH TRANSACTION	inappropriate_isolation_level_for_branch_transaction
25005	NO ACTIVE SQL TRANSACTION FOR BRANCH TRANSACTION	no_active_sql_transaction_for_branch_transaction
25006	READ ONLY SQL TRANSACTION	read_only_sql_transaction
25007	SCHEMA AND DATA STATEMENT MIXING NOT SUPPORTED	schema_and_data_statement_mixing_not_supported
25P01	NO ACTIVE SQL TRANSACTION	no_active_sql_transaction
25P02	IN FAILED SQL	in_failed_sql_transaction

TRANSACTION		
Class 26 — Invalid SQL Statement Name		
26000	INVALID SQL STATEMENT NAME	invalid_sql_statement_name
Class 27 — Triggered Data Change Violation		
27000	TRIGGERED DATA CHANGE VIOLATION	triggered_data_changeViolation
Class 28 — Invalid Authorization Specification		
28000	INVALID AUTHORIZATION SPECIFICATION	invalid_authorization_specification
Class 2B — Dependent Privilege Descriptors Still Exist		
2B000	DEPENDENT PRIVILEGE DESCRIPTORS STILL EXIST	dependent_privilege_descriptors_still_exist
2BP01	DEPENDENT OBJECTS STILL EXIST	dependent_objects_still_exist
Class 2D — Invalid Transaction Termination		
2D000	INVALID TRANSACTION TERMINATION	invalid_transaction_termination
Class 2F — SQL Routine Exception		
2F000	SQL ROUTINE EXCEPTION	sql_routine_exception
2F005	FUNCTION EXECUTED NO RETURN STATEMENT	function_executed_no_return_statement
2F002	MODIFYING SQL DATA NOT PERMITTED	modifying_sql_data_not_permitted
2F003	PROHIBITED SQL STATEMENT ATTEMPTED	prohibited_sql_statement_attempted
2F004	READING SQL DATA NOT PERMITTED	reading_sql_data_not_permitted
Class 34 — Invalid Cursor Name		
34000	INVALID CURSOR NAME	invalid_cursor_name
Class 38 — External Routine Exception		
38000	EXTERNAL ROUTINE EXCEPTION	external_routine_exception
38001	CONTAINING SQL NOT PERMITTED	containing_sql_not_permitted
38002	MODIFYING SQL DATA NOT PERMITTED	modifying_sql_data_not_permitted
38003	PROHIBITED SQL STATEMENT ATTEMPTED	prohibited_sql_statement_attempted
38004	READING SQL DATA NOT PERMITTED	reading_sql_data_not_permitted
Class 39 — External Routine Invocation Exception		
39000	EXTERNAL ROUTINE INVOCATION EXCEPTION	external_routine_invocation_exception
39001	INVALID SQLSTATE	invalid_sqlstate_returned

	RETURNED	
39004	NULL VALUE NOT ALLOWED	null_value_not_allowed
39P01	TRIGGER PROTOCOL VIOLATED	trigger_protocol_violated
39P02	SRF PROTOCOL VIOLATED	srf_protocol_violated
Class 3B — Savepoint Exception		
3B000	SAVEPOINT EXCEPTION	savepoint_exception
3B001	INVALID SAVEPOINT SPECIFICATION	invalid_savepoint_specification
Class 3D — Invalid Catalog Name		
3D000	INVALID CATALOG NAME	invalid_catalog_name
Class 3F — Invalid Schema Name		
3F000	INVALID SCHEMA NAME	invalid_schema_name
Class 40 — Transaction Rollback		
40000	TRANSACTION ROLLBACK	transaction_rollback
40002	TRANSACTION INTEGRITY CONSTRAINT VIOLATION	transaction_integrity_constraintViolation
40001	SERIALIZATION FAILURE	serialization_failure
40003	STATEMENT COMPLETION UNKNOWN	statement_completion_unknown
40P01	DEADLOCK DETECTED	deadlock_detected
Class 42 — Syntax Error or Access Rule Violation		
42000	SYNTAX ERROR OR ACCESS RULE VIOLATION	syntax_error_or_access_ruleViolation
42601	SYNTAX ERROR	syntax_error
42501	INSUFFICIENT PRIVILEGE	insufficient_privilege
42846	CANNOT COERCE	cannot_coerce
42803	GROUPING ERROR	grouping_error
42830	INVALID FOREIGN KEY	invalid_foreign_key
42602	INVALID NAME	invalid_name
42622	NAME TOO LONG	name_too_long
42939	RESERVED NAME	reserved_name
42804	DATATYPE MISMATCH	datatype_mismatch
42P18	INDETERMINATE DATATYPE	indeterminate_datatype
42809	WRONG OBJECT TYPE	wrong_object_type
42703	UNDEFINED COLUMN	undefined_column
42883	UNDEFINED FUNCTION	undefined_function
42P01	UNDEFINED TABLE	undefined_table

42P02	UNDEFINED PARAMETER	undefined_parameter
42704	UNDEFINED OBJECT	undefined_object
42701	DUPLICATE COLUMN	duplicate_column
42P03	DUPLICATE CURSOR	duplicate_cursor
42P04	DUPLICATE DATABASE	duplicate_database
42723	DUPLICATE FUNCTION	duplicate_function
42P05	DUPLICATE PREPARED STATEMENT	duplicate_prepared_statement
42P06	DUPLICATE SCHEMA	duplicate_schema
42P07	DUPLICATE TABLE	duplicate_table
42712	DUPLICATE ALIAS	duplicate_alias
42710	DUPLICATE OBJECT	duplicate_object
42702	AMBIGUOUS COLUMN	ambiguous_column
42725	AMBIGUOUS FUNCTION	ambiguous_function
42P08	AMBIGUOUS PARAMETER	ambiguous_parameter
42P09	AMBIGUOUS ALIAS	ambiguous_alias
42P10	INVALID COLUMN REFERENCE	invalid_column_reference
42611	INVALID COLUMN DEFINITION	invalid_column_definition
42P11	INVALID CURSOR DEFINITION	invalid_cursor_definition
42P12	INVALID DATABASE DEFINITION	invalid_database_definition
42P13	INVALID FUNCTION DEFINITION	invalid_function_definition
42P14	INVALID PREPARED STATEMENT DEFINITION	invalid_prepared_statement_definition
42P15	INVALID SCHEMA DEFINITION	invalid_schema_definition
42P16	INVALID TABLE DEFINITION	invalid_table_definition
42P17	INVALID OBJECT DEFINITION	invalid_object_definition
Class 44 — WITH CHECK OPTION Violation		
44000	WITH CHECK OPTION VIOLATION	with_check_optionViolation
Class 53 — Insufficient Resources		
53000	INSUFFICIENT RESOURCES	insufficient_resources
53100	DISK FULL	disk_full
53200	OUT OF MEMORY	out_of_memory
53300	TOO MANY CONNECTIONS	too_many_connections
Class 54 — Program Limit Exceeded		
54000	PROGRAM LIMIT EXCEEDED	program_limit_exceeded

54001	STATEMENT TOO COMPLEX	statement_too_complex
54011	TOO MANY COLUMNS	too_many_columns
54023	TOO MANY ARGUMENTS	too_many_arguments
Class 55 — Object Not In Prerequisite State		
55000	OBJECT NOT IN PREREQUISITE STATE	object_not_in_prerequisite_state
55006	OBJECT IN USE	object_in_use
55P02	CANT CHANGE RUNTIME PARAM	cant_change_runtime_param
55P03	LOCK NOT AVAILABLE	lock_not_available
Class 57 — Operator Intervention		
57000	OPERATOR INTERVENTION	operator_intervention
57014	QUERY CANCELED	query_canceled
57P01	ADMIN SHUTDOWN	admin_shutdown
57P02	CRASH SHUTDOWN	crash_shutdown
57P03	CANNOT CONNECT NOW	cannot_connect_now
Class 58 — System Error (errors external to Greenplum Database)		
58030	IO ERROR	io_error
58P01	UNDEFINED FILE	undefined_file
58P02	DUPLICATE FILE	duplicate_file
Class F0 — Configuration File Error		
F0000	CONFIG FILE ERROR	config_file_error
F0001	LOCK FILE EXISTS	lock_file_exists
Class P0 — PL/pgSQL Error		
P0000	PLPGSQL ERROR	plpgsql_error
P0001	RAISE EXCEPTION	raise_exception
P0002	NO DATA FOUND	no_data_found
P0003	TOO MANY ROWS	too_many_rows
Class XX — Internal Error		
XX000	INTERNAL ERROR	internal_error
XX001	DATA CORRUPTED	data_corrupted
XX002	INDEX CORRUPTED	index_corrupted

Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统

- 关于Greenplum数据库
发布版本号

- 启动和停止Greenplum数据库

- 访问数据库
- 配置Greenplum数据库系统

- 启用压缩

- 启用高可用和数据持久化特征
- 备份和恢复数据库
- 扩容Greenplum系统
- 监控Greenplum系统

日常系统维护任务

- Recommended Monitoring and Maintenance Tasks

- 管理Greenplum数据库访问
- 定义数据库对象
- 分布与倾斜
- 插入、更新、和删除数据

要保持一个Greenplum数据库系统高效运行，必须对数据库定期清理过期数据并且更新表统计信息，这样查询优化器才能有准确的信息。

Greenplum数据库要求定期执行特定的任务来达到最优性能。这里讨论的任务都是必须的，但数据库管理员可以使用标准的UNIX工具（如cron脚本）来自动化这些任务。管理员建立适当的脚本并且检查它们是否成功执行。关于用来保持Greenplum系统最优化运行的额外建议维护活动，请见[Recommended Monitoring and Maintenance Tasks](#)。

Parent topic: [管理一个Greenplum系统](#)

例行清理和分析

Greenplum数据库中使用的MVCC事务并发模型的设计意味着被删除或者被更新的数据行仍在磁盘上占据物理空间，即便它们已经对新事务不可见。如果数据库进行了很多更新和删除，会有很多过期行存在并且它们所使用的空间必须使用 VACUUM命令来回收。VACUUM命令还会收集表级的统计信息，例如行数和页面数，因此即便无需从被更新或者被删除行回收空间，也还是有必要去清理追加优化表。

清理一个追加优化表遵循一种和清理堆表不同的处理逻辑。在每一个Segment上，会创建一个新的Segment文件并且把可见行从当前Segment复制到该文件中。当Segment文件被拷贝完时，将会安排删除原始的Segment文件并且让新的Segment文件变得可用。这要求足够的可用磁盘空间用于拷贝可见行，直到原始的Segment文件被删除为止。

如果一个Segment文件中隐藏行和所有行的比率低于一个阈值（默认是10），该Segment文件不会被紧缩。该阈值可以通过`gp_appendonly_compaction_threshold`服务器配置参数配置。`VACUUM FULL`忽略`gp_appendonly_compaction_threshold`的值并且不管该比率是多少都会重写Segment文件。

可以使用`gp_toolkit`模式中的`__gp_aovisimap_compaction_info()`函数来查看追加优化表上的VACUUM操作的效果。

有关`__gp_aovisimap_compaction_info()`函数的信息请见[Greenplum Database Reference Guide](#)“检查追加优化表”部分



□ 查询数据

可以使用gp_appendonly_compaction服务器配置参数为追加优化表禁用 VACUUM 操作。

有关清理数据库的细节请见[清理数据库](#)。

有关gp_appendonly_compaction_threshold服务器配置参数和VACUUM 命令的信息请见 *Greenplum Database Reference Guide*。

事务ID管理

Greenplum的MVCC事务机制依赖于比较事务ID (XID) 号来判断当前数据对于其他事务的可见性。 事务ID号使用一种模 2^{32} 的算法来比较，因此一个运行了超过二十亿事务的 Greenplum系统可能会遇到事务ID回卷，即过去的事务变成了未来的事务。这意味着过去的事物的输出变得不可见。因此，每过二十亿个事务就有必要VACUUM每个数据库中的 每个表至少一次。

Greenplum数据库只对涉及DDL或者DML操作的事务分配XID值，通常也只有这些事务需要XID。

Important: Greenplum数据库会监控事务ID。如果没有定期清理数据库，Greenplum 数据库将产生警告和错误。

当事务ID中相当多的一部分变得不再可用并且事务ID回卷还没有发生时，Greenplum数据库会发出下面的警告：

```
WARNING: database "database_name" must be vacuumed
within number_of_transactions transactions
```

当这个警告被发出时，需要一次VACUUM操作。如果没有执行VACUUM 操作，当Greenplum数据库在事务ID回卷发生前达到一个限制，它会停止创建新的事务。在停止创建 事务以避免可能的数据丢失时，Greenplum数据库会发出这个错误：

```
FATAL: database is not accepting commands to avoid
wraparound data loss in database "database_name"
```

Greenplum数据库配置参数xid_warn_limit控制何时显示该警告。参数 xid_stop_limit 控制何时Greenplum数据库停止创建事务。

从一次事务ID限制错误中恢复

当Greenplum数据库由于不频繁的VACUUM维护而达到xid_stop_limit事务ID限制时，它会变得没有响应。为了从这种情况下恢复过来，作为数据库管理员执行下面的步骤：

1. 关闭Greenplum数据库。
2. 临时将xid_stop_limit降低10,000,000。
3. 启动Greenplum数据库。
4. 在所有受影响的数据库上运行VACUUM FREEZE。
5. 将xid_stop_limit重置为原来的值。
6. 重启Greenplum数据库。

有关这些配置参数的信息请见 *Greenplum Database Reference Guide*。

有关事务ID回卷的信息请见 [PostgreSQL documentation](#) .

系统目录维护

多次使用CREATE和DROP命令的数据库更新会增长系统目录尺寸并且影响系统性能。例如，运行很多次DROP TABLE语句会降低总体系统性能，因为在目录表上的元数据查询期间会需要更多扫描时间。性能损失会在数千次或者数万次DROP TABLE语句之间发生，具体时间取决于系统。

应该定期维护系统目录来回收已删除对象所占据的空间。如果长时间没有运行这种定期回收操作，那可能需要运行一个更彻底的回收操作来清理系统目录。这个主题会描述这两种方式。

常规系统目录维护

推荐周期性地在系统目录上运行REINDEX和VACUUM来清理系统索引和表中已删除对象所占用的空间。如果常规的数据库操作包括很多DROP语句，那么每天在非峰值时间用VACUUM命令运行一次系统目录维护是安全且适当的。用户可以在系统可用时执行这种操作。

下面是Greenplum数据库系统目录维护步骤。

1. 在系统表上执行REINDEX命令以重建系统表索引。该操作移除索引膨胀并提高VACUUM性能。

Note: 当在系统表上执行REINDEX操作时，表上会产生锁，此时可能会对当前正在运行的查询产生比较大的影响。建议在系统负载较低时执行REINDEX操作，以避免对正在运行的业务操作产生较大的干扰。

2. 在系统表上执行VACUUM操作。
3. 在系统表上执行ANALYZE操作，用以更新统计信息。

下面的示例脚本在一个Greenplum数据库系统目录上执行一次REINDEX、 VACUUM 以及ANALYZE操作：将脚本中的<database-name>替换为真实数据库名。

```
#!/bin/bash
DBNAME=<database-name>
SYSTABLES=' pg_catalog.' || relname || ';' FROM pg_class
a, pg_namespace b
WHERE a.relnamespace=b.oid AND b.nspname='pg_catalog' AND
a.relkind='r'

reindexdb --system -d $DBNAME
psql -tc "SELECT 'VACUUM' || $SYSTABLES" $DBNAME | psql -a
$DBNAME
analyzedb -s pg_catalog -d $DBNAME
```

Note: 如果在系统维护时间内已经开始了正常的系统目录维护操作，但是由于时间原因想要停止某一维护进程，此时可以运行Greenplum数据库函数pg_cancel_backend(<PID>) 以安全停止该Greenplum数据库进程。

深度系统目录维护

如果很长时间都没有执行一次系统目录维护操作，该目录可能因为废弃空间而膨胀。这会导致简单的元数据操作都会等待很长时间。在psql中用\ d命令列出用户表需要超过两秒的等待，就是目录膨胀的一种征兆。

如果发现系统目录膨胀的征兆，就必须在计划好的停机时段用VACUUM FULL执行一次深度系统目录维护操作。在这一时段中，停止系统上的所有目录活动，这种VACUUM FULL 系统目录维护过程会对系统目录加排他锁。

运行定期系统目录维护操作可以防止对这种更高开销操作的需求。

以下是深度系统目录维护操作的步骤。

1. 停止Greenplum数据库系统上的所有活动元数据操作。

2. 在系统表上执行REINDEX操作以重建系统表索引。该操作移除索引上的膨胀并提高 VACUUM 操作性能。
3. 在系统表上执行VACUUM FULL命令，具体查看下面的注释。
4. 在系统表上执行ANALYZE命令以更新系统表的统计信息。

Note: 通常来说，表pg_attribute是最大的系统表。如果pg_attribute表膨胀的特别厉害，该表上的VACUUM FULL操作可能需要更长的时间并且可能需要分开执行。以下两种情况的出现预示着pg_attribute表膨胀很厉害并且需要长时间的VACUUM FULL操作：

- pg_attribute表包含大量记录。
- gp_toolkit.gp_bloat_diag视图中的信息显示pg_attribute的状态为significant amount of bloat

为查询优化进行清理和分析

Greenplum数据库使用一种基于代价的查询优化器，它依赖于数据库的统计信息。准确的统计信息帮助查询优化器更好的评估选择度以及一个查询操作检索的行数。这些评估会帮助它选择最有效的查询计划。ANALYZE 命令会为查询优化器收集列级的统计信息。

可以在同一个命令中同时执行VACUUM和ANALYZE操作。例如：

```
=# VACUUM ANALYZE mytable;
```

当在一个显著膨胀的表（显著膨胀的表磁盘空间被已删除或者已废弃行占据）上运行VACUUM ANALYZE命令时，该命令可能会产生不正确的统计信息。对于大型的表，ANALYZE命令会从行的一个随机采样中计算统计信息。它会通过计算采样中每页的平均行数与表中实际页数的成绩来估算表中的总行数。如果采样包含很多空页，估计出的行计数可能会不准确。

对于一个表，可以在gp_toolkit视图gp_bloat_diag中查看有关未使用磁盘空间（被已删除或者已废弃行占据的空间）量的信息。如果一个表的bdidiag列包含值significant amount of bloat suspected，那么相当多的表磁盘空间由未使用空间组成。在一个表被清理后，相关项会被加入到gp_bloat_diag视图中。

要从表中移除未使用的磁盘空间，可以在该表上运行命令VACUUM FULL。由于对表锁的需求，VACUUM FULL可能无法在非维护时段

运行。

作为一种临时的变通方案，可以运行ANALYZE来计算列统计信息，然后在该表上运行 VACUUM来生成准确的行计数。这个例子在*cust_info*表上先运行ANALYZE，然后运行VACUUM。

```
ANALYZE cust_info;  
VACUUM cust_info;
```

Important: 如果想要在启用了GPORCA（默认启用）的情况下对分区表执行查询，必须用ANALYZE命令在分区表根分区上收集统计信息。有关GPORCA的信息，请见[GPORCA概述](#)。

Note: 可以使用Greenplum数据库工具analyzedb 来更新表统计信息。表可以被并行地分析。对于追加优化表，只有统计信息不是当前值时，analyzedb 才会更新统计信息。有关analyzedb 工具的信息，请见[analyzedb](#)。

常规的索引重建

对于B-树索引，一个刚刚构建的索引访问起来比一个已经更新过很多次的索引要快一点，因为在新构建的索引中逻辑上相邻的页面在物理上也相邻。定期地重建旧的索引可以提升访问速度。如果一个页面上除了索引以外的数据都被删除，那么索引页上对应的也会产生浪费的空间。重建索引操作会回收这些被浪费的空间。在Greenplum数据库中，删除索引 (DROP INDEX)再重建 (CREATE INDEX)通常比直接使用REINDEX 命令要快。

对于有索引的表列，批量更新或者插入之类的一些操作可能会执行得更慢，因为需要更新索引。为了提高带有索引的表上的批量操作性能，可以先删除掉索引，然后执行批量操作，最后再重建索引。

管理Greenplum数据库日志文件

- [数据库服务器日志文件](#)
- [管理工具日志文件](#)

数据库服务器日志文件

Greenplum数据库的日志输出常常会体量很大，尤其是在调试级别时，

用户不需要无限期保存它。管理员可以应该定期清理日志文件。

Greenplum数据库在Master和所有的Segment实例上都启用了日志文件轮转。日志文件被创建在Master以及每个Segment的数据目录中的pg_log子目录下，这些文件使用命名规

则：gpdb-YYYY-MM-DD_hhmmss.csv。数据库管理员需要编写脚本或程序定期清理Master以及每个Segment日志目录pg_log下的旧文件。

日志轮转可以被当前日志文件的大小或当前日志文件的年龄触发。log_rotation_size配置参数设置触发日志轮转的单个日志文件大小。当日志文件大小等于或大于特定的大小时，该文件会停止写入，系统重新创建一个新的日志文件。log_rotation_size的值的单位为KB。默认为1048576，即1GB。如果log_rotation_size设置为0，表示基于文件大小的日志轮转被禁用。

log_rotation_age配置参数定义了出发日志轮转的日志文件年龄。当文件自创建时开始到达特定长度的时间时，该文件停止写入，系统重新创建一个新的日志文件。默认的log_rotation_age时间为1d，即当前日志文件创建24小时之后。如果log_rotation_age设置为0，表示基于时间的轮转被禁用。

更多查看数据库日志文件的信息，请见[查看数据库服务器日志文件](#)。

管理工具日志文件

Greenplum数据库管理工具的日志文件默认被写入到~/gpAdminLogs。管理日志文件的命名规则是：

```
script_name_date.log
```

日志详情的格式是：

```
timestamp:utility:host:user:[ INFO | WARN | FATAL ] :message
```

每次一个工具运行时，就会向其每日的日志文件中增加与其执行相关的信息。

and Maintenance Tasks

Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统
 - 关于Greenplum数据库
发布版本号
 - 启动和停
止Greenplum数据库
 - 访问数据库
 - 配置Greenplum数据库
系统
 - 启用压缩
 - 启用高可用和数据持久
化特征
 - 备份和恢复数据库
 - 扩容Greenplum系统
 - 监控Greenplum系统
 - 日常系统维护任务
- Recommended Monitoring and Maintenance Tasks**
- 管理Greenplum数据库访问
- 定义数据库对象
- 分布与倾斜
- 插入, 更新, 和删除数据
- 查询数据

This section lists monitoring and maintenance activities recommended to ensure high availability and consistent performance of your Greenplum Database cluster.

The tables in the following sections suggest activities that a Greenplum System Administrator can perform periodically to ensure that all components of the system are operating optimally. Monitoring activities help you to detect and diagnose problems early. Maintenance activities help you to keep the system up-to-date and avoid deteriorating performance, for example, from bloated system tables or diminishing free disk space.

It is not necessary to implement all of these suggestions in every cluster; use the frequency and severity recommendations as a guide to implement measures according to your service requirements.

Parent topic: [管理一个Greenplum系统](#)

Database State Monitoring Activities

Table 1. Database State Monitoring Activities

Activity	Procedure	Corrective Actions
<p>List segments that are currently down. If any rows are returned, this should generate a warning or alert.</p> <p>Recommended frequency: run every 5 to 10 minutes</p> <p>Severity: IMPORTANT</p>	<p>Run the following query in the <code>postgres</code> database:</p> <pre>SELECT * FROM gp_segment_configuration WHERE status <> 'u';</pre>	<p>If the query returns any rows, follow these steps to correct the problem:</p> <ol style="list-style-type: none"> Verify that the hosts with down segments are responsive. If hosts are OK, check the <code>pg_log</code> files for the queries and minimize the down segments to discover the root cause of the segments going down.

		<p>3. If no unexpected errors are found, run the <code>gprecoverseg</code> utility to bring the segments back online.</p>
	<p>Check for segments that are currently in change tracking mode. If any rows are returned, this should generate a warning or alert.</p> <p>Recommended frequency: run every 5 to 10 minutes</p> <p>Severity: IMPORTANT</p>	<p>Execute the following query in the <code>postgres</code> database:</p> <pre>SELECT * FROM gp_segment_configuration WHERE mode = 'c';</pre> <p>If the query returns any rows, follow these steps to correct the problem:</p> <ol style="list-style-type: none"> 1. Verify that hosts with down segments are responsive. 2. If hosts are OK, check the <code>pg_log</code> files for the primaries and mirrors of the down segments to determine the root cause of the segments going down. 3. If no unexpected errors are found, run the <code>gprecoverseg</code> utility to bring the segments back online.
	<p>Check for segments that are currently resyncing. If rows are returned, this should generate a warning or alert.</p> <p>Recommended frequency: run</p>	<p>Execute the following query in the <code>postgres</code> database:</p> <pre>SELECT * FROM gp_segment_configuration WHERE mode = 'r';</pre> <p>When this query returns rows, it implies that the segments are in the process of being resynched. If the state does not change from 'r' to 's', then check the <code>pg_log</code> files from the</p>

<p>every 5 to 10 minutes</p> <p>Severity: IMPORTANT</p>		<p>primaries and mirrors of the affected segments for errors.</p>
<p>Check for segments that are not operating in their optimal role. If any segments are found, the cluster may not be balanced. If any rows are returned this should generate a warning or alert.</p> <p>Recommended frequency: run every 5 to 10 minutes</p> <p>Severity: IMPORTANT</p>	<p>Execute the following query in the <code>postgres</code> database:</p> <pre>SELECT * FROM gp_segment_configuration WHERE preferred_role <> role;</pre>	<p>When the segments are not running in their preferred role, hosts have uneven numbers of primary segments on each host, implying that processing is skewed. Wait for a potential window and restart the database to bring the segments into their preferred roles.</p>
<p>Run a distributed query to test that it runs on all segments. One row should be returned for each primary segment.</p> <p>Recommended frequency: run every 5 to 10 minutes</p> <p>Severity: CRITICAL</p>	<p>Execute the following query in the <code>postgres</code> database:</p> <pre>SELECT gp_segment_id, count(*) FROM gp_dist_random('pg_class') GROUP BY 1;</pre>	<p>If this query fails, there is an issue dispatching to some segments in the cluster. This is a rare event. Check the hosts that are not able to be dispatched to ensure there is no hardware or networking issue.</p>
<p>Test the state of master mirroring</p>	<p>Execute the following query in the <code>postgres</code> database:</p>	<p>Check the <code>pg_log</code></p>

<p>on a Greenplum Database 4.2 or earlier cluster. If the value is "Not Synchronized", raise an alert or warning.</p> <p>Recommended frequency: run every 5 to 10 minutes</p> <p>Severity: IMPORTANT</p>	<pre>SELECT summary_state FROM gp_master_mirroring;</pre>	<p>from the master and standby master for errors. If there are no unexpected errors and the machines are up, run the <code>gpinitstandby</code> utility to bring the standby online. This requires a database restart on GPDB 4.2 and earlier.</p>
<p>Test the state of master mirroring on Greenplum Database. If the value is not "STREAMING", raise an alert or warning.</p> <p>Recommended frequency: run every 5 to 10 minutes</p> <p>Severity: IMPORTANT</p>	<p>Run the following <code>psql</code> command:</p> <pre>psql dbname -c 'SELECT procpid, state FROM pg_stat_replication;'</pre>	<p>Check the <code>pg_log</code> file from the master and standby master for errors. If there are no unexpected errors and the machines are up, run the <code>gpinitstandby</code> utility to bring the standby online.</p>
<p>Perform a basic check to see if the master is up and functioning.</p> <p>Recommended frequency: run every 5 to 10 minutes</p> <p>Severity: CRITICAL</p>	<p>Run the following query in the <code>postgres</code> database:</p> <pre>SELECT count(*) FROM gp_segment_configuration;</pre>	<p>If this query fails the active master may be down. Try again several times and then inspect the active master manually. If the active master is down, reboot or power cycle the active master to ensure no processes remain on the active master and then</p>

		trigger the activation of the standby master.
--	--	---

Database Alert Log Monitoring

Table 2. Database Alert Log Monitoring Activities

Activity	Procedure	Corrective Actions
<p>Check for FATAL and ERROR log messages from the system.</p> <p>Recommended frequency: run every 15 minutes</p> <p>Severity: WARNING</p> <p><i>This activity and the next are two methods for monitoring messages in the log_alert_history table. It is only necessary to set up one or the other.</i></p>	<p>Run the following query in the gpperfmon database:</p> <pre>SELECT * FROM log_alert_history WHERE logseverity in ('FATAL', 'ERROR') AND logtime > (now() - interval '15 minutes');</pre>	<p>Send an alert to the DBA to analyze the alert. You may want to add additional filters to the query to ignore certain messages of low interest.</p>

Hardware and Operating System Monitoring

Table 3. Hardware and Operating System Monitoring Activities

Activity	Procedure	Corrective Actions
<p>Check disk space usage on volumes used for Greenplum Database data storage and the OS.</p> <p>Recommended frequency: every 5 to 30 minutes</p>	<p>Set up a disk space check.</p> <ul style="list-style-type: none"> Set a threshold to raise an alert when a disk reaches a percentage of capacity. The recommended 	<p>Free space on the system by removing some data or files.</p>

Severity: CRITICAL	<p>threshold is 75% full.</p> <ul style="list-style-type: none"> It is not recommended to run the system with capacities approaching 100%. 	
<p>Check for errors or dropped packets on the network interfaces.</p> <p>Recommended frequency: hourly</p> <p>Severity: IMPORTANT</p>	<p>Set up a network interface checks.</p>	<p>Work with network and OS teams to resolve errors.</p>
<p>Check for RAID errors or degraded RAID performance.</p> <p>Recommended frequency: every 5 minutes</p> <p>Severity: CRITICAL</p>	<p>Set up a RAID check.</p>	<ul style="list-style-type: none"> Replace failed disks as soon as possible. Work with system administration team to resolve other RAID or controller errors as soon as possible.
<p>Run the Greenplum <code>gpcheck</code> utility to test that the configuration of the cluster complies with current recommendations.</p> <p>Recommended frequency: when creating a cluster or adding new machines to the cluster</p> <p>Severity: IMPORTANT</p>	<p>Run <code>gpcheck</code>.</p>	<p>Work with system administration team to update configuration according to the recommendations made by the <code>gpcheck</code> utility.</p>
<p>Check for adequate I/O bandwidth and I/O skew.</p>	<p>Run the Greenplum <code>gpcheckperf</code> utility.</p>	<p>The cluster may be under-specified if data transfer rates are not similar to the</p>

<p>Recommended frequency: when create a cluster or when hardware issues are suspected.</p>	<p>following:</p> <ul style="list-style-type: none"> • 2GB per second disk read • 1 GB per second disk write • 10 Gigabit per second network read and write <p>If transfer rates are lower than expected, consult with your data architect regarding performance expectations.</p> <p>If the machines on the cluster display an uneven performance profile, work with the system administration team to fix faulty machines.</p>
--	---

Catalog Monitoring

Table 4. Catalog Monitoring Activities

Activity	Procedure	Corrective Actions
<p>Run catalog consistency checks to ensure the catalog on each host in the cluster is consistent and in a good state.</p> <p>Recommended frequency: weekly</p> <p>Severity: IMPORTANT</p>	<p>Run the Greenplum <code>gpcheckcat</code> utility in each database:</p> <pre data-bbox="991 1583 1356 1650"><code>gpcheckcat -O</code></pre>	<p>Run repair scripts for any issues detected.</p>
<p>Run a persistent table catalog check.</p> <p>Recommended frequency: monthly</p> <p>Severity: CRITICAL</p>	<p>During a downtime, with no users on the system, run the Greenplum <code>gpcheckcat</code> utility in each database:</p> <pre data-bbox="991 2032 1356 2144"><code>gpcheckcat -R persistent</code></pre>	<p>Run repair scripts for any issues detected.</p>

<p>Check for pg_class entries that have no corresponding pg_attribute entry.</p> <p>Recommended frequency: monthly</p> <p>Severity: IMPORTANT</p>	<p>During a downtime, with no users on the system, run the Greenplum gpcheckcat utility in each database:</p> <pre>gpcheckcat -R pgclass</pre>	Run the repair scripts for any issues identified.
<p>Check for leaked temporary schema and missing schema definition.</p> <p>Recommended frequency: monthly</p> <p>Severity: IMPORTANT</p>	<p>During a downtime, with no users on the system, run the Greenplum gpcheckcat utility in each database:</p> <pre>gpcheckcat -R namespace</pre>	Run the repair scripts for any issues identified.
<p>Check constraints on randomly distributed tables.</p> <p>Recommended frequency: monthly</p> <p>Severity: IMPORTANT</p>	<p>During a downtime, with no users on the system, run the Greenplum gpcheckcat utility in each database:</p> <pre>gpcheckcat -R distribution_policy</pre>	Run the repair scripts for any issues identified.
<p>Check for dependencies on non-existent objects.</p> <p>Recommended frequency: monthly</p> <p>Severity: IMPORTANT</p>	<p>During a downtime, with no users on the system, run the Greenplum gpcheckcat utility in each database:</p> <pre>gpcheckcat -R dependency</pre>	Run the repair scripts for any issues identified.

Data Maintenance

Table 5. Data Maintenance Activities

Activity	Procedure	Corrective Actions
Check for missing statistics on tables.	Check the gp_stats_missing view in each database:	Run ANALYZE on tables

	<pre>SELECT * FROM gp_toolkit.gp_stats_missing;</pre>	that are missing statistics.
<p>Check for tables that have bloat (dead space) in data files that cannot be recovered by a regular VACUUM command.</p> <p>Recommended frequency: weekly or monthly</p> <p>Severity: WARNING</p>	<p>Check the <code>gp_bloat_diag</code> view in each database:</p> <pre>SELECT * FROM gp_toolkit.gp_bloat_diag;</pre>	<p>Execute a VACUUM FULL statement at a time when users are not accessing the table to remove bloat and compact the data.</p>

Database Maintenance

Table 6. Database Maintenance Activities

Activity	Procedure	Corrective Actions
<p>Mark deleted rows in heap tables so that the space they occupy can be reused.</p> <p>Recommended frequency: daily</p> <p>Severity: CRITICAL</p>	<p>Vacuum user tables:</p> <pre>VACUUM <table>;</pre>	<p>Vacuum updated tables regularly to prevent bloating.</p>
<p>Update table statistics.</p> <p>Recommended frequency: after loading data and before executing queries</p> <p>Severity:</p>	<p>Analyze user tables. You can use the <code>analyzedb</code> management utility:</p> <pre>analyzedb -d <database> -a</pre>	<p>Analyze updated tables regularly so that the optimizer can produce efficient query execution plans.</p>

CRITICAL		
<p>Backup the database data.</p> <p>Recommended frequency: daily, or as required by your backup plan</p> <p>Severity: CRITICAL</p>	<p>Run the <code>gpbackup</code> utility to create a backup of the master and segment databases in parallel.</p>	<p>Best practice is to have a current backup ready in case the database must be restored.</p>
<p>Vacuum, reindex, and analyze system catalogs to maintain an efficient catalog.</p> <p>Recommended frequency: weekly, or more often if database objects are created and dropped frequently</p>	<ol style="list-style-type: none"> 1. VACUUM the system tables in each database. 2. Run <code>REINDEX SYSTEM</code> in each database, or use the <code>reindexdb</code> command-line utility with the <code>-s</code> option: <pre>reindexdb -s <database></pre> <ol style="list-style-type: none"> 3. ANALYZE each of the system tables: <pre>analyzedb -s pg_catalog -d <database></pre>	<p>The optimizer retrieves information from the system tables to create query plans. If system tables and indexes are allowed to become bloated over time, scanning the system tables increases query execution time. It is important to run <code>ANALYZE</code> after reindexing, because <code>REINDEX</code> leaves indexes with no statistics.</p>

Patching and Upgrading

Table 7. Patch and Upgrade Activities

Activity	Procedure	Corrective Actions
<p>Ensure any bug fixes or enhancements are applied to the kernel.</p> <p>Recommended frequency: at least every 6 months</p> <p>Severity: IMPORTANT</p>	<p>Follow the vendor's instructions to update the Linux kernel.</p>	<p>Keep the kernel current to include bug fixes and security fixes, and to avoid difficult future upgrades.</p>
<p>Install Greenplum Database minor releases, for example 5.0.x.</p> <p>Recommended frequency: quarterly</p> <p>Severity: IMPORTANT</p>	<p>Follow upgrade instructions in the Greenplum Database <i>Release Notes</i>. Always upgrade to the latest in the series.</p>	<p>Keep the Greenplum Database software current to incorporate bug fixes, performance enhancements, and feature enhancements into your Greenplum cluster.</p>

Greenplum数据库® 6.0文档

□ 管理员指南

- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问

□ 配置客户端认证

- 管理角色与权限
- 定义数据库对象
- 分布与倾斜
- 插入、更新、和删除数据
- 查询数据
- 使用外部数据
- 装载和卸载数据
- 性能管理

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

本主题讲解如何为Greenplum数据库配置客户端访问和身份验证。

当首次初始化Greenplum数据库系统时，系统包含一个预定义的超级用户角色。此角色与初始化Greenplum数据库系统的操作系统用户具有相同的名称。此角色被曾为`gpadmin`。默认情况下，系统被配置为仅允许从本地使用`gpadmin`角色连接到数据库。如果要允许其他角色访问或者允许来自远程主机的连接，则必须配置Greenplum数据库以允许此类访问。本节介绍如何为Greenplum数据库配置客户端访问和身份验证。

- 使用**TLS/SSL的LDAP**认证

可以使用LDAP服务器控制对Greenplum数据库的访问。此外，通过对`pg_hba.conf`文件条目增加参数，可使用加密来保护访问连接。

- 使用**Kerberos**认证

您可以使用Kerberos认证服务器控制对Greenplum数据库的访问。

- 为**Linux**客户端进行**Kerberos**配置

您可以配置Linux client客户端应用来连接一个已配置Kerberos认证的Greenplum数据库系统。

- 为**Windows**客户端进行**Kerberos**配置

您可以配置微软的Windows客户端应用程序来连接一个已配置Kerberos认证的Greenplum数据库系统。

Parent topic: 管理Greenplum数据库访问

允许访问Greenplum数据库

客户端访问和身份验证通过标准的PostgreSQL基于主机的身份验证文件`pg_hba.conf`控制。有关此文件的详细信息，请参阅PostgreSQL文档中的[关于`pg_hba.conf`文件](#)。

在Greenplum数据库，主实例的`pg_hba.conf`文件控制对Greenplum数据库系统的客户端访问和身份验证。在Greenplum数据库分段中也有`pg_hba.conf`文件，但这些文件已经配置为仅允许来自主机的客户端连接。这些段永远不会接受外部客户端连接，因此无需更改分段中的`pg_hba.conf`文件。

`pg_hba.conf`文件的一般格式是一组记录，每行一个。Greenplum数据库忽略空行和#注释后的内容。每行记录由许多由空格或制表符分

隔的字段组成。字段值如果被引号括起来，可以包含空格。每个远程客户端访问记录都具有以下格式：

<i>host</i>	<i>database</i>	<i>role</i>	<i>address</i>	<i>authentication-method</i>
-------------	-----------------	-------------	----------------	------------------------------

每个UNIX域套接字访问记录都采用以下格式：

<i>local</i>	<i>database</i>	<i>role</i>	<i>authentication-method</i>
--------------	-----------------	-------------	------------------------------

下表描述了每个字段的含义。

Table 1. pg_hba.conf文件

字段	描述
local	匹配使用UNIX域套接字的连接尝试。如果没有此类记录，则不允许使用UNIX域套接字连接。
host	匹配使用TCP/IP进行的连接尝试。除非服务器以适当的值启动，否则将无法进行远程TCP/IP连接的listen_addresses服务器配置参数。
hostssl	匹配使用TCP/IP进行的连接尝试，但仅限于使用SSL加密进行连接时。必须通过设置ssl服务器配置参数，在服务器启动时启用SSL。
hostnossal	匹配使用TCP/IP进行的不使用SSL的连接尝试。
指定此记录匹配的数据库名称。取值为all指定匹配所有数据库。多个数据库名称可以通过用逗号分隔。在文件名前加上@符号，可以指定一个含有数据库名的单独的文件。	
role	指定该记录匹配的数据库角色名称。取值为all指定匹配所有角色。如果指定角色是一个分组，在角色名前加+前缀，可以匹配该组内所有成员。多个角色名称可以用逗号分隔。在文件名前加入@前缀，指定匹配文件内的包

	含的所有角色名。
address	<p>指定该记录匹配的客户端机器地址。该字段包含IP地址、IP地址范围或主机名</p> <p>指定IP地址范围，使用标准数组标识范围的其实地址，紧跟着一个斜线(/)，在跟上CIDR掩码长度。掩码长度表示必须匹配的客户端IP地址的高比特位。给定的IP地址中，右边的位应为零。在IP地址、斜线/和CIDR掩码长度三者之间不能有空格。</p> <p>以这种方式指定的IPv4地址范围的典型示例是，172.20.143.89/32代表单个主机，172.20.143.0/24代表小型网络，10.6.0.0/16代表更大网络。相应的IPv6地址范围典型案例是，::1/128代表单个主机(在这种情况下是IPv6环回地址)，fe80::7a31:c1ff:0000:0000/96代表小型网络。0.0.0.0/0代表全部IPv4地址，并且::0/0代表全部IPv6地址。要指定单个主机，对于IPv4使用32位掩码长度，对于IPv6使用128位掩码长度。在网络地址中，不要省略拖尾的0数字。</p> <p>使用IPv4格式给出的记录只匹配IPv4连接，使用IPv6格式给出的记录只匹配IPv6连接，即使所表示的地址在IPv4-in-IPv6范围内。 Note: 如果主机系统C库文件不支持IPv6地址，建拒绝所有IPv6格式记录。</p> <p>如果指定了主机名(不是IP地址或IP范围的地址被当做主机名)，则将该名称与客户端IP地址的反向名称解析结果进行比较(如反向DNS查找)。主机名比较不区分大小写。如果存在匹配，如果存在匹配主机名，则对主机名执行转发名称解析(如转发DNS查找)，已检查主机名解析的任何地址是否等于客户端IP地址。如果双向都匹配，则认为该记录匹配。</p> <p>某些主机名数据库允许将IP地址与多个主机名关联，但解析IP地址是，操作系统只返回其中一个主机名。位于文件pg_hba.conf中的主机名必须是客户端IP地址，必须是客户端IP地址返回address-to-name解析结果，否则该行不会被视为匹配。</p>

	当指定在pg_hba.conf文件中的主机名，应该确保名称解析足够快。设置如nscd本地名称解析缓存是有效的。而且可以启用服务器设置参数log_hostname客户端主机名而不是日志中的IP地址。
IP-address IP-mask	该字段可以被用作CIDR地址记号的替代。不是指定掩码长度，而是在单独的列中指定实际掩码。例如，255.0.0.0表示IPv4的CIDR的掩码长度为8，而255.255.255.255表示CIDR的掩码长度为32。
authentication-method	指定建立连接时使用的认证方法。PostgreSQL 9.0支持的 认证方法 Greenplum均支持。

CAUTION:

针对更安全系统，记得移除pg_hba.conf文件中使用trust认证远程连接的配置记录。因trust认证授权能连接到服务器的任何用户，使用任何他们指定的任何角色访问数据库。针对本地UNIX-socket连接，可以使用ident认证，安全的替代trust认证。针对本地及远程TCP客户端，仍然可以使用ident认证，但是客户机主机必须运行ident服务，并且您必须信任该机器的完整性。

编辑pg_hba.conf文件

首先，设置pg_hba.conf文件，需要具有gpadmin用户权限，并且没有其他Greenplum数据库访问角色。需要编辑pg_hba.conf以使用户能够访问数据库并保护gpadmin用户。可以考虑删除具有trust身份验证的记录，因为它们允许任何有服务器访问权限的人，使用其选择的任何角色建立连接。对于本地UNIX套接字连接，请使用ident身份验证，需要操作系统用户匹配特定角色。对于本地TCP和远程TCP链接，ident认证需要客户端主机运行ident服务。针对本地连接，如127.0.0.1/28可以安装ident服务到master主机上。针对远程TCP连接，使用ident认证其安全性极低，因为需要您信任master主机上ident服务的公正性。

此示例显示了如何编辑master主机上的pg_hba.conf文件，通过加密密码认证，允许远程主机使用所有角色，访问所有数据库

编辑pg_hba.conf

1. 使用文本编辑器，打开\$MASTER_DATA_DIRECTORY/pg_hba.conf文件。

2. 针对每种允许的连接类型添加一行记录。文件中的记录按顺序读取，因此记录的顺序很重要。典型的，较早的记录将具有较严格的连接匹配参数和较弱的身份验证方法，而较靠后记录将具有更宽松的匹配参数和更抢的身份验证方法。例如：

```
# 允许gpadmin用户本地访问所有数据库
# 使用ident认证
local all gpadmin ident sameuser
host all gpadmin 127.0.0.1/32 ident
host all gpadmin ::1/128 ident
# 允许'dba'角色，通过192.168.x.xIP地址，使用md5加密密码，授权用户访问任意数据库。
# 注意使用SHA-256加密，替换如下行记录中md5密码。
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配置文件以使更改生效：

```
$ gpstop -u
```

Note: 可以设置对象特权来控制数据库访问，具体参考[管理对象权限](#)中所描述的内容。pg_hba.conf文件只是控制谁可以发起数据库回话以及如何认证这些链接。

限制并发连接

Greenplum数据库以每个连接为基础分配资源，因此建议设置允许的最大连接数。

要限制Greenplum数据库系统的活动并发会话数，可以配置max_connections服务器配置参数。这是一个local参数，这意味着必须在master节点、standby master节点、每个segment实例（包括primary和mirror）的postgresql.conf文件中设置该参数。中主服务器，备用主服务器和每个段实例（主服务器和镜像服务器）的文件。建议在segment节点的max_connections参数值是master节点的5-10倍。

当设置max_connections时，必须设置依赖参

数max_prepared_transactions。该值至少与master节点上的max_connections取值相等。segment实例节点也设置相同的值。

例如：

- 在\$MASTER_DATA_DIRECTORY/postgresql.conf文件（包括standby master节点）

```
max_connections=100
max_prepared_transactions=100
```

- 在所有segment实例节点SEGMENT_DATA_DIRECTORY/postgresql.conf文件

```
max_connections=500
max_prepared_transactions=100
```

以下步骤使用Greenplum数据库工具 gpconfig设置参数值。

有关更多gpconfig的信息，可以参考*Greenplum*数据库工具指南。

更改允许的连接数

1. 以Greenplum数据库管理员身份，登录Greenplum数据库的master节点主机，使用source命令加载\$GPHOME/greenplum_path.sh文件信息。
2. 设置max_connections参数取值。此gpconfig命令将segment实例节点上的参数值设置为1000，将master节点上的参数值设置为200。

```
$ gpconfig -c max_connections -v 1000 -m 200
```

该参数值segement节点上的必须大于master节点上的。推荐segment节点的max_connections参数值是master节点的5-10倍。

3. 设置max_prepared_transactions参数取值。此gpconfig命令将master节点和segment实例节点上的取值都设为200。

```
$ gpconfig -c max_prepared_transactions -v 200
```

segment节点的max_prepared_transactions参数值必须大于等于master节点的max_connections参数值。

4. 停止并重启Greenplum数据库系统。

```
$ gpstop -r
```

5. 可以使用gpconfig -s选项，检查master和segment节点的参数值。此gpconfig命令显示max_connections参数的取值。

```
$ gpconfig -s max_connections
```

Note: 调高这些参数的取值，可能会导致Greenplum数据库需要更多的共享内存。为了缓和这种影响，考虑调小其他内存相关的参数，如gp_cached_segworkers_threshold参数。

加密客户端/服务器连接

为连接到Greenplum数据库的客户端启用SSL，以加密客户端和数据库之间通过网络传递的数据。

Greenplum数据库原生支持支持客户端和master服务器之间的SSL连接。SSL连接可以防止第三方对数据包进行嗅探，还可以防止中间人攻击。只要客户端连接通过不安全的链接就应该使用SSL，并且在使用客户端证书认证时必须使用SSL。

要启用SSL，需要在客户端和master服务器系统上安装OpenSSL。通过在master节点的postgresql.conf文件，设置服务器配置参数ssl=on，可以让Greenplum数据库启动时启用SSL。当以SSL模式启动时，服务器将会在master节点的数据目录中查找服务器私钥server.key文件和服务器证书server.crt文件。在开启SSL的Greenplum数据库系统启动之前，确保这些文件已经正确地设置好。

Important: 请勿使用密码保护私钥。服务器不会提示输入私钥的密码，如果需要密码，则数据库会启动失败并显示错误。

可以使用自签名证书进行测试，但在生产中应使用由数字证书认证机构（CA）签名的证书，以便客户端可以验证服务器的身份。可以使用全球CA或本地CA。如果所有客户端在组织内部本地，则建议使用本地CA。

创建一个仅用于测试没有密码的自签名证书

要为服务器快速创建自签名证书以进行测试，请使用如下OpenSSL命令：

```
# openssl req -new -text -out server.req
```

输入提示所要求的信息。请务必输入本地主机名作为*Common Name*。私钥保护密码可以留空。

该程序将生成一个受密码保护的密钥，并且不接受长度小于四个字符的密码。

要把这个证书用于Greenplum数据库，用下列命令移除该密码：

```
# openssl rsa -in privkey.pem -out server.key  
# rm privkey.pem
```

在提示解锁现有密钥时输入旧密码。

然后，输入以下命令将证书转换为自签名证书，并将密钥和证书复制到服务器，并保存到可以查找到它们的位置。

```
# openssl req -x509 -in server.req -text -key server.key -  
out server.crt
```

最后，使用以下命令更改密钥的权限。如果权限限制不这么严格，则服务器将拒绝该文件。

```
# chmod og-rwx server.key
```

有关如何创建服务器私钥和证书的更多详细信息，请参阅[OpenSSL文档](#)所示。

Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统
 - 管理Greenplum数据库访问
 - 配置客户端认证
 - 管理角色与权限
- 定义数据库对象
- 分布与倾斜
- 插入、更新、和删除数据
- 查询数据
- 使用外部数据
- 装载和卸载数据
- 性能管理

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

Greenplum数据库授权机制存储访问数据库中数据库对象的角色和权限，并使用SQL语句或命令行实用程序进行管理。

Greenplum数据库使用*roles*管理数据库访问权限。角色的概念包含用户和组的概念。一个角色可以是一个数据库用户、一个数据库组或者两者间距。角色可以拥有数据库对象（例如表），并可以将这些对象上的权限赋予其他角色，依此来控制对对象的访问。角色可以是其他角色的成员，因此成员角色可以继承其父角色的对象权限。

每个Greenplum数据库系统都包含一组数据库角色（用户和组）。这些角色与运行服务器的操作系统管理的用户和组相互独立。但是，为方便起见，可能希望维护操作系统用户名和Greenplum数据库角色名之间的关系，因为许多客户端应用程序使用当前操作系统用户名作为其默认值。

在Greenplum数据库中，用户通过master实例登录并连接，然后master实例验证其角色和访问权限。然后，master服务器以当前登录的角色的身份，将命令发布到幕后的segment实例。

角色在系统级别定义，这意味着它们对系统中的所有数据库都有效。

为了引导Greenplum数据库系统，新初始化的系统始终包含一个预定义的超级用户角色（也称为系统用户）。该角色将与初始化Greenplum数据库系统的操作系统用户具有相同的名称。通常，此角色已命名为gpadmin。为了创建更多角色，您首先必须以此初始角色进行连接。

Parent topic: [管理Greenplum数据库访问](#)

角色和权限的安全最佳实践

- 保护gpadmin系统用户 Greenplum需要UNIX用户标识来安装和初始化Greenplum数据库系统。在Greenplum文档中，该系统用户称为gpadmin。该gpadmin是Greenplum数据库中的默认数据库超级用户，也是Greenplum安装文件系统及基础数据文件的所有者。此默认管理员帐户是Greenplum数据库设计的基础。没有它，系统无法运行，并且无法限制此gpadmin用户ID的访问权限。针对特定目的，使用角色来管理谁有权访问数据库。应该只在保护任务时使用gpadmin账户，如扩展和升级。以此用户ID登录Greenplum主机的任何人都可以读取、更改或删除任何数据，包括系统目录数据和数据库访问权限。因此，保护gpadmin用户ID并仅提供对基本系

统管理员的访问权限非常重要。管理员只应执行某些系统维护任务（如升级或扩展）时以gpadmin账户登录Greenplum。数据库用户永远不应登录为gpadmin，ETL或生产工作永远不应该以gpadmin运行。

- 为登录的每个用户分配不同的角色。为了记录和审计，允许每个允许登录Greenplum数据库的用户拥有自己的数据库角色。对于应用程序或Web服务，考虑为每个应用程序或服务创建不同的角色。参阅[创建新角色 \(用户\)](#)。
- 使用组来管理访问权限。参阅[角色成员](#)。
- 限制具有**SUPERUSER**角色属性的用户。只有系统管理员才能获得超级用户权限。参阅[变更角色属性](#)。

创建新角色 (用户)

用户级角色被视为可以登录数据库并启动数据库会话的数据库角色。因此，当你使用CREATE ROLE命令创建新的用户级角色时，必须指定LOGIN权限。例如：

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

数据库角色可以具有许多属性，这些属性定义角色可以在数据库中执行的任务类型。可以在创建角色时设置这些属性，也可以稍后使用ALTER ROLE命令。关于可以设置的角色属性的说明，参阅[Table 1](#)。

变更角色属性

数据库角色可以具有许多属性，这些属性定义角色可以在数据库中执行的任务类型。

Table 1. 角色属性

属性	描述
SUPERUSER NOSUPERUSER	确定角色是否为超级用户。您必须自己是超级用户才能创建新的超级用户。默认值是NOSUPERUSER。
CREATEDB NOCREATEDB	确定是否允许角色创建数据库。默认值是NOCREATEDB。

CREATEROLE NOCREATEROLE	确定是否允许角色创建和管理其他角色。默认值是NOCREATEROLE。
INHERIT NOINHERIT	确定角色是否继承其所属角色的权限。具有INHERIT属性的角色继承可以自动使用已授予其直接或间接成员的所有角色的任何数据库权限。默认值是INHERIT。
LOGIN NOLOGIN	确定是否允许角色登录。具有该LOGIN属性的角色可以被认为是用户。没有此属性的角色对于管理数据库权限（组）非常有用。默认值是NOLOGIN。
CONNECTION LIMIT <i>connlimit</i>	如果角色可以登录，则指定角色可以使用的并发连接数。默认值-1表示没有限制。
CREATEEXTTABLE NOCREATEEXTTABLE	确定是否允许角色创建外部表。默认值是NOCREATEEXTTABLE。具有该CREATEEXTTABLE属性的角色，默认外部表类型是可读的，注意使用文件或执行的外部表只能由超级用户创建。
PASSWORD ' <i>password</i> '	设置角色的密码。如果您不打算使用密码身份验证，则可以省略此选项。如果未指定密码，则密码将设置为空，并且该用户的密码验证将始终失败。可以选择将空密码明确写为PASSWORD NULL。
ENCRYPTED UNENCRYPTED	<p>控制是否将新密码在pg_authid系统目录中存储为哈希字符串。如果没有指定ENCRYPTED，或者指定UNENCRYPTED，则默认行为由password_encryption配置参数决定，该参数默认值为on。</p> <p>如果提供<i>password</i>字符串已经是哈希格式，无论是否指定ENCRYPTED或UNENCRYPTED都原样存储。</p> <p>有关保护登录密码的其他信息，参阅保护Greenplum数据库中的密码。</p>
VALID UNTIL ' <i>timestamp</i> '	设置角色密码失效的日期和时间。如果省略，密码将始终有效。
RESOURCE QUEUE <i>queue_name</i>	将角色分配给指定的资源队列以进行工作负载管理。任何角色问题的声明都受资源

	队列限制的约束。注意RESOURCE QUEUE属性不可继承；必须为每个用户级别（LOGIN）角色分别设置。
DENY {deny_interval deny_point}	限制时间间隔期间的访问，按日期或日期时间指定。更多信息，参阅 基于时间的身份验证 。

可以在创建角色时设置这些属性，也可在使用ALTER ROLE命令。例如：

```
=# 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';
```

因服务的特定设置，角色还可以具有特定于角色的默认值。例如，要为角色设置默认方案搜索路径：

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

角色成员

将用户组合在一起以便于管理对象权限通常很方便：这样，可以将权限授予整个组或从组中撤销。在Greenplum数据库中，通过创建表示组的角色，然后将组角色的成员身份授予单个用户角色来完成的。

使用CREATE ROLE此SQL创建一个新的组角色。例如：

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

一旦组角色存在后，可以使用GRANT和REVOKE命令，来添加和删除成员（用户角色）。例如：

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

为了管理对象权限，您只能为组级角色授予适当的权限（参阅[Table 2](#)）。然后，成员用户角色将继承组角色的对象权限。例如：

```
=# GRANT ALL ON TABLE mytable TO admin;
```

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

角色属性LOGIN、SUPERUSER、CREATEDB、CREATEROLE、CREATEEXTTABLE和RESOURCE QUEUE 永远不会像数据库对象上的普通权限那样被继承。用户成员实际上必须SET ROLE具有这些属性的特定角色，才能使用该属性。在上面的例子中，我们给出了CREATEDB和CREATEROLE到了admin角色。如果sally是admin角色的成员，她可以发出以下命令来承担父角色的角色属性：

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

管理对象权限

当对象（表、视图、序列、数据库、函数、语言、模式或表空间）时，会为其分配一个所有者。所有者通常是执行创建语句的角色。对于大多数类型的对象，初始状态是只有所有者（或超级用户）可以对该对象执行任何操作。要允许其他角色使用它，必须授予权限。Greenplum Database支持每种对象类型的以下权限：

Table 2. 对象权限

对象类型	权限
表、视图、序列	SELECT INSERT UPDATE DELETE RULE ALL
外部表	SELECT RULE ALL

数据库	CONNECT CREATE TEMPORARY TEMP ALL
函数	EXECUTE
过程语言	USAGE
模式	CREATE USAGE ALL
自定义协议	SELECT INSERT UPDATE DELETE RULE ALL

Note: 您必须单独为每个对象授予权限。例如，授予数据库上ALL权限，并不授予对该数据库中的对象的完全访问权限。它只授予数据库级别的（CONNECT、CREATE、TEMPORARY）到数据库本身的权限。

使用GRANT此SQL命令为对象赋予指定的角色权限。例如，要授予名为jsmith的角色在名为mytable的表上插入权限：

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

同样，授予jsmith仅为名为table2表中的名为col1的查询权限，表2：

```
=# GRANT SELECT (col1) on TABLE table2 TO jsmith;
```

要撤消权限，使用REVOKE命令。例如：

```
=# REVOKE ALL 权限 ON mytable FROM jsmith;
```

也可以使用DROP OWNED和REASSIGN OWNED命令 用于管理已弃用角色所拥有的对象（注意：只有对象的所有者或超级用户才能删除对象或重新分配所有权）。例如：

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

模拟行级访问控制

Greenplum数据库不支持行级访问或行级标记的安全性。可以使用视图来限制所选行来模拟行级访问。可以通过向表中添加额外的列来存储敏感度信息，然后使用视图来控制基于此列的行级访问来模拟行级标签。然后，可以授予角色访问视图而不是基本表的权限。

加密数据

Greenplum数据库安装了一个可选的加密解密函数模块pgcrypto。该pgcrypto函数允许数据库管理员以加密形式存储某些数据列。这为敏感数据增加了额外的保护层，没有加密密钥，任何人都无法读取以加密形式存储在Greenplum数据库中的数据，也无法直接从磁盘读取数据。

Note: 该pgcrypto数据库服务器内运行，这意味着在pgcrypto和客户端应用程序之间，所有数据和密码以明文形式移动。为获得最佳安全性，还应考虑在客户端和Greenplum主服务器之间使用SSL连接。要使用pgcrypto函数，在每个要使用此函数的数据库中，注册pgcrypto扩展。例如：

```
$ psql -d testdb -c "CREATE EXTENSION pgcrypto"
```

有关各个函数的更多信息，参阅PostgreSQL文档中的[pgcrypto](#)。

保护Greenplum数据库中的密码

在其默认配置中，Greenplum Database以MD5哈希值形式，将登录用户密码保存在pg_authid系统目录，而不是保存明文密码。任何能

够查看此pg_authid表的人都可以看到哈希字符串，但没有密码。这还可确保在将数据库转储到备份文件时隐藏密码。

使用以下任何命令设置密码时执行哈希函数

- CREATE USER *name* WITH ENCRYPTED PASSWORD
'*password*'
- CREATE ROLE *name* WITH LOGIN ENCRYPTED PASSWORD
'*password*'
- ALTER USER *name* WITH ENCRYPTED PASSWORD
'*password*'
- ALTER ROLE *name* WITH ENCRYPTED PASSWORD
'*password*'

当password_encryption系统配置参数为on (这是默认值) 时，ENCRYPTED关键字可以被省略。当命令未指定ENCRYPTED或UNENCRYPTED时，password_encryption配置参数决定是存储明文密码还是哈希密码。

Note: SQL命令语法和password_encryption配置变量，包括术语加密。但是密码在技术上并不是加密。它们是被哈希，因此无法解密。

哈希是在通过将明文密码和角色名称串起来，然后计算的。MD5哈希生成一个前缀为md5字符的32字节十六进制字符串。哈希密码保存在pg_authid系统表rolpassword的列。

尽管不被推荐，但密码可以以明文形式保存在数据库中，通过UNENCRYPTED命令关键词，或者将password_encryption配置变量参数设置为off。注意，更改配置值不会影响现有密码，只会影响新创建或更新的密码。

要全局地设置password_encryption，在命令行中使用gpadmin用户执行这些命令：

```
$ gpconfig -c password_encryption -v 'off'
$ gpstop -u
```

要在会话中设置password_encryption，使用SQL的SET命令。例如：

```
=# SET password_encryption = 'on';
```

密码可以使用SHA-256哈希算法而不是默认的MD5哈希算法进行哈希。该算法生成一个前缀为sha256字符的64字节十六进制字符串。

Note:

虽然SHA-256使用更强的加密算法并生成更长的哈希字符串，但它不能被用于MD5认证方法。要使用SHA-256密码哈希，必须在pg_hba.conf配置文件中，将认证方法设置为password，以便明文密码被发送给Greenplum数据库。由于明文密码是通过网络发送的，因此在使用SHA-256时使用SSL进行客户端连接非常重要。另一方面，默認md5认证方法，身份验证方法在将密码发送到Greenplum数据库之前对密码进行两次哈希处理。一次是在密码和角色名称上，然后再次在客户端和服务器之间共享盐值，因此明文密码永远不会发送到网络。

要启用SHA-256哈希，更改password_hash_algorithm配置参数的默认值从md5为sha-256。该参数可以全局设置或在会话级别设置。要全局地设置password_hash_algorithm，在命令行中使用gpadmin用户执行这些命令：

```
$ gpconfig -c password_hash_algorithm -v 'sha-256'  
$ gpstop -u
```

要全局地设置password_hash_algorithm，SQL的SET命令。例如：

```
=# SET password_hash_algorithm = 'sha-256';
```

基于时间的身份验证

Greenplum数据库使管理员可以按角色限制对特定时间的访问。使用CREATE ROLE或ALTER ROLE命令，指定基于时间的约束。

有关详细信息，参阅*Greenplum*数据库安全配置指南.

Greenplum数据库® 6.0文档

□ 管理员指南

- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象

创建和管理数据库

创建和管理表空间

创建和管理SCHEMA

创建和管理表

选择表存储模型

对大型表分区

创建和使用序列

在Greenplum数据库中使用索引

创建和管理视图

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

□ 性能管理

一个Greenplum数据库是Greenplum数据库的一个单一实例。可能会安装有多个单独的Greenplum数据库系统，但是通常只会用环境变量设置选择其中一个。详情请咨询用户的Greenplum管理员。

一个Greenplum数据库系统中可以有多个数据库。这与某些数据库管理系统（如Oracle）不同，那些系统中数据库实例就是数据库。尽管用户可以在一个Greenplum系统中创建很多数据库，但是客户端程序一次只能连接上并且访问一个数据库，因此用户无法跨数据库进行查询。

Parent topic: [定义数据库对象](#)

关于模板和默认数据

Greenplum数据库提供了一些模板数据库和一个默认数据库，*template1*, *template0*和*postgres*。

默认情况下，每个新创建的数据库都是基于一个模板数据库。Greenplum数据库默认使用*template1*作为模板，除非手动指定另一个模板。不推荐在在模板数据库*template1*里创建对象。否则，对象将会出现在所有使用默认模板创建的数据库里。

Greenplum数据库内部使用另外一个模板数据库*template0*。不要删除和更改*template0*。你可以使用*template0*创建一个完全干净的，仅包含Greenplum数据库初始化时预定义好的标准对象的数据库。

你可以使用*postgres*作为第一次连接Greenplum数据库时使用的数据库。Greenplum数据库使用*postgres*作为管理连接的默认数据库。例如，*postgres*被启动进程，全局死锁检测进程和FTS（故障恢复服务）进程在访问catalog表时使用。

创建一个数据库

CREATE DATABASE命令会创建一个新的数据库。例如：

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

要创建一个数据库，用户必须具有创建一个数据库的特权或者是一

个Greenplum数据库超级用户。如果用户没有正确的特权，用户就不能创建数据库。可以联系用户的Greenplum数据库管理员为用户授予必要的特权或者替用户创建一个数据库。

用户还可以使用客户端程序`createdb`来创建一个数据库。例如，在一个使用提供的主机名和端口连接到Greenplum数据库命令行终端中运行下列命令，它将会创建一个名为`mydatabase`的数据库：

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

上述主机名和端口必须匹配所安装的Greenplum数据库系统的主机名和端口。

一些对象（如角色）会在一个Greenplum数据库系统的所有数据库之间共享。其他对象（例如用户创建的表）则只出现在创建它们的数据库中。

Warning: `CREATE DATABASE`命令不是事务性的。

克隆一个数据库

默认情况下，一个新数据库通过克隆标准系统数据库模板`template1`而创建。其实在创建新数据库时，任何一个数据库都可以被用作模板，这样就提供了“克隆”或者复制一个现有数据库及其所包含的所有对象和数据的能力。例如：

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

创建一个不同Owner的数据库

在创建数据库时可以指定另一个数据库Owner：

```
=> CREATE DATABASE new_dbname WITH owner=new_user;
```

查看数据库的列表

如果用户在使用`psql`客户端程序，用户可以使用`\l`命令来显示用户的Greenplum数据库系统中的数据库和模板的列表。如果使用的是

另一种客户端程序并且用户是超级用户，用户可以从 pg_database 系统目录表查询数据库的列表。例如：

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

修改一个数据库

ALTER DATABASE 命令可以修改数据库的属性，例如拥有者、名称或者默认配置属性。例如，下面的命令会修改一个数据库的默认方案搜索路径（search_path 配置参数）：

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

要修改一个数据库，用户必须是该数据库的拥有者或者超级用户。

删除一个数据库

DROP DATABASE 命令删除一个数据库。它会移除该数据库的系统目录项并且删除该数据库在磁盘上的目录及其中包含的数据。要删除一个数据库，用户必须是该数据库的拥有者或者超级用户，并且当用户或者其他正连接到该数据库时不能删除它。在删除一个数据库时，可以连接到 postgres（或者另一个数据库）。例如：

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

用户也可以使用客户端程序 dropdb 来删除一个数据库。例如，下面的命令会用给出的主机名和端口连接到 Greenplum 数据库并且删除数据库 *mydatabase*：

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

Warning: 删除数据库不能被撤销。

DROP DATABASE 命令不是事务性的。

Greenplum数据库® 6.0文档

- 管理员指南

- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象

创建和管理数据库

创建和管理表空间

创建和管理SCHEMA

创建和管理表

选择表存储模型

对大型表分区

创建和使用序列

在Greenplum数据库中
使用索引

创建和管理视图

分布与倾斜

插入, 更新, 和删除数据

- 查询数据

- 使用外部数据

- 装载和卸载数据

- 性能管理

Greenplum database 5 管理员

表空间允许数据库管理员在每台机器上拥有多个文件系统并且决定如何最好地使用物理存储来存放数据库对象。表空间允许用户为频繁使用和不频繁使用的数据库对象分配不同的存储，或者在特定的数据库对象上控制I/O性能。例如，把频繁使用的表放在使用高性能固态驱动器（SSD）的文件系统上，而把其他表放在标准的磁盘驱动器上。

表空间需要一个主机文件系统位置来存储其数据库文件。在Greenplum数据库中，文件系统位置必须存在于包括运行master,standby master和每个primary和mirror的所有主机上。

表空间是Greenplum数据库系统对象（全局对象），如果有权限的话，可以使用任何数据库中的表空间。

Parent topic: [定义数据库对象](#)

创建表空间

CREATE TABLESPACE命令创建一个表空间。例如：

```
CREATE TABLESPACE fastspace LOCATION '/fastdisk/gpdb';
```

数据库超级用户定义表空间并通过GRANTCREATE 命令赋予数据库用户访问权限。例如：

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

使用表空间来存储数据库对象

在一个表空间上拥有CREATE特权的用户可以在其中创建数据库对象，例如表、索引和数据库。命令是：

```
CREATE TABLE tablename(options) TABLESPACE spacename
```

例如，下列命令在表空间`space1`中创建一个表：

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

用户也可以使用`default_tablespace`参数为没有指定表空间的CREATE TABLE和CREATE INDEX命令指定默认表空间：



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

如果在创建对象时没有指定TABLESPACE， The tablespace associated with a database stores that database's system catalogs, temporary files created by server processes using that database, and is the default tablespace selected for tables and indexes created within the database, if no TABLESPACE is specified when the objects are created. 如果在创建数据库时没有指定表空间，则使用与该数据库模板库一样的表空间。

如果有足够的权限的话，可以使用任何数据库的表空间。

查看已有的表空间

每个Greenplum数据库系统有如下默认表空间。

- pg_global 用于共享系统的catalogs。
- pg_default， 默认表空间。由*template1*和*template0*数据库使用。

这些表空间使用系统的默认文件空间，其数据目录位置在系统初始化时被创建。

要查看表空间信息，请从pg_tablespacecatalog表获取表空间的对象ID(OID)，然后使用gp_tablespace_location()函数显示表空间路径。下面是一个包含有一个用户定义的表空间myspace的例子：

```
SELECT oid, * FROM pg_tablespace ;

 oid | spcname   | spcowner | spcacl | spcoptions
-----+-----+-----+-----+
 1663 | pg_default |        10 |       |
 1664 | pg_global  |        10 |       |
 16391 | myspace    |        10 |       |
(3 rows)
```

myspace表空间的OID是16391。运行gp_tablespace_location()显示了系统中包含两个节点和master的表空间位置。

```
# SELECT * FROM gp_tablespace_location(16391);
 gp_segment_id | tblspc_loc
-----+-----
 0 | /data/mytblspace
 1 | /data/mytblspace
 -1 | /data/mytblspace
(3 rows)
```

这个查询使用gp_tablespace_location()和catalog表gp_segment_configuration显示包含myspace表空间文件系统路径的节点实例信息。

```
WITH spc AS (SELECT * FROM gp_tablespace_location(16391))
  SELECT seg.role, spc_gp_segment_id as seg_id, seg.hostname,
seg.datadir, tblspc_loc
    FROM spc, gp_segment_configuration AS seg
   WHERE spc_gp_segment_id = seg.content ORDER BY seg_id;
```

这是一个在单一宿主机上包含两个节点和master的测试系统的信息。

role	seg_id	hostname	datadir	tblspc_loc
p	-1	testhost	/data/master/gpseg-1	/data/mytblspace
p	0	testhost	/data/data1/gpseg0	/data/mytblspace
p	1	testhost	/data/data2/gpseg1	/data/mytblspace
(3 rows)				

删除表空间

要删除表空间，必须是表空间的所有者或者超级用户。直到所有依赖该表空间的对象都被删除才可以删除该表空间。

DROP TABLESPACE命令删除一个空的表空间。

Note: 无法删除一个非空或存储了临时或事务文件的表空间。

移动临时或事务文件的位置

您可以将临时文件或事务文件移动到特定的表空间，以在运行查询，创建备份和更顺序地存储数据时提高数据库性能。

Greenplum数据库服务器配置参数 `temp_tablespaces` 控制哈希聚合和哈希连接查询的临时表和临时溢出文件的位置。（用于排序大型数据集等目的的临时文件与`<data_dir>/<seg_ID>/base/pgsql_tmp`中的常规段数据一起存储。）

当CREATE命令没有显式指定表空间时，`temp_tablespaces`指定了创建临时对象时使用的表空间（临时表和临时表上的index）。

另请注意有关临时文件或事务文件的以下信息：

- 尽管您可以使用相同的表空间来存储其他类型的文件，但只能将一个表空间专用于临时文件或事务文件。
- 无法删除一个被临时文件使用的表空间。

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

创建和管理数据库

创建和管理表空间

创建和管理SCHEMA

创建和管理表

选择表存储模型

对大型表分区

创建和使用序列

在Greenplum数据库中
使用索引

创建和管理视图

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

□ 性能管理

Schema从逻辑上组织一个数据库中的对象和数据。 Schema允许用户在同一个数据库中拥有多于一个对象（例如表）具有相同的名称而不发生冲突，只要把它们放在不同的Schema中就好。

Parent topic: [定义数据库对象](#)

默认的 "Public" SCHEMA

每个数据库都有一个名为*public*的schema。 如果用户没有创建任何schema， 对象会被创建在这个*public*的schema中。 所有的数据库角色（用户）都在*public* schema中拥有CREATE和USAGE特权。 在用户创建一个schema时， 用户应该为允许访问该schema的用户授予特权。

创建一个SCHEMA

可以使用CREATE SCHEMA命令来创建一个新的schema。 例如：

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

要在一個schema中创建或者访问对象，需要写一个由schema名和表名构成的限定名，两者之间用点号隔开。 例如：

```
myschema.table
```

有关访问一个schema的信息请见[SCHEMA搜索路径](#)。

用户可以创建一个由他人拥有的schema，例如用来把用户的活动限定在定义良好的名字空间内。 语法是：

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

SCHEMA搜索路径



要在数据库中指定一个对象的位置，请使用schema限定的名称。 例如：

Greenplum database 5管理员

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

用户可以设置search_path配置参数来指定在其中搜索对象的可用schema的顺序。在该搜索路径中第一个列出的方案会成为默认schema。如果没有指定方案，对象会被创建在默认schema中。

设置SCHEMA搜索路径

search_path配置参数设置schema搜索顺序。ALTER DATABASE命令可以设置搜索路径。例如：

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

用户也可以使用ALTER ROLE命令为特定的角色（用户）设置search_path。例如：

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

查看当前SCHEMA

使用current_schema()函数可以查看当前的schema。例如：

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

使用SHOW命令可以查看当前的搜索路径。例如：

```
=> SHOW search_path;
```

删除一个SCHEMA

使用DROP SCHEMA命令可以删除一个schema。例如：

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

默认情况下，在能够删除一个schema前，它必须为空。要删除一个schema连同其中的所有对象（表、数据、函数等等），可以使用：

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

系统SCHEMA

下列系统级schema存在于每一个数据库中：

- `pg_catalog`包含着系统目录表、内建数据类型、函数和操作符。即便在schema搜索路径中没有显式地提到它，它也总是schema搜索路径的一部分。
- `information_schema`有一个包含数据库中对象信息的视图集合组成。这些视图以一种标准化的方式从系统目录表中得到系统信息。
- `pg_toast`存储大型对象，如超过页面尺寸的记录。这个schema由Greenplum数据库系统内部使用。
- `pg_bitmapindex`存储位图索引对象，例如值的列表。这个schema由Greenplum数据库系统内部使用。
- `pg_aoseg`存储追加优化表对象。这个schema由Greenplum数据库系统内部使用。
- `gp_toolkit`是一个管理用途的schema，它包含用户可以用SQL命令访问的外部表、视图和函数。所有的数据库用户都能访问`gp_toolkit`来查看和查询系统日志文件以及其他系统指标。

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

创建和管理数据库

创建和管理表空间

创建和管理SCHEMA

创建和管理表

选择表存储模型

对大型表分区

创建和使用序列

在Greenplum数据库中
使用索引

创建和管理视图

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

□ 性能管理

Greenplum数据库的表与任何一种关系型数据库中的表类似，不过其表中的行被分布在系统中的不同Segment上。当用户创建一个表时，用户会指定该表的分布策略。

创建一个表

`CREATE TABLE`命令创建一个表并且定义它的结果。当用户创建一个表示，用户需要定义：

- 该表的列以及它们的数据类型。参见[选择列的数据类型](#)。
- 任何用于限制列或者表中能包含的数据的表或者列约束。参见[设置表和列约束](#)。
- 表的分布策略，这决定了Greenplum数据库如何在Segment之间划分数据。参见[选择表分布策略](#)。
- 表存储在磁盘上的方式。参见[选择表存储模型](#)。
- 大型表的表分区策略。参见[创建和管理数据库](#)。

选择列的数据类型

一个列的数据类型决定了该列能包含的数据的类型。选择的数据类型应使用最少的空间，但仍能容纳用户的数据并且能最好地约束数据。例如，对字符串使用character数据类型，对于日期使用date或者timestamp数据类型，而对数字使用numeric数据类型。

对于包含文本数据的表列，应指定数据类型为VARCHAR或者TEXT。不推荐指定数据类型为CHAR。在Greenplum数据库中数据类型VARCHAR或者TEXT会把加在数据后面的边距（在最后一个非空白字符后面增加的空白字符）处理为有效字符，而数据类型CHAR不会这样做。关于character数据类型的信息，请见Greenplum数据库参考指南中的`CREATE TABLE`命令。

使用能容纳用户的数字型数据的且允许未来扩张的最小数字数据类型。例如，为适合INT或SMALLINT的数据使用BIGINT会浪费存储空间。如果用户预期用户的数据值将会随着时间扩张，应考虑到在装载大量数据后从较小的数据类型更改成较大的数据类型...有很大的代价。例如，如果用户当前的数据值适合SMALLINT，但是很可能值会扩张，这样INT就是更好的长期选择。

对用户计划要用在交叉表连接中的列使用相同的数据类型。交叉表连接通常使用一个表中的主键和其他表中的外键。当数据类型不同时，数据库必须转换其中之一以便数据值能被正确地比较，这会增加不必要的开销。

Greenplum数据库为用户提供了丰富的本地数据类型集合。有关内建数据类型的信息请见*Greenplum*数据库参考指南。

设置表和列约束

用户可以在列和表上定义约束来限制表中的数据。Greenplum数据库支持和PostgreSQL相同的约束，但是有一些限制，包括：

- CHECK约束只能引用它所在的表。
- UNIQUE和PRIMARY KEY约束必须和它们所在表的分布键和分区键（如果有）兼容。
Note: 在追加优化表上不允许UNIQUE和PRIMARY KEY约束，因为追加优化表上不允许这些约束创建的UNIQUE索引。
- 允许FOREIGN KEY约束，但不会被强制。
- 用户在分区表上定义的约束将作为整体应用到分区表上。用户不能在该表的单独的部分上定义约束。

检查约束

检查约束允许用户指定一个特定列中的值必须满足一个布尔（真值）表达式。例如，要求正的产品价格：

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

非空约束

非空约束指定一个列不能有空值。非空约束总是被写作为列约束。例如：

```
=> CREATE TABLE products
    ( product_no integer NOT NULL,
```

```
name text NOT NULL,
      price numeric );
```

唯一约束

唯一约束确保一列或者一组列中包含的数据对于表中所有的行都是唯一的。该表必须是哈希分布或复制表（不可以是DISTRIBUTED RANDOMLY）。如果表是哈希分布的，约束列必须是该表的分布键列（或者是一个超集）。例如：

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

主键

主键约束是一个UNIQUE约束和一个NOT NULL约束的组合。该表必须是哈希分布（非DISTRIBUTED RANDOMLY）的，并且约束列必须是该表的分布键列（或者是一个超集）。如果一个表具有主键，这个列（或者这一组列）会被默认选中为该表的分布键。例如：

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

外键

不支持外键。用户可以声明它们，但是参照完整性不会被实施。

外键约束指定一列或者一组列中的值必须匹配出现在另一个表的某行中的值，以此来维护两个相关表之间的参照完整性。参照完整性检查不能在一个Greenplum数据库的分布表段之间实施。

选择表分布策略

所有的Greenplum数据库表都会被分布。当用户创建或者修改一个表时，用户可以选择地指定DISTRIBUTED BY（哈希分布），DISTRIBUTED RANDOMLY（随机分布），或DISTRIBUTED REPLICATED（全分布）来决定该表的行分布。

Note: 如果创建表时没有指定DISTRIBUTED BY，Greenplum数据库服务器配置参数gp_create_table_random_default_distribution控制表的分布策略。

更多有关该参数的信息，请见*Greenplum*数据库参考指南的“服务器配置参数”部分。

在决定表分布策略时，请考虑以下几点。

- 均匀数据分布 — 为了最好的性能，所有的Segment应该包含等量的数据。如果数据不平衡或者倾斜，具有更多数据的Segment就必须做更多工作来执行它那一部分的查询处理。请选择对于每一个记录都唯一的分布键，例如主键。
- 本地和分布式操作 — 本地操作比分布式操作更快。在Segment层面上，如果与连接、排序或者聚集操作相关的工作在本地完成，查询处理是最快的。在系统层面完成的工作要求在Segment之间分布元组，其效率会低些。当表共享一个共同的分布键时，在它们共享的分布键列上的连接或者排序工作会在本地完成。对于随机分布策略来说，本地连接操作就行不通了。
- 均匀查询处理 — 为了最好的性能，所有的Segment应该处理等量的查询负载。如果一个表的数据分布策略与查询谓词匹配不好，查询负载可能会倾斜。例如，假定一个销售事务表按照客户ID列（分布键）分布。如果查询中的谓词引用了一个单一的客户ID，该查询处理工作会被集中在一个Segment上。

复制表分布策略(DISTRIBUTED REPLICATED)应该在小表上使用。将大表数据复制到每个节点上无论在存储还是维护上都是有代价的。复制表最基本的用例是：

- 删除用户定义的函数可以对节点执行的操作的限制
- 频繁使用的表不需要广播到所有节点可以提高查询性能。

声明分布键

CREATE TABLE命令的可选子句DISTRIBUTED BY, DISTRIBUTED RANDOMLY和DISTRIBUTED REPLICATED决定了表的分布策略。默认的哈希分布策略使用PRIMARY KEY（如果有的话）或表的第一列

作为分布键。 几何信息列或用户自定义数据类型的列是不能作为Greenplum数据库分布列的。 如果找不到合适的哈希分布的列， Greenplum数据库就选择随机分布策略。

复制表没有分布列，因为每行都分布在Greenplum数据库所有节点上。

为了保证哈希分布数据的均匀分布，最好选一个唯一键作为分布列。如果找不到，则选择DISTRIBUTED RANDOMLY。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;
```

Important: 主键总是表的分布列。如果没有主键，但是有唯一索引存在，则选择它为分布键。

自定义分布键哈希函数

用于哈希分布策略的哈希函数由列的数据类型的哈希运算符类定义。由于默认的Greenplum数据库使用数据类型的默认哈希运算符类，因此用于哈希连接和哈希聚合的运算符类相同，适用于大多数用例。但是，您可以在DISTRIBUTED BY子句中声明非默认的哈希运算符类。

使用自定义哈希运算符类可以用于支持与默认相等运算符 (=) 不同的运算符上的共存连接。

自定义哈希操作符类用例

此示例为整数数据类型创建自定义哈希运算符类，该类用于提高查询性能。 运算符类比较整数的绝对值。

创建一个函数和一个等于运算符，如果两个整数的绝对值相等，则返回true。

```
CREATE FUNCTION abseq(int, int) RETURNS BOOL AS
$$
```

```

begin return abs($1) = abs($2); end;
$$ LANGUAGE plpgsql STRICT IMMUTABLE;

CREATE OPERATOR |=|
PROCEDURE = abseq,
LEFTARG = int,
RIGHTARG = int,
COMMUTATOR = |=|,
hashes, merges);

```

现在，创建一个使用运算符的哈希函数和运算符类。

```

CREATE FUNCTION abshashfunc(int) RETURNS int AS
$$
begin return hashint(abs($1)); end;
$$ LANGUAGE plpgsql STRICT IMMUTABLE;

CREATE OPERATOR CLASS abs_int_hash_ops FOR TYPE int4
USING hash AS
OPERATOR 1 |=|,
FUNCTION 1 abshashfunc(int);

```

并且，为它们创建小于和大于运算符和B树运算符类。我们的查询不需要它们，但是Greenplum数据库的Postgres查询优化器必须依赖它们做连接的co-location。

```

CREATE FUNCTION abslt(int, int) RETURNS BOOL AS
$$
begin return abs($1) < abs($2); end;
$$ LANGUAGE plpgsql STRICT IMMUTABLE;

CREATE OPERATOR |<| (
PROCEDURE = abslt,
LEFTARG = int,
RIGHTARG = int);

CREATE FUNCTION absgt(int, int) RETURNS BOOL AS
$$
begin return abs($1) > abs($2); end;
$$ LANGUAGE plpgsql STRICT IMMUTABLE;

CREATE OPERATOR |>| (
PROCEDURE = absgt,
LEFTARG = int,
RIGHTARG = int);

CREATE FUNCTION abscmp(int, int) RETURNS int AS
$$
begin return btint4cmp(abs($1),abs($2)); end;
$$ LANGUAGE plpgsql STRICT IMMUTABLE;

CREATE OPERATOR CLASS abs_int_btree_ops FOR TYPE int4
USING btree AS
OPERATOR 1 |<|,
OPERATOR 3 |=|;

```

```
OPERATOR 5 |>| ,  
FUNCTION 1 abscmp(int, int);
```

现在，您可以在表中使用自定义哈希运算符类。

```
CREATE TABLE atab (a int) DISTRIBUTED BY (a  
abs_int_hash_ops);  
CREATE TABLE btab (b int) DISTRIBUTED BY (b  
abs_int_hash_ops);  
  
INSERT INTO atab VALUES (-1), (0), (1);  
INSERT INTO btab VALUES (-1), (0), (1), (2);
```

执行使用自定义相等运算符`|=|`的连接的查询 可以利用co-location。
使用默认的整数opclass，此查询将需要Redistribute Motion节点，但使用自定义opclass，可以实现更高效的计划。

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

创建和管理数据库

创建和管理表空间

创建和管理SCHEMA

创建和管理表

选择表存储模型

对大型表分区

创建和使用序列

在Greenplum数据库中
使用索引

创建和管理视图

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

□ 性能管理

Greenplum数据库支持多种存储模型和一种混合存储模型。当用户创建一个表时，用户会选择如何存储它的数据。这个主题解释了表存储的选项以及如何为用户的负载选择最好的存储模型。

- [堆存储](#)
- [追加优化存储](#)
- [选择面向行或者面向列的存储](#)
- [使用压缩 \(只适用于追加优化表\)](#)
- [检查追加优化表的压缩和分布](#)
- [修改一个表](#)
- [删除一个表](#)

Note: 为了简化数据库表的创建，用户可以使用Greenplum数据库的服务器配置参数`gp_default_storage_options`为一些表存储选项指定默认值。

更多有关该参数的信息，请见*Greenplum*数据库参考指南中的“服务器配置参数”部分。

Parent topic: [定义数据库对象](#)

堆存储

默认情况下，Greenplum数据库使用和PostgreSQL相同的堆存储模型。堆表存储在OLTP类型负载下表现最好，这种环境中数据会在初始载入后被频繁地修改。UPDATE和DELETE操作要求存储行级版本信息来确保可靠的数据库事务处理。堆表最适合于较小的表，例如维度表，它们在初始载入数据后会经常被更新。

追加优化存储

追加优化表存储在数据仓库环境中的非规范表表现最好。非规范表通常是系统中最大的表。事实表通常成批地被载入并且被只读查询访问。将大型的事实表改为追加优化存储模型可以消除每行的更新可见性信息负担，这可以为每一行节约大概20字节。这可以得到一种更加简洁并且更容易优化的页面结构。追加优化表的存储模型是为批量数据装载优化的，因此不推荐单行的INSERT语句。

创建一个堆表

面向行的堆表是默认的存储类型。

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

使用CREATE TABLE命令的WITH子句可以声明表的存储选项。 默认是将表创建为面向行的堆存储表。 例如，要创建一个不压缩的追加优化表：

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

Note: 使用appendoptimized=value语法来指定追加优化类型的表存储。 appendoptimized是appendonly传统存储选项的精简别名。 Greenplum数据库在catalog里存储appendonly，并在列出追加优化表的存储选项时显示相同的内容。

在一个可重复读或可序列化事务中的追加优化表上不允许UPDATE和DELETE，它们将导致该事务中止。 追加优化表上不支持CLUSTER、DECLARE...FOR UPDATE和触发器。

选择面向行或者面向列的存储

Greenplum提供面向存储的模型选择：行，列或两者的组合。本主题提供了为表选择最佳存储的一般准则。 使用您自己的数据和查询工作负载评估性能。

- 面向行的存储：适用于具有许多迭代事务的OLTP类型的工作负载以及一次需要多列的单行，因此检索是高效的。
- 面向列的存储：适合于在少量列上计算数据聚集的数据仓库负载，或者是用于需要对单列定期更新但不修改其他列数据的情况。

对于大部分常用目的或者混合负载，面向行的存储提供了灵活性和性能的最佳组合。 不过，也有场景中面向列的存储模型提供了更高效的I/O和存储。 在为一个表决定存储方向模型时，请考虑下列需求：

- 表数据的更新。如果用户会频繁地装载和更新表数据，请选择一个面向行的堆表。 面向列的表存储只能用于追加优化表。

参考[堆存储](#)获得更多信息。

- 频繁的插入。如果频繁地向表中插入行，请考虑面向行的模型。列存表并未对写操作优化，因为一行的列值必须被写到磁盘上的不同位置。
- 查询中要求的列数。如果在查询的SELECT列表或者WHERE子句中常常要求所有或者大部分列，请考虑面向行的模型。面向列的表最适合的情况是，查询会聚集一个单一列中的很多值且WHERE或者HAVING谓词也在该聚集列上。例如：

```
SELECT SUM(salary)...
```

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

或者面向列的情况是WHERE谓词在一个单一列上并且返回相对较少的行。例如：

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

- 表中的列数。在同时要求很多列或者表的行尺寸相对较小时，面向行的存储会更有效。对于具有很多列的表且查询中访问这些列的一个小子集时，面向列的表能够提供更好的查询性能。
- 压缩。列数据具有相同的数据类型，因此在列存数据上支持存储尺寸优化，但在行存数据上则不支持。例如，很多压缩方案使用临近数据的相似性来进行压缩。不过，临近压缩做得越好，随机访问就会越困难，因为必须解压数据才能读取它们。

创建一个面向列的表

CREATE TABLE命令的WITH子句指定表的存储选项。默认是面向行的堆表。使用面向列的存储的表必须是追加优化表。例如，要创建一个列存表：

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

使用压缩（只适用于追加优化表）

对于追加优化表，在Greenplum数据库中有两种类型的库内压缩可用：

- 应用于整个表的表级压缩。
- 应用到一个指定列的列级压缩。用户可以为不同的列应用不同的列级压缩算法。

下面的表总结了可用的压缩算法。

Table 1. 用于追加优化表的压缩算法

表方向	可用的压缩类型	支持的算法
行	表	ZLIB, ZSTD和 QUICKLZ ¹
列	列和表	RLE_TYPE, ZLIB, ZSTD和 QUICKLZ ¹

Note: ¹QuickLZ压缩在Greenplum数据库的开源版本中不可用。

在为追加优化表选择一种压缩类型和级别时，要考虑这些因素：

- CPU使用。用户的Segment系统必须具有可用的CPU能力来压缩和解压数据。
- 压缩率/磁盘尺寸。最小化磁盘尺寸是一个因素，但也要考虑压缩和扫描数据所需的时间和CPU计算能力。要找到能高效压缩数据但不导致过长压缩时间或者过慢扫描率的最优设置。
- 压缩的速度。与zlib比较，QuickLZ在较低的压缩率下通常使用较少的CPU计算能力、能更快地压缩数据。zlib提供更高的压缩率，但是速度较慢。
例如，在压缩级别1上（compresslevel=1），QuickLZ和zlib具有差不多的压缩率，但是压缩速度不同。使用compresslevel=6的zlib能比QuickLZ更显著地提升压缩率，但是压缩速度更慢。ZStandard压缩可以提供良好的压缩比或速度，具体取决于压缩级别，或两者的良好折衷。
- 解压速度/扫描率。压缩的追加优化表的性能取决于硬件、查询调优设置和其他因素。请执行对比测试来判断在用户的环境中的真实性能。

Note: 不要在使用压缩的文件系统上创建压缩的追加优化表。如果用户的Segment数据目录所在的文件系统是一个压缩的文件系统，用户的追加优化表不能使用压缩。

压缩的追加优化表的性能取决于硬件、查询调优设置和其他因素。请执行对比测试来判断在用户的环境中的真实性能。

Note: Zstd压缩级别可以设置为1-19。QuickLZ的压缩级别只能被设置为级别1，没有其他的选项可用。zlib的压缩级别可以设置为1-9的

值。 RLE的压缩级别可以设置为1-4的值。

ENCODING子句指定个别列的压缩类型和级别。当ENCODING子句与WITH子句冲突时，ENCODING子句的优先级高于WITH子句。

创建一个压缩表

CREATE TABLE命令的WITH子句声明表的存储选项。使用压缩技术的表必须是追加优化表。例如，要创建一个使用zlib压缩且压缩级别为5的追加优化表：

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

检查追加优化表的压缩和分布

Greenplum提供了内建函数来检查一个追加优化表的压缩率和分布情况。这些函数要求对象ID或者表名作为参数。用户可以用schema名来限定表名。

Table 2. 用于压缩追加优化表元数据的函数

函数	返回类型	描述
get_ao_distribution(name) get_ao_distribution(oid)	(dbid, tuplecount)行 的集合	展示一个追加 优化表的行在 阵列中的分布 情况。返回一 个行集合，其 中每一个行包 括了一个节 点dbid以及存 储在其中的元 组数。
get_ao_compression_ratio(name) get_ao_compression_ratio(oid)	float8	为一个压缩的 追加优化表计 算压缩率。如 果该信息不可 用，这个函数 会返回-1。

压缩比作为公共比率返回。例如，返回值3.19或者3.19:1表示解压后的表比压缩表的尺寸的3倍略大。

表的分布被返回为一个行集合，它们反映了每个Segment上存储了多少元组。例如，在一个具有四个Primary Segment (*dbid*值从0-3) 的系统中，该函数返回与以下类似的四个行：

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

游程编码支持

Greenplum数据库对列级压缩支持游程编码 (RLE)。RLE数据压缩把重复的数据存储为一个单一的数据值和一个计数。例如，在有两个列date和description的表中，它包含200,000个含有值date1的项以及400,000个含有值date2的项，对这个date域的RLE压缩会类似于date1 200000 date2 400000。RLE对于没有大量重复数据集合的文件来说用处不大，因为它会大幅度增加这种文件的尺寸。

有四种级别的RLE压缩可用。这些级别逐步增加了压缩率，但是同时也会降低压缩速度。

Greenplum数据库4.2.1及其后的版本支持面向列的RLE压缩。如果要备份一个用了RLE压缩的表用来恢复到一个早期版本的Greenplum数据库中，应在开始备份操作前修改该表为不采用压缩或者采用早期版本中支持的压缩类型 (ZLIB或者QUICLZ)。

Greenplum数据库为BIGINT、INTEGER、DATE、TIME或者TIMESTAMP列中的数据的RLE压缩结合了增量压缩。该增量压缩算法基于连续列值之间的变化来进行压缩，它的设计目的是在以排序顺序装载数据时或者对已排序数据应用压缩时改进压缩性能。

增加列级压缩

用户可以为列存追加优化表的列增加下列存储指令：

- 压缩类型
- 压缩级别
- 列的块尺寸

使用CREATE TABLE、ALTER TABLE以及CREATE TYPE命令增加存储指令。

下面的表格详细介绍了存储指令的类型以及每一个指令可能的值。

Table 3. 列级压缩的存储指令

名称	定义	值	注释
compressstype	压缩的类型。	zstd: ZStandard算法 zlib: 缩小算法 quicklz: 快速算法 RLE_TYPE: 游程编码 none: 无压缩	值不区分大小写。
compresslevel	压缩级别。	zlib压缩: 1-9 zstd压缩: 1-19 QuickLZ压缩: 1 -	1是最快的方法但压缩率最低。1是默认值。 9是最慢的方法但压缩率最高。
			1是最快的方法但压缩率最低。1是默认值。 19是最慢的方法但压缩率最高。

	使用压缩	
RLE_TYPE压 缩: 1 – 4	1是最快的方法 但压缩率最低。	
1 - 只应用RLE 2 - 应用RLE然 后应用 zlib 压缩 级别1 3 - 应用RLE然 后应用 zlib 压缩 级别5 4 - 应用RLE然 后应用 zlib 压缩 级别9	4是最慢的方法 但压缩率最高。 1是默认值。	
blocksize	表中每 一块的 以字节 计的尺 寸	8192 – 2097152

下面是增加存储指令的格式。

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

其中单词ENCODING是必需的并且存储指令有三个部分：

- 指令的名称
- 一个等号
- 参数

多个存储指令用逗号分隔。如下面的CREATE TABLE子句所示，可以把一个存储指令应用到单一列或者把它作为所有列的默认指令。

一般用法：

```
column_name data_type ENCODING ( storage_directive [, ...] ) , ...
```

```
COLUMN column_name ENCODING ( storage_directive [, ...] ) , ...
```

```
DEFAULT COLUMN ENCODING ( storage_directive [, ...] )
```

示例：

```
C1 char ENCODING (compressstype=quicklz, blocksize=65536)
```

```
COLUMN C1 ENCODING (compressstype=zlib, compresslevel=6, blocksize=65536)
```

```
DEFAULT COLUMN ENCODING (compressstype=quicklz)
```

默认压缩值

如果没有定义压缩类型、压缩级别和块尺寸， 默认是无压缩并且块尺寸被设置为服务器配置参数block_size。

压缩设置的优先级

列压缩设置从表级继承到分区级， 再到子分区级。 最低级别上的设置优先。

- 表级定义的列压缩设置会覆盖该类型的任何压缩设置。
- 表级指定的列压缩设置会覆盖整个表的任何压缩设置。
- 为分区指定的列压缩设置会覆盖列级或表级别的任何压缩设置。
- 为子分区指定的列压缩设置会覆盖分区， 列或表级别的任何压缩设置。
- 当ENCODING子句与WITH子句冲突时， ENCODING子句的优先级高于WITH子句。

Note: 在一个含有存储指令或者列引用存储指令的表中不允许INHERITS子句。

使用LIKE子句创建的表忽略存储指令以及列引用存储指令。

列压缩设置的最佳位置

最佳做法是在数据所在的层次上设置列压缩设置。参考[例 5](#)，它展示了一个分区深度为2的表。`RLE_TYPE`压缩在子分区层次上被增加到一个列。

存储指令示例

下面的例子展示了在`CREATE TABLE`语句中存储指令的使用。

例 1

在这个例子中，列`c1`被使用`zstd`压缩并且使用系统中定义的块尺寸。列`c2`用`quicklz`压缩并且使用的块尺寸为`65536`。列`c3`没有被压缩并且使用系统中定义的块尺寸。

```
CREATE TABLE T1 (c1 int ENCODING (compressstype=zstd),
                  c2 char ENCODING (compressstype=quicklz,
blocksize=65536),
                  c3 char)      WITH (appendoptimized=true,
orientation=column);
```

例 2

在这个例子中，列`c1`使用`zlib`压缩并且使用系统中定义的块尺寸。列`c2`使用`quicklz`压缩，并且使用的块尺寸为`65536`。列`c3`使用`RLE_TYPE`压缩并且使用系统中定义的块尺寸。

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

例 3

在这个例子中，列`c1`使用`zlib`压缩并且使用系统中定义的块尺寸。列`c2`使用`quicklz`压缩，并且使用的块尺寸为`65536`。列`c3`使用`zlib`压缩并且使用系统中定义的块尺寸。注意，列`c3`在分区中使

用zlib (而不是RLE_TYPE) , 因为在分区子句中的列存储比表的列定义中的存储指令优先级高。

```
CREATE TABLE T3 (c1 int ENCODING (compressstype=zlib),
                  c2 char ENCODING (compressstype=quicklz,
blocksize=65536),
                  c3 text, COLUMN c3 ENCODING
(cpressstype=RLE_TYPE) )
      WITH (appendoptimized=true, orientation=column)
      PARTITION BY RANGE (c3) (START ('1900-01-01'::DATE)

END ('2100-12-31'::DATE),
COLUMN c3 ENCODING
(cpressstype=zlib));
```

例 4

在这个例子中, CREATE TABLE把zlib压缩类型存储指令分配给c1。列c2没有存储指令并且从DEFAULT COLUMN ENCODING子句继承了压缩类型 (quicklz) 和块尺寸 (65536) 。

列c3的ENCODING子句定义其压缩类型RLE_TYPE。为特定列定义的ENCODING子句会覆盖DEFAULT ENCODING子句, 因此列c3使用默认块大小32768。

列c4的压缩类型为none, 并使用默认块大小。

```
CREATE TABLE T4 (c1 int ENCODING (compressstype=zlib),
                  c2 char,
                  c3 text,
                  c4 smallint ENCODING (compressstype=none),
                  DEFAULT COLUMN ENCODING
(cpressstype=quicklz,
blocksize=65536),

COLUMN c3 ENCODING
(cpressstype=RLE_TYPE)
)
      WITH (appendoptimized=true, orientation=column);
```

例 5

这个例子创建一个追加优化的列存表T5。T5有两个分区p1和p2, 每一个都有子分区。每一个子分区都有ENCODING子句:

- 分区p1的子分区sp1的ENCODING子句定义了列i的压缩类型是zlib并且块尺寸是65536。
- 分区p2的子分区sp1的ENCODING子句定义了列i的压缩类型是rle_type并且块尺寸为默认值。列k使用默认压缩并且其块尺寸为8192。

```
CREATE TABLE T5(i int, j int, k int, l int)
  WITH (appendoptimized=true, orientation=column)
  PARTITION BY range(i) SUBPARTITION BY range(j)
  (
    partition p1 start(1) end(2)
    ( subpartition sp1 start(1) end(2)
      column i encoding(compressstype=zlib,
      blocksize=65536)
    ) ,
    partition p2 start(2) end(3)
    ( subpartition sp1 start(1) end(2)
      column i encoding(compressstype=rle_type)
      column k encoding(blocksize=8192)
    )
  );

```

用ALTER TABLE命令把一个压缩列增加到现有表的例子，请见[为表增加一个压缩列](#)。

在**TYPE**命令中增加压缩

创建新类型时，可以为该类型定义默认压缩属性。例如，以下CREATE TYPE命令定义了一个名为int33的类型，它指定了quicklz压缩：

```
CREATE TYPE int33 (
  internallength = 4,
  input = int33_in,
  output = int33_out,
  alignment = int4,
  default = 123,
  passedbyvalue,
  compressstype="quicklz",
  blocksize=65536,
  compresslevel=1
);
```

在CREATE TABLE命令中将int33指定为列类型时，将使用您为该类型指定的存储指令创建该列：

```
CREATE TABLE t2 (c1 int33)
```

```
WITH (appendoptimized=true, orientation=column);
```

您在表定义中指定的表级或列级存储属性会覆盖类型级存储属性。有关为类型创建和添加压缩属性的信息，请参阅[CREATE TYPE](#)。有关更改类型中的压缩规范的信息，请参阅[ALTER TYPE](#)。

选择块尺寸

`blocksize`是一个表中每一块的尺寸，以字节计。块尺寸必须介于8192字节和2097152字节之间，并且必须是8192的倍数。默认是32768。

指定大的块尺寸可能会消耗大量的内存。块大小决定了存储层中的缓存。在面向列的表中，Greenplum维护为每个分区每个列维护了一个缓存。具有许多分区或列的表占用大量内存。

修改一个表

`ALTER TABLE`命令改变一个表的定义。使用`ALTER TABLE`可以更改表的属性，例如列定义、分布策略、存储模型以及分区结构（另见[维护分区表](#)）。例如，为一个表列增加一个非空约束：

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

修改表的分布

`ALTER TABLE`提供了选项来改变一个表的分布策略。当表分布选项改变时，表数据会被在磁盘上重新分布，这可能会造成资源紧张。用户也可以使用现有的分布策略重新分布表数据。

更改分布策略

对于已分区的表，对于分布策略的更改会递归地应用到子分区上。这种操作会保留拥有关系和该表的所有其他属性。例如，下列命令使用`customer_id`列作为分布键在所有Segment之间重新分布表`sales`：

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

在用户改变一个表的哈希分布时，表数据会被自动重新分布。把分布策略改成随机分布不会导致数据被重新分布。例如，下面的ALTER TABLE命令不会立刻产生效果：

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

将表的分布策略更改为DISTRIBUTED REPLICATED或从DISTRIBUTED REPLICATED修改为其他分布，会自动重新分配表数据。

重新分布表的数据

要用一种随机分布策略（或者当哈希分布策略没有被更改时）对表重新分布数据，可使用REORGANIZE=TRUE。重新组织数据对于更正一个数据倾斜问题是必要的，当系统中增加了Segment资源后也需要重新组织数据。例如，下面的命令会使用当前的分布策略（包括随机分布）在所有Segment上重新分布数据。

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

将表的分布策略更改为DISTRIBUTED REPLICATED或从DISTRIBUTED REPLICATED改为其它总是重新分配表数据，即使REORGANIZE=False也是如此。

修改表的存储模型

表存储、压缩和行列类型只能在创建时声明。要改变存储模型，用户必须用正确的存储选项创建一个表，再把原始表的数据载入到新表中，接着删除原始表并且把新表重命名为原始表的名称。用户还必须重新授权原始表上有的权限。例如：

```
CREATE TABLE sales2 (LIKE sales)
WITH (appendoptimized=true, compressstype=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;
```

可参考[分裂分区](#)来学习如何改表一个已分区表的存储模型。

为表增加一个压缩列

使用ALTER TABLE命令为一个表增加一个压缩列。所有在[增加列级压缩](#)中描述的用于压缩列的选项和约束都适用于用ALTER TABLE命令增加的列。

下面的例子展示了如何向一个表中增加一个使用zlib压缩的列T1。

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

压缩设置的继承

如果一个表有子分区且子分区带有压缩设置，则向该表增加的一个分区会从子分区继承那些压缩设置。下面的例子展示了如何创建一个有子分区编码的表，然后修改它来增加一个分区。

```
CREATE TABLE ccddl (i int, j int, k int, l int)
WITH
    (appendoptimized = TRUE, orientation=COLUMN)
PARTITION BY range(j)
SUBPARTITION BY list (k)
SUBPARTITION template(
    SUBPARTITION sp1 values(1, 2, 3, 4, 5),
    COLUMN i ENCODING(compressstype=ZLIB),
    COLUMN j ENCODING(compressstype=QUICKLZ),
    COLUMN k ENCODING(compressstype=ZLIB),
    COLUMN l ENCODING(compressstype=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命令会创建表ccddl的名为ccddl_1_prt_p3和ccddl_1_prt_p3_2_prt_sp1的分区。分区ccddl_1_prt_p3继承了子分区sp1的不同的压缩编码。

删除一个表

DROP TABLE命令把表从数据库中移除。例如：

```
DROP TABLE mytable;
```

要清空一个表的行但不移除该表的定义，可使用DELETE或者TRUNCATE。例如：

```
DELETE FROM mytable;
```

```
TRUNCATE mytable;
```

DROP TABLE总是会移除目标表上存在的任何索引、规则、触发器和约束。删除一个被视图引用的表应指定CASCADE。CASCADE会移除依赖表的视图。

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

创建和管理数据库

创建和管理表空间

创建和管理SCHEMA

创建和管理表

选择表存储模型

对大型表分区

创建和使用序列

在Greenplum数据库中使用索引

创建和管理视图

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

□ 性能管理

表分区让我们能通过把表划分成较小的、更容易管理的小块来支持非常大的表，例如事实表。通过让Greenplum数据库查询优化器只扫描满足给定查询所需的数据而避免扫描大表的全部内容，分区表能够提升查询性能。

- [关于表分区](#)
- [决定表的分区策略](#)
- [创建分区表](#)
- [加载分区表](#)
- [验证分区策略](#)
- [查看用户的分区设计](#)
- [维护分区表](#)

Parent topic: [定义数据库对象](#)

关于表分区

分区并不会改变表数据在Segment之间的物理分布。表分布是物理的：Greenplum数据库会在物理上把分区表和未分区表划分到多个Segment上来启用并行查询处理。表分区是逻辑的：Greenplum数据库在逻辑上划分大表来提升查询性能并且有利于数据仓库维护任务，例如把旧数据滚出数据仓库。

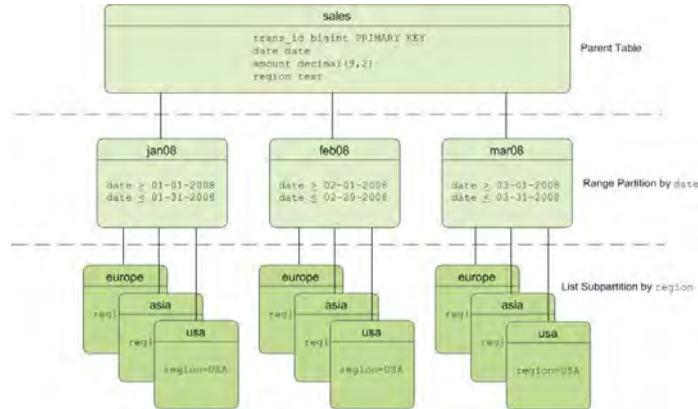
Greenplum数据库支持：

- 范围分区：基于一个数字型范围划分数据，例如按照日期或价格划分。
- 列表分区：基于一个值列表划分数据，例如按照销售范围或产品线划分。
- 两种类型的组合。

Figure 1. 多层分区设计的例子



Greenplum database 5管理员



Greenplum数据库中的表分区

Greenplum数据库把表划分成部分（也称为分区）来启用大规模并行处理。表分区在使用PARTITION BY（以及可选的SUBPARTITION BY）子句的CREATE TABLE执行期间进行。分区操作会创建一个顶层（父）表以及一层或者多层次子表。在内部，Greenplum数据库会在顶层表和它的底层分区之间创建继承关系，类似于PostgreSQL的INHERITS子句的功能。

Greenplum使用表创建时定义的分区标准来创建每一个分区及其上一个可区分的CHECK约束，这个约束限制了该表能含有的数据。查询优化器使用CHECK约束来决定要扫描哪些表分区来满足一个给定的查询谓词。

Greenplum系统目录存储了分区层次信息，这样插入到顶层父表的行会被正确地传播到子表分区。要更改分区设计或者表结构，可使用带有PARTITION子句的ALTER TABLE修改父表。

要把数据插入到一个分过区的表中，用户需要指定根分区表，也就是用CREATE TABLE命令创建的那个表。用户也可以在INSERT命令中指定分区表的一个叶子子表。如果该数据对于指定的叶子子表不合法，则会返回一个错误。不支持在DML命令中指定一个非叶子或者非根分区表。

决定表的分区策略

Greenplum数据库不支持对复制表进行分区(DISTRIBUTED REPLICATED)。不是所有的哈希分布表或随机分布表都适合于分区。如果下列问题的答案全部或者大部分都是yes，表分区就是一种可行的改进查询性能的数据库设计策略。如果下列问题的答案大部分都是no，表分区对于该表就不是正确的方案。请测试用户的设计策略

来确保查询性能能得到预期的改进。

- 表是否足够大？大型的事实表是进行表划分很好的候选。如果在一个表中有几百万或者几十亿个记录，从逻辑上将数据分成较小的块会让用户在性能方面受益。对于只有几千行或者更少数据的小表来说，维护分区的管理开销将会超过用户可能得到的性能收益。
- 用户是否体验到不满意的性能？正如任何性能调节的动机一样，只有针对一个表的查询产生比预期还要慢的响应时间时才应该对该表分区。
- 用户的查询谓词有没有可识别的访问模式？检查用户的查询负载的WHERE子句并且查找一直被用来访问数据的表列。例如，如果大部分查询都倾向于用日期查找记录，那么按月或者按周的日期分区设计可能会对用户有益。或者如果用户倾向于根据地区访问记录，可考虑一种列表分区设计来根据地区划分表。
- 用户的数据仓库是否维护了一个历史数据的窗口？另一个分区设计的考虑是用户的组织对维护历史数据的业务需求。例如，用户的数据仓库可能要求用户保留过去十二个月的数据。如果数据按月分区，用户可以轻易地从仓库中删除最旧的月份分区并且把当前数据载入到最近的月份分区中。
- 数据能否基于某种定义的原则被划分成差不多相等的部分？尽可能选择将把用户的数据均匀划分的分区原则。如果分区包含基本同等数量的记录，查询性能会基于创建的分区数量而提升。例如，通过将一个大型表划分成10个分区，一个查询的执行速度将比在未分区表上快10倍，前提是这些分区就是为支持该查询的条件而设计。

不要创建超过所需数量的分区。创建过多的分区可能会拖慢管理和维护工作，例如清理、恢复Segment、扩展集群、检查磁盘用量等等。

除非查询优化器能基于查询谓词排除一些分区，否则分区技术不能改进查询性能。每个分区都扫描的查询运行起来会比表没有分区时还慢，因此如果用户的查询中很少能排除分区，请避免进行分区。请检查查询的解释计划来确认分区被排除。参考[查询分析](#)获取更多关于分区的信息。

Warning: 请对多级分区格外谨慎，因为分区文件的数量可能会增长得非常快。例如，如果一个表被按照日和城市划分并且有1,000个日以及1,000个城市，那么分区的总数就是一百万。列存表会把每一列存在一个物理表中，因此如果这个表有100个列，系统就需要为该表管理一亿个文件。

在选定一种多级分区策略之前，可以考虑一种带有位图索引的单级分区。索引会降低数据装载的速度，因此推荐用用户的数据和模式进行性能测试以决定最佳的策略。

创建分区表

在使用CREATE TABLE创建表时就可以对它们分区。这个主题提供了用于创建带有数个分区的表的SQL语法的例子。

对一个表分区：

1. 决定分区设计：日期范围、数字范围或者值的列表。
2. 选择要按哪个（哪些）列对表分区。
3. 决定用户需要多少个分区级别。例如，用户可以按月创建一个日期范围分区表，然后对每个月的分区按照销售地区划分子分区。
 - [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](#)
 - [对已有的表进行分区](#)

定义日期范围分区表

一个按日期范围分区的表使用单个date或者timestamp列作为分区键列。如果需要，用户可以使用同一个分区键列来创建子分区，例如按月分区然后按日建子分区。请考虑使用最细的粒度分区。例如，对于一个用日期分区的表，用户可以按日分区并且得到365个每日的分区，而不是先按年分区然后按月建子分区再然后按日建子分区。一种多级设计可能会减少查询规划时间，但是一种平面的分区设计运行得更快。

用户可以通过给出一个START值、一个END值以及一个定义分区增量值的EVERY子句让Greenplum数据库自动产生分区。默认情况下，START值总是被包括在内而END值总是被排除在外。例如：

```
CREATE TABLE sales (id int, date date, amt decimal(10,2))
DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
( START (date '2016-01-01') INCLUSIVE
  END (date '2017-01-01') EXCLUSIVE
  EVERY (INTERVAL '1 day') );
```

用户也可以逐个声明并且命名每一个分区。例如：

```

CREATE TABLE sales (id int, date date, amt decimal(10,2))
DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
( PARTITION Jan16 START (date '2016-01-01') INCLUSIVE ,
  PARTITION Feb16 START (date '2016-02-01') INCLUSIVE ,
  PARTITION Mar16 START (date '2016-03-01') INCLUSIVE ,
  PARTITION Apr16 START (date '2016-04-01') INCLUSIVE ,
  PARTITION May16 START (date '2016-05-01') INCLUSIVE ,
  PARTITION Jun16 START (date '2016-06-01') INCLUSIVE ,
  PARTITION Jul16 START (date '2016-07-01') INCLUSIVE ,
  PARTITION Aug16 START (date '2016-08-01') INCLUSIVE ,
  PARTITION Sep16 START (date '2016-09-01') INCLUSIVE ,
  PARTITION Oct16 START (date '2016-10-01') INCLUSIVE ,
  PARTITION Nov16 START (date '2016-11-01') INCLUSIVE ,
  PARTITION Dec16 START (date '2016-12-01') INCLUSIVE
                END (date '2017-01-01') EXCLUSIVE );

```

用户不需要为每一个分区声明一个END子句，只需要为最后一个分区写上就好。在这个例子中，Jan16会在Feb16开始处结束。

定义数字范围分区表

一个按数字范围分区的表使用单个数字数据类型列作为分区键列。例如：

```

CREATE TABLE rank (id int, rank int, year int, gender
char(1), count int)
DISTRIBUTED BY (id)
PARTITION BY RANGE (year)
( START (2006) END (2016) EVERY (1),
  DEFAULT PARTITION extra );

```

更多有关默认分区的信息，请见[增加默认分区](#)。

定义列表分区表

一个按列表分区的表可以使用任意允许等值比较的数据类型列作为它的分区键列。一个列表分区也可以用一个多列（组合）分区键，反之一个范围分区只允许单一列作为分区键。对于列表分区，用户必须为每一个用户想要创建的分区（列表值）声明一个分区说明。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 );
```

Note: 当前的Greenplum数据库传统优化器允许列表分区带有多列（组合）分区键。一个范围分区只允许单一列作为分区键。Greenplum查询优化器不支持组合键，因此用户不能使用组合分区键。

更多有关默认分区的信息，请见[增加默认分区](#)。

定义多级分区表

用户可以用分区的子分区创建一种多级分区设计。使用一个子分区模板可以确保每一个分区都有相同的子分区设计，包括用户后来增加的分区。例如，下面的SQL创建所示的两级分区设计[Figure 1](#)：

```
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 '2011-01-01') INCLUSIVE
END (date '2012-01-01') EXCLUSIVE
EVERY (INTERVAL '1 month'),
DEFAULT PARTITION outlying_dates );
```

下面的例子展示了一个三级分区设计，其中sales表被按照year分区，然后按照month分区，再然后按照region分区。

SUBPARTITION TEMPLATE子句保证每一个年度的分区都有相同的子分区结构。这个例子在该层次的每一个级别上都声明了一个DEFAULT分区。

```
CREATE TABLE p3_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 (2012) EVERY (1),
  DEFAULT PARTITION outlying_years );

```

CAUTION:

当用户创建基于范围的多级分区时，很容易会创建大量的子分区，有一些包含很少的甚至不包含数据。这可能会在系统表中增加很多项，这些项增加了优化和执行查询所需的时间和内存。增加范围区间或者选择一种不同的分区策略可减少创建的子分区数量。

对已有的表进行分区

表只能在创建时被分区。如果用户有一个表想要分区，用户必须创建一个分过区的表，把原始表的数据载入到新表，再删除原始表并且把分过区的表重命名为原始表的名称。用户还必须重新授权表上的权限。例如：

```

CREATE TABLE sales2 (LIKE sales)
PARTITION BY RANGE (date)
( START (date '2016-01-01') INCLUSIVE
  END (date '2017-01-01') 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

对于每个分区级别，一个已分区的表最多能有32,767个分区。

一个已分区表上的主键或者唯一约束必须包含所有的分区列。一个唯一索引可以忽略分区列，但是它只能在已分区表的每个部分而不是整个已分区的表上被强制。

用DISTRIBUTED REPLICATED分布策略创建的表不能被分区。

Greenplum的下一代查询优化器GPORCA支持统一的多级分区表。如果启用了GPORCA（默认情况）并且多级分区表不统一，Greenplum数据库会用传统查询优化器对该表执行查询。有关统一多级分区表的信息，请见[关于统一多级分区表](#)。

有关交换叶节点和外部表的信息，请参考[用外部表交换叶子子分区](#)。

当一个叶子子分区是外部表时，对分区表有一些限制：

- 针对包含外部表分区的分区表运行的查询将用传统查询优化器执行。
- 外部表分区是一个只读外部表。尝试在该外部表分区中访问或者修改数据的命令会返回一个错误。例如：
 - 尝试在外部表分区中改变数据的INSERT、DELETE以及UPDATE命令会返回一个错误。
 - TRUNCATE命令返回一个错误。
 - COPY命令无法复制数据到一个会更新外部表分区的分区表中。
 - 尝试从一个外部表分区中复制出数据的COPY命令会返回一个错误，除非用户为COPY命令指定IGNORE EXTERNAL PARTITIONS子句。如果用户指定该子句，数据不会被从外部表分区复制出来。

要对一个有外部表作为叶子子表的分区表使用COPY命令，可以使用一个SQL查询来拷贝数据。例如，如果表my_sales包含一个外部表作为叶子子表，这个命令可以把其数据发送到stdout：

```
COPY (SELECT * from my_sales ) TO stdout
```

- VACUUM命令会跳过外部表分区。
- 如果在外部表分区上没有数据改变，则支持下列操作。否则，返回一个错误。
 - 增加或者删除一列。
 - 更改列的数据类型。
- 如果分区表包含一个外部表分区，则不支持这些ALTER PARTITION操作：
 - 设置一个子分区模板。
 - 更改分区属性。
 - 创建一个默认分区。
 - 设置一种分布策略。
 - 设置或者删除列的一个NOT NULL约束。
 - 增加或者删除约束。
 - 分裂一个外部分区。
- 如果分区表的一个叶子子分区是一个可读的外部表，Greenplum数据库工具gpbackup不会从该叶子子分区中备份数据。

加载分区表

在用户创建了分区表结构之后，顶层父表为空。 数据会被路由到底层的子表分区中。 在一个两级分区设计中，只有层次底部的子分区能够包含数据。

不能被映射到一个子表分区的行会被拒绝并且载入会失败。 为了避免无法映射的行在载入时被拒绝，可以为用户的分区层次定义一个DEFAULT分区。 任何不匹配一个分区的CHECK约束的行会被载入到DEFAULT分区。 参见[增加默认分区](#)。

在运行时，查询优化器扫描整个表继承层次并使用CHECK表约束来决定要扫描哪个子表分区来满足查询的条件。 DEFAULT分区（如果用户的层次中有一个）总是会被扫描。 包含数据的DEFAULT分区会拖慢总体扫描时间。

当用户使用COPY或者INSERT来载入数据到父表时，数据会被自动路由到正确的分区，这就像是向一个常规表中载入数据一样。

向分区表中载入数据的最佳方法是创建一个中间状态表，把数据载入其中，然后把它交换到用户的分区设计中。 参见[交换分区](#)。

验证分区策略

当一个表基于查询谓词被分区时，用户可以使用EXPLAIN来验证查询优化器只扫描相关的数据来检查查询计划。

例如，假设一个*sales* 表被按日期范围分区，先用月份分区然后用地区建立子分区，如图[Figure 1](#) 所示。 对于下列查询：

```
EXPLAIN SELECT * FROM sales WHERE date='01-07-12' AND
region='usa';
```

这个查询的查询计划应该展示只涉及到下列表的表扫描：

- 返回0-1行的默认分区（如果用户的分区设计有一个默认分区）
- 返回0-1行的January 2012分区（*sales_1_prt_1*）
- 返回若干行的USA地区子分区（*sales_1_2_prt_usa*）。

下面的例子展示了相关的查询计划片段。

```

-> Seq Scan on sales_1_prt_1 sales (cost=0.00..0.00 rows=0
    width=0)
Filter: "date"=01-07-12::date AND region='USA'::text
-> Seq Scan on sales_1_2_prt_usa sales (cost=0.00..9.87
rows=20
    width=40)

```

确保查询优化器不会扫描不必要的分区或者子分区（例如，扫描没有在查询谓词中指定的月份或者地区），以及顶层表的扫描返回0-1行。

选择性分区扫描疑难解答

下列限制可能导致一个对用户的分区层次进行非选择性扫描的查询计划。

- 只有当查询包含表的使用不可变操作符，例如 $=, <, <=, >, >=,$ 和 $<>$ 的直接或者简单限制时，查询优化器才能有选择地扫描分区表。
- 选择性扫描识别查询中的STABLE以及IMMUTABLE函数，但是不识别VOLATILE函数。例如，`date > CURRENT_DATE`这样的WHERE子句导致查询优化器选择性扫描分区表，但是`time > TIMEofday`不会。

查看用户的分区设计

用户可以使用`pg_partitions`视图查看有关分区设计的信息。例如，要查看`sales`表的分区设计：

```

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

```

下列表和视图展示了关于分区表的信息。

- `pg_partition` - 跟踪分区表以及它们的继承层次关系。
- `pg_partition_templates` - 展示使用一个子分区模板创建的子分区。
- `pg_partition_columns` - 显示在一个分区设计中用到的分区键列。

有关Greenplum数据库系统目录表和视图的信息，请见Greenplum数据库参考指南。

维护分区表

要维护一个分区表，对顶层父表使用ALTER TABLE命令。最常用的情景是删除旧的分区以及增加新的分区，以此在一种范围分区设计中维护数据的一个滚动窗口。用户可以把旧的分区转换（交换）成追加优化的压缩存储格式来节省空间。如果在用户的分区设计中有一个默认分区，用户可以通过分裂默认分区来增加一个分区。

- [增加分区](#)
- [重命名分区](#)
- [增加默认分区](#)
- [删除分区](#)
- [截断分区](#)
- [交换分区](#)
- [分裂分区](#)
- [修改子分区模板](#)
- [用外部表交换叶子子分区](#)

Important: 重要：在定义和改变分区设计时，要使用给定的分区名而不是表对象名。尽管用户可以直接使用SQL命令来查询和装载任何表（包括分区表），用户只能使用ALTER TABLE...PARTITION子句修改一个分区表的结构。

分区并不要求有名称。如果一个分区没有名称，可使用下列表达式之一来指定它：PARTITION FOR (value)或者PARTITION FOR(RANK(number))。

增加分区

用户可以用ALTER TABLE命令为一个分区设计增加一个分区。如果原始分区设计包括由一个子分区模板定义的子分区，新增加的分区也会根据该模板划分子分区。例如：

```
ALTER TABLE sales ADD PARTITION
    START (date '2017-02-01') INCLUSIVE
    END (date '2017-03-01') EXCLUSIVE;
```

如果在创建表时没有使用一个子分区模板，用户可以在增加分区时定

义子分区：

```
ALTER TABLE sales ADD PARTITION
    START (date '2017-02-01') INCLUSIVE
    END (date '2017-03-01') EXCLUSIVE
( SUBPARTITION usa VALUES ('usa'),
  SUBPARTITION asia VALUES ('asia'),
  SUBPARTITION europe VALUES ('europe') );
```

当用户为一个现有分区增加一个子分区时，用户可以指定要更改的分区。例如：

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

Note: 用户不能向一个具有默认分区的分区设计中增加分区。用户必须分裂默认分区来增加分区。参见[分裂分区](#)。

重命名分区

分区表使用下列命名习惯。分区子表的名称服从唯一性要求和长度限制。

```
<parentname>_<level>_prt_<partition_name>
```

例如：

```
sales_1_prt_jan16
```

对于自动生成的范围分区，在没有给出名称时会分配一个数字：

```
sales_1_prt_1
```

要重命名一个已分区的子表，应重命名顶层父表。在所有相关的子表分区的表名中，**<parentname>**都会改变。例如下面的命令：

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

会修改相关的表名：

```
globalsales_1_prt_1
```

用户可以更改一个分区的名称让它更容易标识。例如：

```
ALTER TABLE sales RENAME PARTITION FOR ('2016-01-01') TO jan16;
```

会把相关的表名改为如下：

```
sales_1_prt_jan16
```

在使用ALTER TABLE命令修改分区表时，总是用它们的分区名(*jan16*)而不是它们的完整表名(*sales_1_prt_jan16*)引用表。

Note: 表名不能是一个ALTER TABLE语句中的分区名。例如，ALTER TABLE sales...是正确的。ALTER TABLE sales_1_part_jan16...则不被允许。

增加默认分区

用户可以用ALTER TABLE命令为一个分区设计增加一个默认分区。

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

如果用户的分区设计是多级的，该层次中每一级都必须有一个默认分区。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;
```

如果到来的数据不匹配一个分区的CHECK约束并且没有默认分区，该数据就会被拒绝。默认分区确保到来的不匹配一个分区的数据能被插入到默认分区中。

删除分区

用户可以使用ALTER TABLE命令从用户的分区设计中删除一个分

区。当用户删除一个具有子分区的分区时，子分区（以及其中的所有数据）也会被自动删除。对于范围分区，从范围内删除较老的分区很常见，因为旧的数据会被滚出数据仓库。例如：

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

截断分区

用户可以使用ALTER TABLE命令截断一个分区。当用户截断一个具有子分区的分区时，子分区也会被自动截断。

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

交换分区

用户可以使用ALTER TABLE命令交换一个分区。交换一个分区用一个表换掉一个现有的分区。用户只能在分区层次的最底层交换分区（只有包含数据的分区才可以被交换）。

不可以交换包含复制表的分区。不支持使用分区表或分区表的子分区交换分区。

分区交换对数据装载有用。例如，装载一个分段表并且把装载好的表换入到用户的分区设计中去。用户可以使用分区交换来把较老分区的存储类型改为追加优化表。例如：

```
CREATE TABLE jan12 (LIKE sales) WITH
(appendoptimized=true);
INSERT INTO jan12 SELECT * FROM sales_1_prt_1 ;
ALTER TABLE sales EXCHANGE PARTITION FOR (DATE '2012-01-
01')
WITH TABLE jan12;
```

Note: 这个例子指的是表sales的单级定义，就是在前面那些例子对它增加或者修改分区之前。

Warning: 如果用户指定WITHOUT VALIDATION子句，用户必须确保用户用于交换现有分区的表中的数据对于该分区上的约束是合法的。否则，针对分区表的查询可能会返回不正确的结果。

Greenplum数据库服务器配置参

数gp_enable_exchange_default_partition控制EXCHANGE

DEFAULT PARTITION子句的可用性。该参数的默认值是off，表示该子句不可用，如果在ALTER TABLE命令中指定了该子句，Greenplum数据库会返回一个错误。

关于该参数的信息，请见*Greenplum*数据库参考指南中的“服务器配置参数”。

Warning: 在用户交换默认分区前，用户必须确保要被交换的表中的数据（即新的默认分区）对于默认分区是合法的。例如，新默认分区中的数据不能含有对分区表其他叶子子分区有效的数据。否则，交换过默认分区的分区表上由GPORCA执行的查询可能会返回不正确的结果。

分裂分区

分裂一个分区会把一个分区划分成两个分区。用户可以使用ALTER TABLE命令分裂分区。只能在最低层级的分区做分裂（包含数据的分区）。对于对级别分区，只是范围分区可以分裂，列表分区不行。用户指定的分裂值会分在后一个分区中。

例如，把一个月度分区分裂成两个，第一个分区包含日期January 1-15而第二个分区包含日期January 16-31：

```
ALTER TABLE sales SPLIT PARTITION FOR ('2017-01-01')
AT ('2017-01-16')
INTO (PARTITION jan171to15, PARTITION jan1716to31);
```

如果用户的分区设计有一个默认分区，用户必须分裂该默认分区来增加分区。

在使用INTO子句时，指定当前的默认分区为第二个分区名。例如，要分裂一个默认的范围分区来为January 2017增加一个新的月度分区：

```
ALTER TABLE sales SPLIT DEFAULT PARTITION
START ('2017-01-01') INCLUSIVE
END ('2017-02-01') EXCLUSIVE
INTO (PARTITION jan17, default partition);
```

修改子分区模板

使用ALTER TABLE SET SUBPARTITION TEMPLATE来修改一个分区表的子分区模板。在用户设置了新子分区模板之后增加的分区会具

有新的分区设计。现有的分区不会被改变。

下面的例子修改这个分区表的子分区模板：

```
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 '2014-01-01') INCLUSIVE
      END (date '2014-04-01') EXCLUSIVE
      EVERY (INTERVAL '1 month') );
```

这个ALTER TABLE命令修改子分区模板。

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

当用户为表sales增加一个日期范围分区时，它包括非洲的新地区列表子分区。例如，下面的命令创建子分区usa、asia、europe、africa以及一个名为other的默认分区：

```
ALTER TABLE sales ADD PARTITION "4"
  START ('2014-04-01') INCLUSIVE
  END ('2014-05-01') EXCLUSIVE ;
```

要查看为分区表sales创建的表，用户可以从psql命令行使用\dt sales*。

要移除一个子分区模板，使用带有空圆括号的SET SUBPARTITION TEMPLATE。例如，要清除sales表的子分区模板：

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

用外部表交换叶子子分区

用户可以用一个可读的外部表交换一个分区表中的一个叶子子分区。外部表数据可以位于一个主机文件系统、一个NFS挂载或者一个Hadoop文件系统 (HDFS)。

例如，如果用户有一个分区表，它按月被分成月度分全局并且对该表的大部分查询值访问较新的数据，用户可以把较旧的、较少访问的数据拷贝到外部表并且把较旧的分区与这些外部表交换。对于之访问较新数据的查询，用户可以创建使用分区排除的查询来防止扫描较旧的、不需要的分区。

如果分区表含有一个带检查约束或者NOT NULL约束的列，用一个外部表交换一个叶子子分区是不可以的。

关于交换和修改一个叶子子分区的信息，请见*Greenplum数据库命令参考*中的ALTER TABLE命令。

关于含有外部表分区的分区表的限制，请见[分区表的限制](#)。

用外部表交换分区的例子

这是一个简单的例子，它把这个分区表的一个叶子子分区交换为一个外部表。分区表包含2010至2013年份的数据。

```
CREATE TABLE sales (id int, year int, qtr int, day int,
region text)
DISTRIBUTED BY (id)
PARTITION BY RANGE (year)
( PARTITION yr START (2010) END (2014) EVERY (1) ) ;
```

该分区表有四个叶子子分区。每一个叶子子分区含有单一年份的数据。叶子子分区表sales_1_prt_yr_1包含2010年的数据。这些步骤把表sales_1_prt_yr_1交换为一个使用gpfdist协议的外部表：

1. 确保Greenplum数据库系统启用了该外部表协议。

这个例子使用了gpfdist协议。这个命令开始gpfdist协议。

```
$ gpfdist
```

2. 创建一个可写的外部表。

这个CREATE WRITABLE EXTERNAL TABLE命令用和分区表相同的列创建一个可写的外部表。

```
CREATE WRITABLE EXTERNAL TABLE my_sales_ext ( LIKE
```

```

sales_1_prt_yr_1 )
LOCATION ( 'gpfdist://gpdb_test/sales_2010' )
FORMAT 'csv'
DISTRIBUTED BY (id) ;

```

3. 创建一个可读的外部表，它从前一步创建的可写外部表的目的地读取数据。

这个CREATE EXTERNAL TABLE创建一个可读外部表，它使用和可写外部数据相同的外部数据。

```

CREATE EXTERNAL TABLE sales_2010_ext ( LIKE
sales_1_prt_yr_1)
LOCATION ( 'gpfdist://gpdb_test/sales_2010' )
FORMAT 'csv' ;

```

4. 从叶子子分区中拷贝数据到该可写外部表。

这个INSERT

```

INSERT INTO my_sales_ext SELECT * FROM sales_1_prt_yr_1
;

```

5. 用该外部表交换现有的叶子子分区。

这个ALTER TABLE命令指定了EXCHANGE PARTITION子句来切换可读外部表和叶子子分区。

```

ALTER TABLE sales ALTER PARTITION yr_1
EXCHANGE PARTITION yr_1
WITH TABLE sales_2010_ext WITHOUT VALIDATION;

```

以表名sales_1_prt_yr_1成为叶子子分区的外部表，并且旧的叶子子分区变成表sales_2010_ext。

Warning: 为了确保针对分区表的查询返回正确的结果，外部表数据必须针对叶子子分区上的CHECK约束有效。在这种情况下，数据会从其上定义有CHECK约束的叶子子分区表中取出。

6. 删除滚出分区表的表。

```

DROP TABLE sales_2010_ext ;

```

用户可以重命名该叶子子分区的名称来表明sales_1_prt_yr_1是一个外部表。

这个示例命令把分区名改为yr_1_ext把叶子子分区表的名称改为sales_1_prt_yr_1_ext。

```
ALTER TABLE sales RENAME PARTITION yr_1 TO yr_1_ext ;
```

Greenplum数据库® 6.0文档

- 管理员指南

- Greenplum数据库概念

- 管理一个Greenplum系统

- 管理Greenplum数据库访问

- 定义数据库对象

- 创建和管理数据库

- 创建和管理表空间

- 创建和管理SCHEMA

- 创建和管理表

- 选择表存储模型

- 对大型表分区

- 创建和使用序列**

- 在Greenplum数据库中使用索引

- 创建和管理视图

- 分布与倾斜

- 插入, 更新, 和删除数据

- 查询数据

- 使用外部数据

- 装载和卸载数据

- 性能管理

Greenplum数据库序列对象是一个特殊的单行表，用作数字生成器。您可以使用序列为添加到表中的行生成唯一的整数标识符。声明SERIAL类型的列会隐式创建一个序列计数器，以便在该表列中使用。

Greenplum数据库提供了创建, 更改和删除序列的命令。Greenplum数据库还提供内置函数来返回序列中的下一个值 (`nextval()`) 或将序列设置为特定的起始值 (`setval()`)。

Note: Greenplum数据库不支持PostgreSQL`currval()`和`lastval()`序列函数。

序列对象的属性包括序列的名称, 其增量值以及序列计数器的最后, 最小和最大值。序列还有一个名为`is_called`的特殊布尔属性, 用于控制序列计数器上`nextval()`操作的自动递增行为。当序列的`is_called`属性为`true`时, `nextval()`会在返回值之前递增序列计数器。当序列的`is_called`属性值为`false`时, `nextval()`在返回值之前不会递增计数器。

Parent topic: [定义数据库对象](#)

创建一个序列

`CREATE SEQUENCE`命令使用给定的序列名称和可选的起始值创建和初始化序列。序列名称必须与同一Schema中任何其他序列, 表, 索引或视图的名称不同。例如:

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

创建新序列时, Greenplum数据库将序列`is_called`属性设置为`false`。在新创建的序列上调用`nextval()`不会递增序列计数器, 但会返回序列起始值并将`is_called`设置为`true`。

使用一个序列

使用`CREATE SEQUENCE`命令创建序列后, 可以检查序列并使用序列内置函数。

检查序列属性

要检查序列的当前属性，请直接查询序列。例如，要检查名为myserial的序列：

```
SELECT * FROM myserial;
```

返回下一个序列计数器值

您可以调用nextval()内置函数来返回并使用序列中的下一个值。以下命令将名为myserial的序列的下一个值插入名为vendors的表的第一列：

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

nextval()使用序列的is_called属性值来确定在返回值之前是否递增序列计数器。当is_called为true时，nextval()使计数器前进。nextval()在返回之前将序列is_called属性设置为true。

nextval()操作永远不会回滚。即使执行nextval()的事务失败，获取的值也会被使用。这意味着失败的事务可能会在指定值的序列中留下未使用的漏洞。

Note: 如果在Greenplum数据库中启用了镜像，则不能在UPDATE或DELETE语句中使用nextval()函数。

设置序列计数器值

您可以使用Greenplum数据库setval()内置函数来设置序列的计数器值。例如，以下命令将名为myserial的序列的计数器值设置为201：

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

setval()有两个函数签名：setval(sequence, start_val)和setval(sequence, start_val, is_called)。setval(sequence, start_val)的默认行为设置序列is_called属性值为true。

如果您不希望序列计数器在下一个nextval()调用时递增，请使用setval(sequence, start_val, is_called)函数签名，传递false参数：

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

setval()操作永远不会回滚。

修改一个序列

[ALTER SEQUENCE](#)命令更改现有序列的属性。您可以更改序列的开始，最小，最大和增量值。您也可以在起始值或指定值处重新启动序列。

未在ALTER SEQUENCE命令中设置的任何参数都保留其先前的设置。

ALTER SEQUENCE *sequence* START WITH *start_value*将序列的start_value属性设置为新的起始值。它对last_value属性或nextval(*sequence*)函数返回的值没有影响。

ALTER SEQUENCE *sequence* RESTART 将序列的last_value属性重置为start_value属性的当前值，并将is_called属性重置为false。对nextval(*sequence*)函数的下一次调用将返回*start_value*。

ALTER SEQUENCE *sequence* RESTART WITH *restart_value*将序列的last_value属性设置为新值，将is_called属性设置为false。下一次调用nextval(*sequence*)会返回*restart_value*。这等效于调用setval(*sequence*, *restart_value*, false)。

以下命令在值105处重新启动名为myserial的序列：

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

删除一个序列

[DROP SEQUENCE](#)命令可以删除序列。例如，以下命令将删除名为myserial的序列：

```
DROP SEQUENCE myserial;
```

将序列指定为列的默认值

除了使用SERIAL或BIGSERIAL类型之外，您还可以直接在[CREATE TABLE](#)命令中引用序列。例如：

```
CREATE TABLE tablename ( id INT4 DEFAULT
nextval('myserial'), name text );
```

您还可以更改表列以将其默认值设置为序列计数器：

```
ALTER TABLE tablename ALTER COLUMN id SET DEFAULT
nextval('myserial');
```

序列回绕

默认情况下，序列不会回绕。这意味着当一个序列到达了最大值（SMALLSERIAL的+32767，SERIAL的+2147483647，BIGSERIAL的+9223372036854775807），再调用nextval()会失败。可以修改一个序列让它从1开始回绕：

```
ALTER SEQUENCE myserial CYCLE;
```

也可以在创建序列时指定回绕行为：

```
CREATE SEQUENCE myserial CYCLE;
```

引

Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统
 - 管理Greenplum数据库访问
 - 定义数据库对象
 - 创建和管理数据库
 - 创建和管理表空间
 - 创建和管理SCHEMA
 - 创建和管理表
 - 选择表存储模型
 - 对大型表分区
 - 创建和使用序列
 - 在Greenplum数据库中使用索引**
 - 创建和管理视图
 - 分布与倾斜
 - 插入、更新、和删除数据

- 查询数据
- 使用外部数据
- 装载和卸载数据
- 性能管理

在大部分传统数据库中，索引能够极大地改善数据访问时间。不过，在一个Greenplum之类的分布式数据库中，索引应该被更保守地使用。Greenplum数据库会执行非常快的顺序扫描，索引则使用一种随机搜索的模式在磁盘上定位记录。Greenplum的数据分布在Segment上，因此每个Segment会扫描全体数据的一小部分来得到结果。通过表分区，要扫描的数据量可能会更少。因为商业智能 (BI) 查询负载通常会返回非常大的数据集，使用索引并不是很有效。

首先在不加索引时尝试用户的查询负载。索引更有可能为OLTP负载改进性能，在那种场景中查询会返回一个单一记录或者数据的一个小的子集。在被压缩过的追加优化表上，索引也可以提高返回一个目标行集合的查询的性能，因为优化器在适当的时候可以使用一种索引访问方法而不是全表扫描。对于压缩过的数据，使用一种索引访问方法意味着只有必要的行会被解压。

Greenplum数据库会自动为带有主键的表创建PRIMARY KEY约束。要在在一个被分区的表上创建索引，就在用户创建的分区表上创建一个索引。该索引会被传播到Greenplum数据库所创建的所有子表上。不支持在Greenplum数据库为分区表创建的子表上创建索引。

注意一个UNIQUE CONSTRAINT（例如PRIMARY KEY CONSTRAINT）会隐式地创建一个UNIQUE INDEX，它必须包括分布键中所有的列以及任何分区键。UNIQUE CONSTRAINT会在整个表上被强制要求，包括所有的表分区（如果有）。

索引会增加一些数据库负担，它们使用存储空间并且在表被更新时需要被维护。要确保查询负载会用到用户创建的索引，并且检查用户增加的索引是否改进了查询性能（与表的顺序扫描相比）。要确定是否使用了索引，检查查询的EXPLAIN计划。参见[查询分析](#)。

在创建索引时请考虑以下几点。

- 用户的查询负载。索引能改进查询返回单一记录或者非常小的数据集的性能，例如OLTP负载。
- 压缩表。在被压缩过的追加优化表上，索引也可以提高返回一个目标行集合的查询的性能。对于压缩过的数据，一种索引访问方法意味着只有必要的行会被解压。
- 避免在频繁更新的列上建立索引。在一个被频繁更新的列上建立索引会增加该列被更新时所要求的写操作数据量。
- 创建选择性的B-树索引。索引选择度是一个列中具有的可区分值的

Greenplum database 5管理员

数量除以表中行数得到的比例。例如，如果一个表有1000行并且一个列中有800个可区分的值，则该索引的选择度为0.8，这还不错。唯一索引的选择度总是1.0，这是最好的选择度。Greenplum数据库只允许在分布键列上的唯一索引。

- 为低选择度的列使用位图索引。Greenplum数据库的位图索引类型在常规的PostgreSQL中不可用。参见[关于位图索引](#)。
- 索引在连接中用到的列。在被用于频繁连接的一个列（例如一个外键列）上的索引能够提升连接性能，因为这让查询优化器有更多的连接方法可以使用。
- 索引在谓词中频繁使用的列。频繁地在WHERE子句中被引用的列是索引的首选。
- 避免重叠的索引。具有相同前导列的索引是冗余的。
- 批量载入前删掉索引。对于载入大量数据到一个表中，请考虑先删掉索引并且在数据装载完成后重建它们。这常常比更新索引更快。
- 考虑一个聚簇索引。聚簇一个索引意味着记录会根据索引被物理排序后存储在磁盘上。如果用户需要的数据被随机分布在磁盘上，数据库必须在磁盘上来回寻找以取得所需的记录。如果这些记录被存储得彼此临近，那么取得它们的操作就会更高效。例如，一个在日期列上的聚簇索引中数据会按照日期顺序存放。针对一个指定日期范围的查询将会导致对磁盘的一次有序地读取，这会利用快速的顺序访问。

在Greenplum数据库中聚簇一个索引

使用CLUSTER命令根据一个索引从物理上重新排序一个非常大的表可能会花费很长的时间。为了更快达到同样的结果，用户可以通过创建一个中间表并且按照想要的顺序重载数据来手工在磁盘上重排数据。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;
```

Parent topic: 定义数据库对象

索引类型

Greenplum数据库数据库支持Postgres索引类型B-树、GiST和GIN，不支持Hash索引。每一种索引类型都使用一种不同的算法，它们最适合的查询类型也不同。B-树索引适合于最常见的情况并且是默认的索引类型。对于这些类型的描述请见PostgreSQL文档中的[索引类型](#)。

Note: 只有索引键的列与Greenplum分布键相同（或者是其超集）时，Greenplum数据库才允许唯一索引。在追加优化表上不支持唯一索引。在分区表上，唯一索引无法在一个分区表的所有子表分区之间被实施。唯一索引只能在一个分区实施。

关于位图索引

Greenplum数据库提供位图索引类型。位图索引最适合于拥有大量数据、很多临时查询以及少量数据修改（DML）事务的数据仓库应用和决策支持系统。

一个索引提供了指向表中包含一个给定键值的行的指针。常规索引存储了每个键存储了一个元组ID的列表，列表中的元组ID对应于具有那个键值的行。位图索引为每一个键值都存储一个位图。常规索引可能会比表中的数据大几倍，但位图索引提供了和常规索引相同的功能并且只需要被索引数据尺寸的一小部分。

位图中的每一个位对应于一个可能的元组ID。如果该位被设置，则具有相应元组ID的行包含该键值。一个映射函数负责将这个位的位置转换成一个元组ID。位图被压缩存储。如果可区分键值的数量很小，位图索引会小很多同时也会被压缩得更好，并且比常规索引节省可观的空间。一个位图索引的大小与该表中行数乘以被索引列中不同值数量的结果成比例。

位图索引对于在WHERE子句中包含多个条件的查询最有效。满足某些但不是全部条件的行在访问表之前就会被过滤掉。这通常会极大地改善响应时间。

何时使用位图索引

位图索引最适合用户只查询数据而不更新数据的数据仓库应用。对于拥有100至100,000个可区分值的列并且当被索引列常常与其他被索引列联合查询时，位图索引表现最好。低于100个可区分值的列通常无法从任何类型的索引受益，例如有两个可区分值的性别列（男和女）。而在具有超过100,000个可区分值的列上，位图索引的性能和空间效率会下降。

位图索引能够提升临时查询的查询性能。在将结果位图转换成元组ID之前，一个查询的WHERE子句中的AND以及OR条件可以通过在位图上直接执行相应的布尔操作快速地解决。如果结果行数很小，查询能够在不做全表扫描的情况下很快地被回答。

何时不用位图索引

不要为唯一列或者具有高基数数据的列使用位图索引，例如顾客姓名或者电话号码。位图索引的性能增益和磁盘空间优势在具有100,000或者更多唯一值的列上开始减小，这与表中的行数无关。

位图索引不适合有大量并发事务修改数据的OLTP应用。

请保守地使用位图索引。测试并且比较使用索引和不使用索引的查询性能。只有被索引列的查询性能有提升时才增加索引。

创建一个索引

CREATE INDEX命令在一个表上定义一个索引。例如，要在表*employee*的*gender*列上创建一个B-树索引：

```
CREATE INDEX gender_idx ON employee (gender);
```

要在表*title*中的列*title*上创建一个位图索引：

```
CREATE INDEX title_bmp_idx ON films USING bitmap (title);
```

表达式索引

索引列不必只是表的一列，而是可以是从表的一列或多列计算的函数或标量表达式。此功能对于根据计算结果快速访问表非常有用。

索引表达式的维护成本相对较高，因为必须在插入和每次更新时为每一行计算派生表达式。但是，索引表达式在索引搜索期间不会重新计算，因为它们已存储在索引中。在下面两个例子中，系统把查询视为WHERE indexedcolumn = 'constant'，所以查询的速度与其他普通索引相同。因此，当检索速度比插入和更新速度更重要时，表达式上的索引很有用。

第一个示例是使用lower函数进行不区分大小写的比较的常用方法：

```
SELECT * FROM test1 WHERE lower(col1) = 'value';
```

如果一个索引已经在lower(col1)函数的结果上被定义，该查询可以使用索引：

```
CREATE INDEX test1_lower_col1_idx ON test1 (lower(col1));
```

此示例假定经常执行以下类型的查询。

```
SELECT * FROM people WHERE (first_name || ' ' || last_name)
= 'John Smith';
```

查询可能会受益于以下索引。

```
CREATE INDEX people_names ON people ((first_name || ' ' || last_name));
```

CREATE INDEX命令的语法通常需要在索引表达式周围编写括号，如第二个示例所示。当表达式只是函数调用时，可以省略括号，如第一个示例中所示。

检查索引使用

Greenplum数据库的索引并不要求维护和调优。用户可以检查实际的查询负载使用了哪些索引。使用EXPLAIN命令可以检查一个查询的索引使用。

查询计划展示了数据库将用来回答一个查询的步骤或者计划节点以及每一个计划节点的时间估计。要检查索引的使用，请在用户的EXPLAIN输出中寻找以下查询计划节点类型：

- 索引扫描 - 一次索引的扫描。
- 位图堆扫描 - 检索所有由BitmapAnd、BitmapOr或者BitmapIndexScan生成的位图并且访问堆以检索相关的行。
- 位图索引扫描 - 计算一个由所有来自底层索引的满足查询谓词的位图通过OR操作形成的位图。
- **BitmapAnd或BitmapOr** - 取得从多个BitmapIndexScan节点生成的位图，把它们AND或者OR在一起，并且生成一个新的位图作为其

输出。

用户必须做实验来确定要创建哪些索引。请考虑以下几点。

- 在创建或者更新一个索引后运行ANALYZE。 ANALYZE会收集表统计信息。查询优化器使用表统计信息来估算一个查询所返回的行数并且为每一种可能的查询计划赋予实际开销。
- 实验中使用真实数据。使用测试数据建立索引会告诉用户该测试数据需要什么样的索引，但也仅此而已。
- 不要使用非常小的测试数据集，因为它们的结果很可能是不真实的或者倾斜的。
- 在开发测试数据时要小心。相似的、完全随机的或者排序后插入的值都将使统计信息偏离真实数据的分布。
- 通过使用运行时参数来关闭特定的计划类型，用户可以强制使用索引来进行测试。例如，关闭顺序扫描（enable_seqscan）以及嵌套循环连接（enable_nestloop）两种最基本的计划来强制系统使用一种不同的计划。对用户的查询使用索引和不用索引的执行进行计时，并且使用EXPLAIN ANALYZE命令来比较结果。

管理索引

使用REINDEX命令可以重建一个表现不好的索引。 REINDEX使用存储在一个索引的基表中的数据重建该索引来替换该索引。

重建一个表上的所有索引

```
REINDEX my_table;
```

```
REINDEX my_index;
```

删除一个索引

DROP INDEX命令移除一个索引。例如：

```
DROP INDEX title_idx;
```

在载入数据时，删除所有索引、载入数据然后重建索引会更快。

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

创建和管理数据库

创建和管理表空间

创建和管理SCHEMA

创建和管理表

选择表存储模型

对大型表分区

创建和使用序列

在Greenplum数据库中
使用索引

创建和管理视图

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

□ 性能管理

视图允许用户保存常用的或者复杂的查询，然后在一个SELECT语句中把它们当作表来访问。视图在磁盘上并没有被物理存储：当用户访问视图时查询会作为一个子查询运行。

如果一个子查询与一个单一查询相关联，考虑使用SELECT命令的WITH子句而不是创建一个很少使用的视图。

Parent topic: [定义数据库对象](#)

创建视图

CREATE VIEW命令定义一个查询的视图。例如：

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

视图会忽略存储在视图中的ORDER BY以及SORT操作。

删除视图

DROP VIEW命令删除一个视图。例如：

```
DROP VIEW topten;
```

DROP VIEW ... CASCADE命令也可以移除所有依赖的对象。例如，如果另一个视图依赖于将要被删除的视图，这个其他的视图也将被删除。如果没有CASCADE选项，这个DROP VIEW命令将会失败。



Greenplum数据库® 6.0文档

- 参考指南
- SQL Command Reference
- SQL 2008可选特性兼容性
- Greenplum环境变量
- 保留标识符和SQL关键字
- 系统目录参考
- gp_toolkit管理模式
- gpperfmon 数据库
- Greenplum 数据库数据类型
- 字符集支持
- 服务器配置参数
 - 参数类型和值
 - 设置参数
 - 参数类别
 - 配置参数**

按字母顺序列出Greenplum数据库服务器配置参数的说明。

● application_name	● gp_ignore_error_table (不推荐使用)	● max_files_per_process
● array_nulls	● gp_initial_bad_row_limit	● max_function_args
● authentication_timeout	● gp_instrument_shmem_size	● max_identifier_length
● backslash_quote	● gp_interconnect_debug_retry_interval	● max_index_keys
● block_size	● gp_interconnect_fc_method	● max_locks_per_transaction
● bonjour_name	● gp_interconnect_hash_multiplier	● max_prepared_transactions
● check_function_bodies	● gp_interconnect_queue_depth	● max_resource_portals_per_transaction
● client_encoding	● gp_interconnect_setup_timeout	● max_resource_queues
● client_min_messages	● gp_interconnect_snd_queue_depth	● max_stack_depth
● cpu_index_tuple_cost	● gp_interconnect_type	● max_statement_mem
● cpu_operator_cost	● gp_log_format	● memory_spill_ratio
● cpu_tuple_cost	● gp_log_fts	● optimizer
● cursor_tuple_fraction	● gp_log_interconnect	● optimizer_array_expansion_threshold
● data_checksums	● gp_log_gang	● optimizer_analyze_root_partition
● DateStyle	● gp_max_local_distributed_cache	● optimizer_control
● db_user_namespace	● gp_max_packet_size	● optimizer_cteInlining_bound
● deadlock_timeout	● gp_max_plan_size	● optimizer_enable_associativity
● debug_assertions	● gp_max_slices	● optimizer_enable_master_on_series
● debug_pretty_print	● gp_motion_cost_per_row	● optimizer_force_agg_skew_avoidance
● debug_print_parse	● gp_recursive_cte	● optimizer_force_multistage_agg
● debug_print_plan	● gp_reject_percent_threshold	● optimizer_force_three_stage_scalar_dqa
● debug_print_prelim_plan	● gp_reraise_signal	● optimizer_join_arity_for_associativity_commutativity
● debug_print_rewritten	● gp_resgroup_memory_policy	● optimizer_join_order
● debug_print_slice_table	● gp_resource_group_bypass	● optimizer_join_order_threshold
● default_statistics_target	● gp_resource_group_cpu_limit	● optimizer_mdcache_size
● default_tablespace	● gp_resource_group_memory_limit	● optimizer_metadata_caching
● default_text_search_config	● gp_resource_manager	● optimizer_minidump
● #default_transction_deferrable	● gp_resqueue_memory_policy	● optimizer_nestloop_factor
● default_transaction_isolation	● gp_resqueue_priority	● optimizer_parallel_union
● default_transaction_read_only	● gp_resqueue_priority_cpuscores_per_segment	● optimizer_print_missing_stats
● dynamic_library_path	● gp_resqueue_priority_sweeper_interval	● optimizer_print_optimization_stats

- [effective_cache_size](#)
- [enable_bitmapscan](#)
- [enable_groupagg](#)
- [enable_hashagg](#)
- [enable_hashjoin](#)
- [enable_indexscan](#)
- [enable_mergejoin](#)
- [enable_nestloop](#)
- [enable_seqscan](#)
- [enable_sort](#)
- [enable_tidscan](#)
- [escape_string_warning](#)
- [explain_pretty_print](#)
- [extra_float_digits](#)
- [fromCollapse_limit](#)
- [gp_adjust_selectivity_for_outerjoins](#)
- [gp_appendonly_compaction](#)
- [gp_appendonly_compaction_threshold](#)
- [gp_autostats_mode](#)
- [gp_autostats_mode_in_functions](#)
- [gp_autostats_on_change_threshold](#)
- [gp_cached_segworkers_threshold](#)
- [gp_command_count](#)
- [gp_connection_send_timeout](#)
- [gp_content](#)
- [gp_create_table_random_default_distribution](#)
- [gp_dbid](#)
- [gp_debug_linger](#)
- [gp_default_storage_options](#)
- [gp_dynamic_partition_pruning](#)
- [gp_enable_adaptive_nestloop](#)
- [gp_enable_agg_distinct](#)
- [gp_enable_agg_distinct_pruning](#)
- [gp_enable_direct_dispatch](#)
- [gp_role](#)
- [gp_safefswritesize](#)
- [gp_segment_connect_timeout](#)
- [gp_segments_for_planner](#)
- [gp_server_version](#)
- [gp_server_version_num](#)
- [gp_session_id](#)
- [gp_set_proc_affinity](#)
- [gp_set_read_only](#)
- [gp_statistics_pullup_from_child_partition](#)
- [gp_statistics_use_fkeys](#)
- [gp_use_legacy_hashops](#)
- [gp_vmem_idle_resource_timeout](#)
- [gp_vmem_protect_limit](#)
- [gp_vmem_protect_segworker_cache_limit](#)
- [gp_workfile_compression](#)
- [gp_workfile_limit_files_per_query](#)
- [gp_workfile_limit_per_query](#)
- [gp_workfile_limit_per_segment](#)
- [gpperfmon_port](#)
- [ignore_checksum_failure](#)
- [integer_datetimes](#)
- [IntervalStyle](#)
- [joinCollapse_limit](#)
- [keep_wal_segments](#)
- [krb_caseins_users](#)
- [krb_server_keyfile](#)
- [lc_collate](#)
- [lc_ctype](#)
- [lc_messages](#)
- [lc_monetary](#)
- [lc_numeric](#)
- [lc_time](#)
- [listen_addresses](#)
- [optimizer_sort_factor](#)
- [password_encryption](#)
- [password_hash_algorithm](#)
- [pljava_classpath](#)
- [pljava_classpath_insecure](#)
- [pljava_statement_cache_size](#)
- [pljava_release_lingering_savepoints](#)
- [pljava_vmoptions](#)
- [port](#)
- [random_page_cost](#)
- [readable_external_table_timeout](#)
- [repl_catchup_within_range](#)
- [replication_timeout](#)
- [regex_flavor](#)
- [resource_cleanup_gangs_on_wait](#)
- [resource_select_only](#)
- [runaway_detector_activation_percent](#)
- [search_path](#)
- [seq_page_cost](#)
- [server_encoding](#)
- [server_version](#)
- [server_version_num](#)
- [shared_buffers](#)
- [shared_preload_libraries](#)
- [ssl](#)
- [ssl_ciphers](#)
- [standard_conforming_strings](#)
- [statement_mem](#)
- [statement_timeout](#)
- [stats_queue_level](#)
- [superuser_reserved_connections](#)
- [tcp_keepalives_count](#)
- [tcp_keepalives_idle](#)
- [tcp_keepalives_interval](#)

- | | | |
|--------|-----|-------------------|
| 取值范围 | 默认值 | 设置分类 |
| string | | master
session |
- [gp_enable_exchange_default_partition](#)
 - [gp_enable_fast_sri](#)
 - [gp_enable_global_deadlock_detector](#)
 - [gp_enable_gpperfmon](#)
 - [gp_enable_groupext_distinct_gather](#)
 - [gp_enable_groupext_distinct_pruning](#)
 - [gp_enable_multiphase_agg](#)
 - [gp_enable_predicate_propagation](#)
 - [gp_enable_preunique](#)
 - [gp_enable_query_metrics](#)
 - [gp_enable_resize_collection](#)
 - [gp_enable_segment_copy_checking](#)
 - [gp_enable_sort_distinct](#)
 - [gp_enable_sort_limit](#)
 - [gp_external_enable_exec](#)
 - [gp_external_max_segs](#)
 - [gp_external_enable_filter_pushdown](#)
 - [gp_fts_probe_interval](#)
 - [gp_fts_probe_retries](#)
 - [gp_fts_probe_threadcount](#)
 - [gp_fts_probe_timeout](#)
 - [gp_global_deadlock_detector_period](#)
 - [gp_gpperfmon_send_interval](#)
 - [gpperfmon_log_alert_level](#)
 - [gp_hashjoin_tuples_per_bucket](#)
 - [local_preload_libraries](#)
 - [lock_timeout](#)
 - [log_autostats](#)
 - [log_connections](#)
 - [log_disconnections](#)
 - [log_dispatch_stats](#)
 - [log_duration](#)
 - [log_error_verbosity](#)
 - [log_executor_stats](#)
 - [log_hostname](#)
 - [log_min_duration_statement](#)
 - [log_min_error_statement](#)
 - [log_min_messages](#)
 - [log_parser_stats](#)
 - [log_planner_stats](#)
 - [log_rotation_age](#)
 - [log_rotation_size](#)
 - [log_statement](#)
 - [log_statement_stats](#)
 - [log_temp_files](#)
 - [log_timezone](#)
 - [log_truncate_on_rotation](#)
 - [maintenance_work_mem](#)
 - [max_appendonly_tables](#)
 - [max_connections](#)
 - [temp_buffers](#)
 - [TimeZone](#)
 - [timezone_abbreviations](#)
 - [track_activity_query_size](#)
 - [transaction_isolation](#)
 - [transaction_read_only](#)
 - [transform_null_equals](#)
 - [unix_socket_directory](#)
 - [unix_socket_group](#)
 - [unix_socket_permissions](#)
 - [update_process_title](#)
 - [vacuum_cost_delay](#)
 - [vacuum_cost_limit](#)
 - [vacuum_cost_page_dirty](#)
 - [vacuum_cost_page_hit](#)
 - [vacuum_cost_page_miss](#)
 - [vacuum_freeze_min_age](#)
 - [validate_previous_free_tid](#)
 - [verify_gpfdists_cert](#)
 - [vmem_process_interrupt](#)
 - [wal_receiver_status_interval](#)
 - [writable_external_table_bufsize](#)
 - [xid_stop_limit](#)
 - [xid_warn_limit](#)
 - [xmlobinary](#)
 - [xmloption](#)

application_name

设置客户端会话的应用程序名称。例如，如果通过psql连接，则将其设置为psql。设置应用程序名称允许在日志消息和统计信息视图中报告。

取值范围	默认值	设置分类
string		master session

reload

array_nulls

这可以控制数组输入解析器是否将未加引号的NULL识别为指定空数组元素。默认情况下，此选项处于启用状态，允许输入包含空值的数组值。3.0之前的Greenplum数据库版本不支持数组中的空值，因此将NULL视为指定字符串值为“NULL”的普通数组元素。

取值范围	默认值	设置分类
Boolean	on	master session reload

authentication_timeout

完成客户端认证的最大时间，这样可以防止挂起的客户端无限期占用连接。

取值范围	默认值	设置分类
任何有效的时间表达式（数字和单位）	1分钟	local system restart

backslash_quote

这可以控制是否在字符串中可以用\表示引号。代表引号的首选SQL标准是用“”表示，但是PostgreSQL历来也使用\。但是，使用\会导致安全风险，因为在一些客户端字符集编码中，有很多字节字符，其中最后一个字节等同于ASCII字符\。

取值范围	默认值	设置分类
on (总是允许 \)	safe_encoding	master
off (总是拒绝)		session
safe_encoding (只有客户端编码不允许多字节中的ASCII字符才允许)		reload

block_size

报告磁盘块大小。

取值范围	默认值	设置分类
字节数	32768	read only

bonjour_name

指定Bonjour广播名称。默认情况下，使用计算机名称，指定为空字符串。如果服务器未支持Bonjour服务，则忽略此选项。

取值范围	默认值	设置分类
string	unset	master system restart

check_function_bodies

设置为off时，在CREATE FUNCTION期间禁用函数体字符串的验证。在从转储中恢复函数定义时，禁用验证有时可以避免诸如前向引用之类的问题。

取值范围	默认值	设置分类
Boolean	on	master session reload

client_encoding

设置客户端编码（字符集）。默认是使用与数据库相同的编码。请参阅PostgreSQL文档中的[支持的字符集](#)。

取值范围	默认值	设置分类
字符集	UTF8	master session reload

client_min_messages

控制哪些消息级别发送到客户端。每个级别包括跟它随后的所有级别，越往后的级别，发送的消息就越少。

取值范围	默认值	设置分类
DEBUG5	NOTICE	master
DEBUG4		session
DEBUG3		reload
DEBUG2		
DEBUG1		
LOG		
NOTICE		
WARNING		
ERROR		
FATAL		
PANIC		

cpu_index_tuple_cost

对于传统的查询优化器（planner），在索引扫描期间设置对处理每个索引行代价的估计。这是作为顺序页面提取代价的一部分来衡量的。

取值范围	默认值	设置分类
浮点数	0.005	master session

reload

cpu_operator_cost

对于传统的查询优化器（planner），设置对处理WHERE语句中每个操作符代价的估计。这是作为顺序页面提取代价的一部分来衡量的。

取值范围	默认值	设置分类
浮点数	0.0025	master session reload

cpu_tuple_cost

对于传统的查询优化器（planner），设置对处理一个查询中每行（元组）代价的估计。这是作为顺序页面提取代价的一部分来衡量的。

取值范围	默认值	设置分类
浮点数	0.01	master session reload

cursor_tuple_fraction

告知传统查询优化器（planner）预期在游标查询中提取多少行，从而允许传统优化器使用此信息来优化查询计划。默认值为1表示获取所有行。

取值范围	默认值	设置分类
整数	1	master session reload

data_checksums

报告是否为数据库系统中的堆数据存储启用了校验和。 初始化数据库系统且无法更改时，将启用或禁用堆数据的校验和。

堆数据页存储堆表，目录表，索引和数据库元数据。 追加优化存储具有与此参数无关的内置校验和支持。

Greenplum数据库使用校验和来防止将文件系统中损坏的数据加载到由数据库进程管理的内存中。 启用堆数据校验和时，Greenplum Database会在堆数据页写入磁盘时计算并存储校验和。 从磁盘检索页面时，将验证校验和。 如果验证失败，则会生成错误，并且不允许将页面加载到托管内存中。

如果[ignore_checksum_failure](#)配置参数已设置为on，则校验和验证失败会生成警告，但允许将页面加载到托管内存中。 如果页面随后更新，则会刷新到磁盘并复制到镜像。 这可能导致数据损坏传播到镜像并阻止完全恢复。 由于可能会丢失数据，因此只应在需要恢复数据时启用[ignore_checksum_failure](#)参数。 有关更多信息，请参阅[ignore_checksum_failure](#)。

取值范围	默认值	设置分类
Boolean	on	read only

DateStyle

设置日期和时间值的显示格式，以及解释模糊日期输入值的规则。该变量值包含两个独立的而部分：输出格式规范和输入输出规范中年月日的顺序。

取值范围	默认值	设置分类
<format>, <date style> 其中： <format> 是ISO, Postgres, SQL或German <date style> 是DMY, MDY或YMD	ISO, MDY	master session reload

db_user_namespace

这启用了每个数据库的用户名。 如果打开，用户应该以*username@dbname* 创建用户。 要创建普通的全局用户，只需要在客户端指定用户名时附加@。

取值范围	默认值	设置分类
Boolean	off	local system

restart

deadlock_timeout

在检查以查看是否存在死锁情况之前等待锁的时间。在一个比较重的服务器上，用户可能希望提高此值。理想的情况下，设置的值应该超过用户的典型处理时间，以此提高在等待线程在决定检查死锁之前自动解锁的几率。

取值范围	默认值	设置分类
任何有效时间的表达式（数字或者单位）。	1s	local system restart

debug_assertions

打开各种断言检查。

取值范围	默认值	设置分类
Boolean	off	local system restart

debug_pretty_print

缩进调试输出产生更可读但是更长的输出格式。`client_min_messages` 或者 `log_min_messages` 必须是DEBUG1或者更低。

取值范围	默认值	设置分类
Boolean	on	master session reload

debug_print_parse

对于每一个执行的查询，打印出结果分析树。*client_min_messages* 或 *log_min_messages* 必须是DEBUG1或者更低。

取值范围	默认值	设置分类
Boolean	off	master session reload

debug_print_plan

对于每个执行的查询，打印出Greenplum并行查询执行计划。*client_min_messages* 或 *log_min_messages* 必须是DEBUG1或者更低。

取值范围	默认值	设置分类
Boolean	off	master session reload

debug_print_prelim_plan

对于每个执行的查询，打印出初步查询计划。*client_min_messages* 或 *log_min_messages* 必须是DEBUG1或者更低。

取值范围	默认值	设置分类
Boolean	off	master session reload

debug_print_rewritten

对于每个执行的查询，打印出查询重写输出。*client_min_messages* 或 *log_min_messages* 必须是DEBUG1或者更低。

取值范围	默认值	设置分类
Boolean	off	master session reload

debug_print_slice_table

对于每个执行的查询，打印Greenplum查询分片计划。`client_min_messages` 或`log_min_messages` 必须是DEBUG1或者更低。

取值范围	默认值	设置分类
Boolean	off	master session reload

default_statistics_target

通过ALTER TABLE SET STATISTICS设置未具有列特定目标集的表列，默认统计信息采样目标（存储在公共值列表中的值的数量）。较大的值可能会提高Postgres查询优化器（规划器）估计的质量。

取值范围	默认值	设置分类
0 < 整数 < 10000	100	master session reload

default_tablespace

当CREATE命令没有明确指定一个表空间，会在默认的表空间创建对象（表和索引）。

取值范围	默认值	设置分类
表空间的名字	unset	master session

reload

default_text_search_config

选择文本搜索功能的那些变体使用的文本搜索配置，这些变体没有指定配置的显式参数。有关详细信息，请参阅[使用全文搜索](#)。内置缺省值为`pg_catalog.simple`，但是如果可以识别与该语言环境匹配的配置，则`initdb`将使用与所选`lc_ctype`语言环境对应的设置初始化配置文件。

取值范围	默认值	设置分类
文本搜索配置的名称。	<code>pg_catalog.simple</code>	master session reload

default_transaction_deferrable

Note: 只有只读和可串行化事务才可以被推迟。Greenplum数据库不支持SERIALIZABLE事务隔离级别，因此将`default_transaction_deferrable`设置为on对Greenplum数据库没有影响。

在可序列化隔离级别运行时，可延迟的只读SQL事务可能会在允许继续之前被延迟。但是，一旦它开始执行，它不会产生确保可串行化所需的任何开销；因此，序列化代码没有理由强制它因并发更新而中止，所以此选项适用于长时间运行的只读事务。

此参数控制每个新事务的默认可延迟状态。它目前对读写事务或低于可序列化的隔离级别的操作没有影响。默认为关闭。

取值范围	默认值	设置分类
Boolean	<code>off</code>	master session reload

default_transaction_isolation

控制每个新事务的默认隔离级别。Greenplum数据库将`read uncommitted`视为与`read committed`相同，并将`serializable`视为与`repeatable read`相同。

取值范围	默认值	设置分类
read committed	read committed	master
read uncommitted		session
repeatable read		reload
serializable		

default_transaction_read_only

控制每个新事务的默认只读状态。只读的SQL事务不能修改非临时表。

取值范围	默认值	设置分类
Boolean	off	master session reload

dynamic_library_path

如果需要打开动态加载的模块，并且在CREATE FUNCTION或LOAD命令中指定的文件名没有目录部分（即：目录不包括斜杠），系统会搜索该路径以获取所需的文件。此时，PostgreSQL内置编译的包库目录会替换\$libdir。这是由标准PostgreSQL发行版提供的模块安装位置。

取值范围	默认值	设置分类
由冒号分隔的绝对目录路径列表	\$libdir	local system restart

effective_cache_size

设置有关Postgres查询优化器（计划程序）的单个查询可用的磁盘高速缓存的有效大小的假设。这是用于估算使用指数的成本的因素；较高的值使得更有可能使用索引扫描，较低的值使得更有可能使用顺序扫描。此参数对Greenplum服务器实例分配的共享内存大小没有影响，也不保留内核磁盘缓存；它仅用于估算目的。

将此参数设置为32K块的数量（例如，对于16MB，为512），或指定有效缓存的大小（例如，对于1024个块，为'32MB'）。`gpconfig`工具和`SHOW`命令以“MB”或“kB”为单位显示有效的高速缓存大小值。

取值范围	默认值	设置分类
浮点数	512 (16GB)	master session reload

enable_bitmapscan

启用或禁用Postgres查询优化器（规划器）使用位图扫描计划类型。请注意，这与位图索引扫描不同。位图扫描意味着索引将在适当的时候在内存中动态转换为位图，从而在针对非常大的表的复杂查询上提供更快的索引性能。当不同的索引列上有多个谓词时使用它。可以比较每列的每个位图以创建所选元组的最终列表。

取值范围	默认值	设置分类
Boolean	on	master session reload

enable_groupagg

启用或禁用Postgres查询优化器（规划器）使用组聚合计划类型。

取值范围	默认值	设置分类
Boolean	on	master session reload

enable_hashagg

启用或禁用Postgres查询优化器（规划器）使用哈希聚合计划类型。

取值范围	默认值	设置分类
Boolean	on	master session reload

enable_hashjoin

启用或禁用Postgres查询优化器（规划器）使用哈希连接计划类型。

取值范围	默认值	设置分类
Boolean	on	master session reload

enable_indexscan

启用或禁用Postgres查询优化器（规划器）使用索引扫描计划类型。

取值范围	默认值	设置分类
Boolean	on	master session reload

enable_mergejoin

启用或禁用Postgres查询优化器（规划器）使用合并连接计划类型。 合并连接基于将左侧和右侧表按顺序排序然后并行扫描的想法。 因此，两种数据类型必须能够完全排序，并且连接运算符必须是只能成功处于排序顺序中“相同位置”的值对的连接运算符。 实际上，这意味着连接运算符必须表现得像相等一样。

取值范围	默认值	设置分类
Boolean	off	master

		session
		reload

enable_nestloop

启用或禁用Postgres查询优化器（规划器）使用嵌套循环连接计划。不可能完全抑制嵌套循环连接，但如果有其他可用方法，则关闭此变量会阻止Postgres优化器使用它。

取值范围	默认值	设置分类
Boolean	off	master session reload

enable_seqscan

启用或禁用Postgres查询优化器（规划器）使用顺序扫描计划类型。不可能完全抑制顺序扫描，但如果有其他方法可用，则关闭此变量会阻止Postgres优化器使用。

取值范围	默认值	设置分类
Boolean	on	master session reload

enable_sort

启用或禁用Postgres查询优化器（规划器）使用显式排序步骤。完全禁止显式排序是不可能的，但如果有其他可用方法，则关闭此变量会阻止Postgres优化器使用。

取值范围	默认值	设置分类
Boolean	on	master session

reload

enable_tidscan

启用或禁用Postgres查询优化器（规划器）使用元组标识符（TID）扫描计划类型。

取值范围	默认值	设置分类
Boolean	on	master session reload

escape_string_warning

打开的时候，如果在普通字符串文字（‘...’语法）中出现反斜杠（\），则会发出警告。转义字符语法（E'...'）应用于转义，因为在将来的版本中，普通字符串将具有字面上处理反斜杠的符合SQL标准的行为。

取值范围	默认值	设置分类
Boolean	on	master session reload

explain.pretty_print

确定 EXPLAIN VERBOSE 是否使用缩进或非缩进格式显示详细的查询树信息。

取值范围	默认值	设置分类
Boolean	on	master session reload

extra_float_digits

调整浮点值显示的位数，包括float4, float8, 和几何数据类型。将参数将加到数位上。该值可以设置为高达2，包括部分有效位。这对于转储需要精确恢复的浮点数据尤其有用。或者设置为负以摒弃不需要的位。

取值范围	默认值	设置分类
integer	0	master session reload

fromCollapseLimit

Postgres查询优化器（规划器）将子查询合并到上层查询中，如果生成的FROM列表只有这么多项。较小的值会减少计划时间，但可能会产生较差的查询计划。

取值范围	默认值	设置分类
1-n	20	master session reload

gp_adjust_selectivity_for_outerjoins

在外连接上启用NULL测试的选择性。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_appendonly_compaction

在VACUUM命令期间启用压缩segment文件。 禁用时， VACUUM仅将segment文件截断为EOF值，当前行为也是如此。 管理员可能希望在高I/O负载情况或低空间情况下禁用压缩。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_appendonly_compaction_threshold

指定在没有FULL选项（惰性VACUUM）的情况下运行VACUUM时隐藏行与触发压缩segment文件的总行的阈值比率（百分比）。如果segment上的segment文件中的隐藏行的比率小于此阈值，则不压缩段文件，并发出日志消息。

取值范围	默认值	设置分类
整数(%)	10	master session reload

gp_autostats_mode

指定使用ANALYZE触发自动统计信息收集的模式。 on_no_stats选项可以触发对任何没有统计信息的表上的CREATE TABLE AS SELECT, INSERT, 或COPY操作的统计信息收集。

当受影响的行数超过由gp_autostats_on_change_threshold定义的阈值时， on_change选项才会触发统计信息收集。 可以使用on_change触发自动统计信息收集的操作有：

CREATE TABLE AS SELECT

UPDATE

DELETE

INSERT

COPY

默认值是on_no_stats。

Note: 对于分区表来说，如果从分区表的顶级父表插入数据，则不会触发自动统计信息收集。

如果数据直接插入到分区表的叶表（数据的存储位置）中，则触发自动统计信息收集。统计数据仅在叶表上收集。

取值范围	默认值	设置分类
none	on_no_stats	master
on_change		session
on_no_stats		reload

gp_autostats_mode_in_functions

指定使用过程语言函数中的ANALYZE语句触发自动统计信息收集的模式。none选项禁用统计信息收集。on_no_stats选项在任何没有现有统计信息表上的函数中执行的CREATE TABLE AS SELECT, INSERT, 或COPY操作触发统计信息收集。

只有当受影响的行数超过由gp_autostats_on_change_threshold定义的阈值时，on_change选项才会触发统计信息收集。可以使用on_change触发自动信息统计收集功能的操作有：

CREATE TABLE AS SELECT

UPDATE

DELETE

INSERT

COPY

取值范围	默认值	设置分类
none	none	master
on_change		session
on_no_stats		reload

gp_autostats_on_change_threshold

当gp_autostats_mode设定为on_change时，指明自动统计信息收集的阈值。当触发表操作影响超过此阈值的行数时，将添加ANALYZE并收集表的统计信息。

取值范围	默认值	设置分类
整数	2147483647	master session reload

gp_cached_segworkers_threshold

当用户使用Greenplum数据库启动会话并发出查询时，系统会在每个segment上创建工作进程的组或“gangs”来完成工作。完成工作后，除了由此参数设置的缓存数之外，将销毁段工作进程。较低的设置可以节省段主机上的系统资源，但更高的设置可以提高想要连续发出许多复杂查询的高级用户的性能。

取值范围	默认值	设置分类
整数 > 0	5	master session reload

gp_command_count

显示主服务器从客户端收到的命令数。请注意，单个SQL命令实际上可能在内部涉及多个命令，因此对于单个查询，计数器可能会增加多个。该计数器也由处理该命令的所有segment进程共享。

取值范围	默认值	设置分类
整数 > 0	1	read only

gp_connection_send_timeout

在查询处理期间向无响应的Greenplum数据库用户客户端发送数据的超时。值为0将禁用超时，Greenplum数据库将无限期地等待客户端。达到超时后，将使用以下消息取消查询：

```
Could not send data to client: Connection timed out.
```

取值范围	默认值	设置分类
秒数	3600 (1小时)	master

		system reload
--	--	------------------

gp_content

Segment的本地内容ID。

取值范围	默认值	设置分类
整数		read only

gp_create_table_random_default_distribution

使用不包含DISTRIBUTED BY子句的CREATE TABLE或CREATE TABLE AS 创建Greenplum数据库表时，控制表的创建。

对于CREATE TABLE，如果参数的值为off（缺省值），并且创建表命令不包含DISTRIBUTED BY子句，Greenplum数据库将根据以下命令选择表分布键：

- 如果指定了LIKE或INHERITS子句，则Greenplum将从源表或父表复制分布键。
- 如果指定了PRIMARY KEY或UNIQUE约束，则Greenplum会选择所有键列的最大子集作为分布键。
- 如果既没有指定约束也没有指定LIKE或INHERITS子句，则Greenplum选择第一个合适的列作为分布键。（具有几何或用户定义数据类型的列不符合Greenplum分布键列的条件。）

如果参数的值设置为on，则当未指定DISTRIBUTED BY子句时，Greenplum数据库将遵循这些规则来创建表：

- 如果未指定PRIMARY KEY或UNIQUE列，则表的分布是随机的（DISTRIBUTED RANDOMLY）。即使表创建命令包含LIKE或INHERITS子句，表分发也是随机的。
- 如果指定了PRIMARY KEY或UNIQUE列，则还必须指定DISTRIBUTED BY子句。如果未将DISTRIBUTED BY子句指定为表创建命令的一部分，则该命令将失败。

对于不包含分布子句的CREATE TABLE AS 命令：

- 如果Postgres查询优化器创建表，并且参数的值为off，则根据该命令确定表分发策略。
- 如果Postgres查询优化器创建表，并且参数的值为on，则表分发策略是随机的。
- 如果GPORCA创建表，则表分发策略是随机的。参数值没有影响。

有关Postgres查询优化器和GPORCA的信息，请参阅Greenplum数据库管理员指南中的“查询数据”。

取值范围	默认值	设置分类
boolean	off	master system

reload

gp_dbid

对于segment，则为本地dbid。

取值范围	默认值	设置分类
整数		read only

gp_debug_linger

在致命的内部错误之后，Greenplum进程保留的秒数。

取值范围	默认值	设置分类
任何有效的时间表达式（数字和单位）	0	master session reload

gp_default_storage_options

使用CREATE TABLE命令创建表时，设置以下表存储选项的默认值。

- appendoptimized

Note: 您可以使用appendoptimized=value语法指定追加优化的表存储类型。appendoptimized是appendonly传统存储选项的精简别名。Greenplum数据库在catalog中存储appendonly，并在列出追加优化表的存储选项时显示相同内容。

- blocksize
- checksum
- compressstype
- compresslevel
- orientation

将多个存储选项值指定为逗号分隔列表。

您可以使用此参数设置存储选项，而不是在CREATE TABLE命令的WITH中指定表存储选项。使用CREATE TABLE命令指定的表存储选项会覆盖此参数指定的值。

并非所有存储选项值组合都有效。如果指定的存储选项无效，则返回错误。有关表存储选项的信息，请参阅CREATE TABLE命令。

可以为数据库和用户设置默认值。如果服务器配置参数设置在不同的级别，则当用户登录到数据库并创建表时，这是表存储值的优先顺序，从最高到最低：

1. CREATE TABLE命令中使用WITH子句或ENCODING子句指定的值
2. 使用ALTER ROLE...SET命令为用户设置的gp_default_storage_options的值
3. 使用ALTER DATABASE...SET命令为数据库设置的gp_default_storage_options的值
4. 使用gpconfig工具为Greenplum数据库系统设置的gp_default_storage_options的值

参数值不是累积的。例如，如果参数指定数据库的appendoptimized和compressstype选项并且用户登录并设置参数以指定orientation选项的值，则忽略在数据库级别设置的appendoptimized和compressstype值。

此示例ALTER DATABASE命令为数据库mytest设置默认orientation和compressstype表存储选项。

```
ALTER DATABASE mytest SET gp_default_storage_options = 'orientation=column, compressstype=rle_type'
```

使用面向列的表和RLE压缩在mytest数据库中创建追加优化的表。用户需要在WITH子句中仅指定appendoptimized=TRUE。

此示例gpconfig工具命令设置Greenplum数据库系统的默认存储选项。如果为多个表存储选项设置默认值，则该值必须用单引号括起来。

```
gpconfig -c 'gp_default_storage_options' -v 'appendoptimized=true, orientation=column'
```

此示例gpconfig工具命令显示参数的值。参数值必须在Greenplum数据库主数据库和所有segment之间保持一致。

```
gpconfig -s 'gp_default_storage_options'
```

取值范围	默认值	设置分类 ¹
appendoptimized= TRUE FALSE	appendoptimized=FALSE	master
blocksize= integer between 8192 and 2097152	blocksize=32768	session
checksum= TRUE FALSE	checksum=TRUE	reload
compressstype= ZLIB ZSTD QUICKLZ ² RLE_TYPE NONE	compressstype=none	
compresslevel= integer between 0 and 19	compresslevel=0	
orientation= ROW COLUMN	orientation=ROW	

Note: ¹当参数使用gpconfig工具设置在系统级时，为集合分类。

Note: QuickLZ 压缩仅在Pivotal Greenplum数据库的商业版本中可用。

gp_dynamic_partition_pruning

启用可以动态消除分区扫描的计划。

取值范围	默认值	设置分类
on/off	on	master session reload

gp_enable_adaptive_nestloop

允许在Postgres查询优化器（规划器）的查询执行时使用称为“Adaptive Nestloop”的新类型的连接节点。如果连接外侧的行数超过预先计算的阈值，这会导致Postgres优化器相比嵌套循环连接更偏爱哈希连接。此参数提高了索引操作的性能，这些操作以前支持较慢的嵌套循环连接。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_agg_distinct

启用或禁用两阶段聚合以计算单个不同限定的聚合。这仅适用于包含单个不同限定聚合函数的子查询。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_agg_distinct_pruning

启用或禁用三阶段聚合和连接以计算不同限定的聚合。这仅适用于包含一个或多个不同限定聚合函数的子查询。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_direct_dispatch

启用或禁用为访问单个segment上的数据的查询调度目标查询计划。启用时，单个segment上的目标行的查询将仅将其查询计划分派到该segment（而不是所有segment）。这明显缩短了限定查询的响应时间，因为没有涉及互连设置。直接分发确实需要主服务器上更多的CPU利用率。

取值范围	默认值	设置分类
Boolean	on	master system restart

gp_enable_exchange_default_partition

控制ALTER TABLE的EXCHANGE DEFAULT PARTITION子句的可用性。参数的默认值为off。如果在ALTER TABLE命令中指定了该子句，该子句不可用，Greenplum数据库将返回错误。

如果该值为on，则Greenplum数据库会返回一条警告，指出由于默认分区中的数据无效，交换默认分区可能会导致错误的结果。

Warning: 在交换默认分区之前，必须确保要交换的表中的数据（新的默认分区）对默认分区有效。例如，新默认分区中的数据不得包含在分区表的其他叶子分区中有效的数据。否则，使用GPORCA执行的交换的默认分区对分区表的查询可能会返回不正确的结果。

取值范围	默认值	设置分类
Boolean	off	master session reload

gp_enable_fast_sri

当设置为on时，Postgres查询优化器（planner）计划单行插入，以便将它们直接发送到正确的segment实例（无需motion操作）。这显着提高了单行插入语句的性能。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_global_deadlock_detector

控制是否启用Greenplum数据库全局死锁检测器来管理堆表上的并发UPDATE和DELETE操作以提高性能。见[全局死锁检测](#)。默认为off，全局死锁检测被禁用。

如果禁用全局死锁检测器（默认），Greenplum数据库将串行地对堆表执行并发更新和删除操作。

如果启用了全局死锁检测器，则允许并发更新，并且全局死锁检测器确定何时存在死锁，并通过终止与所涉及的最新事务关联的一个或多个后端进程来中断死锁。

取值范围	默认值	设置分类
Boolean	off	master system restart

gp_enable_gpperfmon

启用或禁用为Greenplum Command Center填充gpperfmon数据库的数据收集代理。

取值范围	默认值	设置分类
Boolean	off	local system

restart

gp_enable_groupext_distinct_gather

启用或禁用将数据收集到单个节点以计算分组扩展查询的不同限定聚合。当此参数和`gp_enable_groupext_distinct_pruning`都启用时，Postgres查询优化器（规划器）使用成本更低的计划。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_groupext_distinct_pruning

启用或禁用三阶段聚合和联接以计算分组扩展查询的不同限定聚合。通常，启用此参数会生成代价更小的查询计划，Postgres查询优化程序（优化器）将优先使用现有计划。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_multiphase_agg

启用或禁用使用两阶段或三阶段并行聚合计划Postgres查询优化器（planner）。此方法适用于具有聚合的任何子查询。如果`gp_enable_multiphase_agg`关闭，则禁用`gp_enable_agg_distinct`和`gp_enable_agg_distinct_pruning`。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_predicate_propagation

启用后，Postgres查询优化器（planner）会在表格在其分布键列上连接的情况下将查询谓词应用于两个表表达式。在进行连接之前过滤两个表（如果可能）更有效。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_preunique

为SELECT DISTINCT查询启用两阶段重复删除（不是SELECT COUNT(DISTINCT））。启用后，它会在移动之前添加一组额外的SORT DISTINCT计划节点。在不同操作大大减少行数的情况下，这个额外的SORT DISTINCT比通过互连发送行的成本代价低得多。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_query_metrics

启用查询指标的收集。启用查询指标收集后，Greenplum数据库会在查询执行期间收集指标。默认为关闭。

更改此配置参数后，必须重新启动Greenplum数据库才能使更改生效。

Greenplum数据库指标收集扩展在启用时，会将收集的指标通过UDP发送到Pivotal Greenplum Command Center代理¹。

Note: ¹指标收集扩展包含在Pivotal的Greenplum数据库的商业版本中。Pivotal Greenplum Command Center仅支持Pivotal Greenplum数据库。

取值范围	默认值	设置分类
Boolean	off	master system

restart

gp_enable_relsize_collection

如果表没有统计信息，则允许GPORCA和Postgres查询优化器（planner）使用表的估计大小（`pg_relation_size`函数）。默认情况下，如果统计信息不可用，GPORCA和计划程序将使用默认值来估计行数。默认行为可以改善查询优化时间并减少繁重工作负载中的资源队列使用，但可能导致计划不理想。

对于分区表的根分区，将忽略此参数。启用GPORCA且根分区没有统计信息时，GPORCA始终使用默认值。您可以使用`ANALYZE ROOTPARTITION`收集根分区的统计信息。请参见[ANALYZE](#)。

取值范围	默认值	设置分类
Boolean	off	master session reload

gp_enable_segment_copy_checking

控制在使用`COPY FROM... ON SEGMENT`命令将数据复制到表中时是否检查表的分发策略（来自表`DISTRIBUTED`子句）。如果为`true`，则如果一行数据违反了`segment`实例的分发策略，则会返回错误。默认值为`true`。

如果值为`false`，则不检查分发策略。添加到表中的数据可能违反了`segment`实例的表分发策略。可能需要手动重新分配表数据。请参阅`ALTER TABLE`子句`WITH REORGANIZE`。

The parameter can be set for a database system or a session. The parameter cannot be set for a specific database.

取值范围	默认值	设置分类
Boolean	true	master session reload

gp_enable_sort_distinct

在排序时启用删除重复项。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_sort_limit

在排序时启用LIMIT操作。当计划最多需要第一行*limit_number*时，排序更有效。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_external_enable_exec

启用或禁用在segment主机上执行OS命令或脚本的外部表的使用（CREATE EXTERNAL TABLE EXECUTE语法）。如果使用Command Center或MapReduce功能，则必须启用。

取值范围	默认值	设置分类
Boolean	on	master system restart

gp_external_max_segs

设置在外部表操作期间扫描外部表数据的segment数量，目的是使系统不要超载扫描数据并从其他并发操作中夺走资源。这仅适用于使用gpfdist://协议访问外部表数据的外部表。

取值范围	默认值	设置分类

integer	64	master session reload
---------	----	-----------------------------

gp_external_enable_filter_pushdown

从外部表读取数据时启用过滤器下推。如果下推失败，则执行查询而不将过滤器推送到外部数据源（相反，Greenplum数据库会对结果应用相同的约束）。有关更多信息，请参见[定义外部表](#)。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_fts_probe_interval

指定故障检测过程的轮询间隔 (ftsprobe)。ftsprobe过程将花费大约这段时间来检测段故障。

取值范围	默认值	设置分类
10 - 3600秒	1分钟	master system restart

gp_fts_probe_retries

指定在报告segment故障之前故障检测过程 (ftsprobe) 尝试连接到segment的次数。

取值范围	默认值	设置分类
integer	5	master system

		restart
--	--	---------

gp_fts_probe_threadcount

指定要创建的`ftsprobe`线程数。此参数应设置为等于或大于每个主机的segment数量的值。

取值范围	默认值	设置分类
1 - 128	16	master system restart

gp_fts_probe_timeout

指定故障检测过程（`ftsprobe`）允许的超时，以在声明它之前建立与segment的连接。

取值范围	默认值	设置分类
10 - 3600秒	20秒	master system restart

gp_global_deadlock_detector_period

指定全局死锁检测器后端进程的执行间隔（以秒为单位）。

取值范围	默认值	设置分类
5 - INT_MAX秒	120秒	master system reload

gp_log_fts

控制故障检测过程 (`ftsprobe`) 写入日志文件的详细信息量。

取值范围	默认值	设置分类
OFF	TERSE	master
TERSE		system
VERBOSE		restart
DEBUG		

gp_log_interconnect

控制写入日志文件的有关Greenplum数据库segment实例工作进程之间通信的信息量。 默认值为`terse`。 日志信息将写入master实例日志和segment实例日志。

增加日志记录量可能会影响性能并增加磁盘空间使用量。

取值范围	默认值	设置分类
off	terse	master
terse		session
verbose		reload
debug		

gp_log_gang

控制写入日志文件的有关查询工作进程创建和查询管理的信息量。 默认值为`OFF`， 不记录信息。

取值范围	默认值	设置分类
OFF	OFF	master
TERSE		session
VERBOSE		restart

DEBUG

gp_gpperfmon_send_interval

设置Greenplum数据库服务器将查询执行更新发送到用于为Command Center填充gpperfmon数据库的数据收集代理的频率。在此间隔期间执行的查询操作通过UDP发送到segment监视器代理。如果在长时间运行的复杂查询期间发现丢弃了过多的UDP数据包，则可以考虑增加此值。

取值范围	默认值	设置分类
任何有效的时间表达式（数量和单位）	1sec	master system restart

gpperfmon_log_alert_level

控制将哪些消息级别写入gpperfmon日志。每个级别包括其后的所有级别。级别越靠后，发送到日志的消息越少。

Note: 如果安装了gpperfmon数据库并且正在监视数据库，则默认值为warning。

取值范围	默认值	设置分类
none	none	local
warning		system
error		restart
fatal		
panic		

gp_hashjoin_tuples_per_bucket

设置HashJoin操作使用的哈希表的目标密度。较小的值往往会产生较大的哈希表，这可以提高连接性能。

取值范围	默认值	设置分类
integer	5	master

		session
		reload

gp_ignore_error_table (不推荐使用)

在CREATE EXTERNAL TABLE或COPY命令中指定不推荐使用的INTO ERROR TABLE子句时，控制Greenplum数据库行为。

默认值为false，如果在命令中指定了INTO ERROR TABLE子句，Greenplum数据库将返回错误。

如果值为true，则Greenplum数据库会忽略该子句，发出警告，并在没有INTO ERROR TABLE子句的情况下执行该命令。在Greenplum数据库5.x及更高版本中，您可以使用内置SQL函数访问错误日志信息。请参见[CREATE EXTERNAL TABLE](#)或[COPY](#)命令。

您可以将此值设置为true，以避免在运行包含Greenplum数据库4.3.x INTO ERROR TABLE子句的CREATE EXTERNAL TABLE或COPY命令的应用程序时出现Greenplum数据库错误。

在Greenplum数据库5中不推荐使用INTO ERROR TABLE子句并将其删除。在Greenplum数据库7中，此参数也将被删除，导致所有INTO ERROR TABLE调用产生语法错误。

取值范围	默认值	设置分类
Boolean	false	master session reload

gp_initial_bad_row_limit

对于参数值n，Greenplum数据库在使用COPY命令或从外部表导入数据时，如果处理的前n行包含格式错误，则停止处理输入行。如果在前n行中处理有效行，则Greenplum数据库继续处理输入行。

将值设置为0将禁用此限制。

还可以为COPY命令或外部表定义指定SEGMENT REJECT LIMIT子句，以限制被拒绝的行数。

INT_MAX是可以作为整数存储在系统中的最大值。

取值范围	默认值	设置分类
整数 0 - INT_MAX	1000	master session

		reload
--	--	--------

gp_instrument_shmem_size

为查询指标分配的共享内存量（以KB为单位）。默认值为5120，最大值为131072。启动时，如果[gp_enable_query_metrics](#)设置为on，则Greenplum数据库会在共享内存中分配空间以保存查询指标。该内存被组织为标题和插槽列表。所需的插槽数取决于并发查询的数量和每个查询的执行计划节点的数量。默认值5120基于Greenplum数据库系统，该系统最多执行约250个并发查询，每个查询120个节点。如果[gp_enable_query_metrics](#)配置参数关闭，或者插槽耗尽，则指标将保留在本地内存中而不是共享内存中。

取值范围	默认值	设置分类
整数0 ~ 131072	5120	master system restart

gp_interconnect_debug_retry_interval

指定在服务器配置参数[gp_log_interconnect](#)设置为DEBUG时记录Greenplum数据库interconnect调试消息的时间间隔（以秒为单位）。默认值为10秒。

日志消息包含有关Greenplum数据库segment实例工作进程之间的interconnect通信的信息。调试segment实例之间的网络问题时，这些信息可能会有所帮助。

取值范围	默认值	设置分类
1 =< 整数 < 4096	10	master session reload

gp_interconnect_fc_method

指定用于默认Greenplum数据库UDPIFC interconnect的流控制方法。

对于基于容量的流量控制，当接收器没有容量时，发送器不发送数据包。

基于损耗的流量控制基于基于容量的流量控制，并且还根据包丢失来调整发送速度。

取值范围	默认值	设置分类
CAPACITY	LOSS	master
LOSS		session reload

gp_interconnect_hash_multiplier

设置Greenplum数据库用于跟踪与默认UDPIFC interconnect的连接的哈希表的大小。此数字乘以segment数以确定哈希表中的桶数。增加该值可能会增加复杂多切片查询的interconnect性能（同时在segment主机上消耗更多内存）。

取值范围	默认值	设置分类
2-25	2	master session reload

gp_interconnect_queue_depth

设置接收器上的排队的Greenplum数据库每方用于默认UDPIFC的interconnect数据量（当接收到数据但没有可用于接收数据的空间时，数据将被丢弃，并且发送器将需要重新发送它）。将深度从其默认值增加将导致系统使用更多内存，但可能会提高性能。将此值设置在1到10之间是合理的。具有数据倾斜的查询可能会随着队列深度的增加而更好地执行。增加这可能会从根本上增加系统使用的内存量。

取值范围	默认值	设置分类
1-2048	4	master session reload

gp_interconnect_setup_timeout

指定在超时之前等待Greenplum数据库interconnect完成设置的时间。

取值范围	默认值	设置分类
任何有效的时间表达式 (数量和单位)	2 hours	master session reload

gp_interconnect_snd_queue_depth

设置发件人上默认UDPIFC interconnect排队的每方的数据量。 将深度从其默认值增加将导致系统使用更多内存，但可能会提高性能。 此参数的合理值介于1和4之间。 增加该值可能会从根本上增加系统使用的内存量。

取值范围	默认值	设置分类
1 - 4096	2	master session reload

gp_interconnect_type

设置用于Greenplum数据库interconnect流量的网络协议。 UDPIFC指定使用UDP和流量控制来实现interconnect流量，并且是唯一支持的值。

UDPIFC（默认值）指定使用UDP和流量控制来实现interconnect流量。 使用[gp_interconnect_fc_method](#) 指定interconnect流控制方法。

使用TCP作为interconnect协议，Greenplum数据库的上限为1000个segment实例 - 如果查询工作负载涉及复杂的多切片查询时，上限小于该值。

取值范围	默认值	设置分类
UDPIFC	UDPIFC	local
TCP		system restart

gp_log_format

指定服务器日志文件的格式。 如果使用[gp_toolkit](#) 管理模式，则日志文件必须为CSV格式。

取值范围	默认值	设置分类
csv	csv	local
text		system restart

gp_max_local_distributed_cache

设置要在segment实例的后端进程中缓存的最大分布式事务日志条目数。

日志条目包含有关SQL语句正在访问的行的状态的信息。该信息用于确定在MVCC环境中多个SQL语句同时执行时SQL事务可见的行。通过提高行可见性确定过程的性能，本地缓存分布式事务日志条目可提高事务处理速度。

默认值适用于各种SQL处理环境。

取值范围	默认值	设置分类
整数	1024	local system restart

gp_max_packet_size

设置Greenplum数据库interconnect的元组序列化块大小。

取值范围	默认值	设置分类
512-65536	8192	master system restart

gp_max_plan_size

指定查询执行计划的最大未压缩总大小乘以计划中的Motion运算符（切片）数。如果查询计划的大小超过该值，则取消查询并返回错误。值为0表示不监视计划的大小。

您可以指定以kB, MB或GB为单位的值。默认单位是kB。例如，值200是200kB。值1GB与1024MB或1048576kB相同。

取值范围	默认值	设置分类
整数	0	master superuser session

gp_max_slices

指定可以由查询生成的最大切片数（在segment实例上执行的查询计划的部分）。如果查询生成的切片数超过指定数量，则Greenplum数据库将返回错误并且不执行查询。默认值为0，没有最大值。

执行生成大量切片的查询可能会影响Greenplum数据库性能。例如，在多个复杂视图上包含UNION或UNION ALL运算符的查询可以生成大量切片。您可以在查询上运行EXPLAIN ANALYZE以查看查询的切片统计信息。

取值范围	默认值	设置分类
0 - INT_MAX	0	master session reload

gp_motion_cost_per_row

设置Motion运算符的Postgres查询优化器（planner）成本估算，以将行从一个segment传输到另一个segment，以连续页面提取的成本的小数部分来衡量。如果为0，则使用的值是*cpu_tuple_cost* 值的两倍。

取值范围	默认值	设置分类
浮点数	0	master session reload

gp_recursive_cte

控制SELECT [INTO]命令的WITH子句或DELETE, INSERT或UPDATE命令中RECURSIVE关键字的可用性。该关键字允许命令的WITH子句中的子查询引用自身。默认值为false, 命令的WITH子句中不允许使用RECURSIVE关键字。

有关RECURSIVE关键字 (Beta) 的信息, 请参阅[SELECT命令和WITH查询 \(公用表表达式\)](#)。

可以为数据库系统, 单个数据库或会话或查询设置该参数。

Note: 此参数之前名为gp_recursive_cte_prototype, 但已重命名以反映实现的当前状态。

取值范围	默认值	设置分类
Boolean	true	master session restart

gp_reject_percent_threshold

对于COPY和外部表SELECT上的单行错误处理, 设置在SEGMENT REJECT LIMIT n PERCENT开始计算之前处理的行数。

取值范围	默认值	设置分类
1- n	300	master session reload

gp_reraise_signal

如果启用, 将在发生致命服务器错误时尝试生成core文件。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_resgroup_memory_policy

Note: 仅当资源管理设置为资源组时, `gp_resgroup_memory_policy`服务器配置参数才会生效。

由资源组用于管理查询运算符的内存分配。

设置为auto时, Greenplum数据库使用资源组内存限制在查询运算符之间分配内存, 为非内存密集型运算符分配固定大小的内存, 其余内存为内存密集型运算符。

当您指定eager_free时, Greenplum数据库通过将已完成处理的运算符释放的内存重新分配给稍后查询阶段的运算符, 从而更好地在运算符之间分配内存。

取值范围	默认值	设置分类
auto, eager_free	eager_free	local system superuser restart/reload

gp_resource_group_bypass

Note: 仅当资源管理设置为资源组时, `gp_resource_group_bypass`服务器配置参数才会生效。

启用或禁用对Greenplum数据库资源的资源组并发事务限制的实施。 默认值为false, 这会强制执行资源组事务限制。 资源组管理资源, 例如CPU, 内存以及查询和外部组件 (如PL / Container) 使用的并发事务数。

您可以将此参数设置为true以绕过资源组并发事务限制, 以便查询可以立即运行。 例如, 您可以将参数设置为true, 以便会话运行系统目录查询或需要最少量资源的类似查询。

将此参数设置为true并运行查询时, 查询将在此环境中运行:

- 查询在资源组内运行。查询的资源组分配不会更改。
- 查询内存配额大约为每个查询10 MB。内存从资源组共享内存或全局共享内存中分配。如果没有足够的共享内存可用于满足内存分配请求, 则查询将失败。

可以为会话设置此参数。无法在事务或函数中设置该参数。

取值范围	默认值	设置分类
Boolean	false	session

gp_resource_group_cpu_limit

Note: 仅当资源管理设置为资源组时, `gp_resource_group_cpu_limit`服务器配置参数才会生效。

标识要分配给每个Greenplum数据库segment节点上的资源组的系统CPU资源的最大百分比。

取值范围	默认值	设置分类
0.1 - 1.0	0.9	local system restart

gp_resource_group_memory_limit

Note: 仅当资源管理设置为资源组时, `gp_resource_group_memory_limit`服务器配置参数才会生效。

标识要分配给每个Greenplum数据库segment节点上的资源组的系统内存资源的最大百分比。

取值范围	默认值	设置分类
0.1 - 1.0	0.7	local system restart

Note: 当基于资源组的资源管理处于活动状态时, 分配给segment主机的内存将由活动primary segment平均共享。

当segment为primary时, Greenplum数据库会将内存分配给primary segment。即使在故障转移情况下, primary segment的初始内存分配也不会更改。这可能导致segment主机使用比`gp_resource_group_memory_limit`设置允许的更多内存。

例如, 假设您的Greenplum数据库群集使用默认的`gp_resource_group_memory_limit`为0.7, 而名为seghost1的segment主机有4个primary和4个mirror。Greenplum数据库在seghost1上为每个primary segment分配整个系统内存的($0.7 / 4 = 0.175\%$)。如果发生故障转移并且seghost1上的两个mirror故障转移成为primary, 则原始4个primary中的每一个保留其0.175的存储器分配, 并且两个新的primary segment分别被分配系统内存的($0.7 / 6 = 0.116\%$)。seghost1在这种情况下的总体内存分配是

```
0.7 + (0.116 * 2) = 0.932%
```

它高于`gp_resource_group_memory_limit`设置中配置的百分比。

gp_resource_manager

标识当前在Greenplum数据库群集中启用的资源管理方案。默认方案是使用资源队列。

取值范围	默认值	设置分类
group	queue	local
queue		system
		restart

gp_resqueue_memory_policy

Note: 仅当资源管理设置为资源队列时，`gp_resqueue_memory_policy`服务器配置参数才会生效。

启用Greenplum内存管理功能。分配算法`eager_free`利用了并非所有运算符同时执行的事实（在Greenplum Database 4.2及更高版本中）。查询计划分为几个阶段，Greenplum数据库急切地释放在该阶段执行结束时分配给前一阶段的内存，然后将急切释放的内存分配给新阶段。

When set to `none`, memory management is the same as in Greenplum Database releases prior to 4.1.

When set to `auto`, query memory usage is controlled by `statement_mem` and resource queue memory limits.

取值范围	默认值	设置分类
none, auto, eager_free	eager_free	local system restart/reload

gp_resqueue_priority

Note: 仅当资源管理设置为资源队列时，`gp_resqueue_priority`服务器配置参数才会生效。

启用或禁用查询优先级。禁用此参数时，不会在查询运行时计算现有优先级设置。

取值范围	默认值	设置分类
Boolean	on	local system restart

gp_resqueue_priority_cpucores_per_segment

Note: 仅当资源管理设置为资源队列时, `gp_resqueue_priority_cpucores_per_segment`服务器配置参数才会生效。

指定每个segment实例分配的CPU单元数。例如, 如果Greenplum数据库群集具有配置有四个segment的10核心segment主机, 请将segment实例的值设置为2.5。对于master实例, 值为10。master主机通常只运行master实例, 因此主机的值应反映所有可用CPU内核的使用情况。

设置不正确可能导致CPU使用率不足或查询优先级不按设计工作。

Greenplum数据计算设备V2的默认值为segment 4, master 25。

取值范围	默认值	设置分类
0.1 - 512.0	4	local system restart

gp_resqueue_priority_sweeper_interval

Note: 仅当资源管理设置为资源队列时, `gp_resqueue_priority_sweeper_interval`服务器配置参数才会生效。

指定清理程序进程评估当前CPU使用率的时间间隔。当新语句变为活动状态时, 将评估其优先级, 并在达到下一个间隔时确定其CPU份额。

取值范围	默认值	设置分类
500 - 15000毫秒	1000	local system restart

gp_role

此服务器进程的角色。master设置为*dispatch*, segment设置为*execute*。

取值范围	默认值	设置分类
dispatch		read only
execute		
utility		

gp_safefswritesize

指定在非成熟文件系统中对追加优化表进行安全写入操作的最小尺寸。当指定大于零的字节数时，追加优化的写入器将填充数据添加到该数字，以防止由于文件系统错误导致的数据损坏。每个非成熟文件系统都具有已知的安全写入大小，在使用具有该类型文件系统的Greenplum数据库时必须在此处指定。这通常设置为文件系统的范围大小的倍数；例如，Linux ext3是4096字节，因此通常使用值32768。

取值范围	默认值	设置分类
整数	0	local system restart

gp_segment_connect_timeout

Greenplum interconnect将尝试通过网络连接到segment实例的超时时间。控制master和primary之间的网络连接超时，以及primary到mirror复制过程的网络连接超时。

取值范围	默认值	设置分类
任何有效的时间表达式（数量和单位）	10分钟	local system reload

gp_segments_for_planner

设置Postgres查询优化器（planner）在其成本和大小估计中假设的primary实例数。如果为0，则使用的值是primary的实际数量。此变量影响Postgres优化器对Motion运算符中每个发送和接收过程处理的行数的估计。

取值范围	默认值	设置分类
0-n	0	master session reload

gp_server_version

以字符串形式报告服务器的版本号。 版本修饰符参数可能会附加到版本字符串的数字部分，例如： *5.0.0 beta*。

取值范围	默认值	设置分类
字符串。示例： <i>5.0.0</i>	n/a	read only

gp_server_version_num

以整数形式报告服务器的版本号。 对于每个版本，该数字始终保持增加，并可用于数字比较。 主要版本按原样表示，次要版本和补丁版本为零填充，始终为两位数宽。

取值范围	默认值	设置分类
<i>Mmmpp</i> , 其中 <i>M</i> 是主要版本, <i>mm</i> 是零填充的次要版本, <i>pp</i> 是零填充的补丁版本。 示例： <i>50000</i>	n/a	read only

gp_session_id

系统为客户端会话分配的ID号。 首次启动master实例时从1开始计数。

取值范围	默认值	设置分类
1- <i>n</i>	14	read only

gp_set_proc_affinity

如果启用，当启动Greenplum服务器进程（postmaster）时，它将绑定到一个CPU。

取值范围	默认值	设置分类
Boolean	off	master system restart

gp_set_read_only

设置为on以禁用对数据库的写入。任何正在进行的事务必须在只读模式生效之前完成。

取值范围	默认值	设置分类
Boolean	off	master system restart

gp_statistics_pullup_from_child_partition

在Postgres查询优化器（planner）规划父表的查询时，允许使用子表中的统计信息。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_statistics_use_fkeys

启用后，允许Postgres查询优化器（planner）使用存储在系统目录中的外键信息来优化外键和主键之间的连接。

取值范围	默认值	设置分类
Boolean	off	master session reload

gp_use_legacy_hashops

对于使用DISTRIBUTED BY *key_column*子句定义的表，此参数控制用于在segment实例之间分发表数据的哈希算法。默认值为false，使用跳

转一致性哈希算法。

将值设置为true使用与Greenplum数据库5.x和早期版本兼容的取模哈希算法。

取值范围	默认值	设置分类
Boolean	false	master session reload

gp_vmem_idle_resource_timeout

如果数据库会话空闲的时间超过指定的时间，则会话将释放系统资源（例如共享内存），但仍保持与数据库的连接。这允许一次更多并发连接到数据库。

取值范围	默认值	设置分类
任何有效的时间表达式（数量和单位）	18s	master system reload

gp_vmem_protect_limit

Note: 仅当资源管理设置为资源队列时，`gp_vmem_protect_limit`服务器配置参数才会生效。

设置活动segment实例的所有postgres进程可以使用的内存量（以MB为单位）。如果查询导致超出此限制，则不会分配内存，查询将失败。请注意，这是一个本地参数，必须为系统中的每个segment（primary和mirror）设置。设置参数值时，仅指定数值。例如，要指定4096MB，请使用值4096.不要将单位MB添加到该值。

为了防止内存过度分配，这些计算可以估计安全的`gp_vmem_protect_limit`值。

首先计算值`gp_vmem`。这是主机上可用的Greenplum数据库内存

```
gp_vmem = ((SWAP + RAM) - (7.5GB + 0.05 * RAM)) / 1.7
```

其中`SWAP`是主机交换空间，`RAM`是主机上的RAM，以GB为单位。

接下来，计算`max_acting_primary_segments`。这是由于故障而激活mirror时主机上可以运行的primary的最大数量。例如，如果mirror布置在4个主机块中，每个主机有8个primary，则单个segment主机故障将激活故障主机块中每个剩余主机上的两个或三个mirror。此配置

的`max_acting_primary_segments` 值为11（8个primary加上3个失败时激活的mirror）。

这是`gp_vmem_protect_limit`的计算。该值应转换为MB。

```
gp_vmem_protect_limit = gp_vmem / acting_primary_segments
```

对于生成大量工作文件的情况，这是计算工作文件的`gp_vmem` 的计算。

```
gp_vmem = ((SWAP + RAM) - (7.5GB + 0.05 * RAM - (300KB * total_#_workfiles))) / 1.7
```

有关监视和管理工作文件使用情况的信息，请参阅Greenplum数据库管理员指南。

根据`gp_vmem` 值，您可以计算`vm.overcommit_ratio`操作系统内核参数的值。配置每个Greenplum数据库主机时，将设置此参数。

```
vm.overcommit_ratio = (RAM - (0.026 * gp_vmem)) / RAM
```

Note: Red Hat Enterprise Linux中内核参数`vm.overcommit_ratio`的默认值为50。

有关内核参数的信息，请参阅Greenplum数据库安装指南。

取值范围	默认值	设置分类
整数	8192	local system restart

gp_vmem_protect_segworker_cache_limit

如果查询执行程序进程消耗的数量超过此配置的数量，则在进程完成后，将不会高速缓存该进程以用于后续查询。具有大量连接或空闲进程的系统可能希望减少此数量以释放segment上的更多内存。请注意，这是一个本地参数，必须为每个segment设置。

取值范围	默认值	设置分类
MB	500	local system restart

gp_workfile_compression

指定在哈希聚合或哈希连接操作溢出到磁盘时创建的临时文件是否压缩。

如果您的Greenplum数据库安装使用串行ATA (SATA) 磁盘驱动器，则启用压缩可能有助于避免使用IO操作使磁盘子系统过载。

取值范围	默认值	设置分类
Boolean	off	master session reload

gp_workfile_limit_files_per_query

设置每个segment的每个查询允许的临时溢出文件（也称为工作文件）的最大数量。溢出文件会在执行需要更多内存的查询时被创建。超出限制时终止当前查询。

将值设置为0（零）以允许无限数量的溢出文件。主会话重新加载

取值范围	默认值	设置分类
整数	100000	master session reload

gp_workfile_limit_per_query

设置允许单个查询用于在每个segment创建临时溢出文件的最大磁盘大小。默认值为0，表示不强制执行限制。

取值范围	默认值	设置分类
KB	0	master session reload

gp_workfile_limit_per_segment

设置允许所有正在运行的查询用于在每个segment创建临时溢出文件的最大总磁盘大小。默认值为0，表示不强制执行限制。

取值范围	默认值	设置分类
KB	0	local system restart

gpperfmon_port

设置所有数据收集代理（用于Command Center）与master通信的端口。

取值范围	默认值	设置分类
整数	8888	master system restart

ignore_checksum_failure

仅在启用了[data checksums](#) 时才有效。

Greenplum数据库使用校验和来防止将文件系统中已损坏的数据加载到由数据库进程管理的内存中。

默认情况下，当读取堆数据页时发生校验和验证错误时，Greenplum数据库会生成错误并阻止将页面加载到托管内存中。

当ignore_checksum_failure设置为on并且校验和验证失败时，Greenplum Database会生成警告，并允许将页面读入托管内存。如果页面随后更新，则会将其保存到磁盘并复制到mirror。如果页眉已损坏，即使启用此选项，也会报告错误。

Warning: 将ignore_checksum_failure设置为on可能会传播或隐藏数据损坏或导致其他严重问题。但是，如果已检测到校验和失败且页眉未损坏，则将ignore_checksum_failure设置为on可能允许您绕过错误并恢复表中可能仍存在的未损坏元组。

默认设置为关闭，只能由超级用户更改。

取值范围	默认值	设置分类
Boolean	off	local system restart

integer_datetimes

报告PostgreSQL是否构建为支持64位整数日期和时间。

取值范围	默认值	设置分类
Boolean	on	read only

IntervalStyle

设置间隔值的显示格式。 值*sql_standard* 生成与SQL标准区间文字匹配的输出。 当*DateStyle* 参数设置为ISO时，*postgres* 值会生成与8.4之前的PostgreSQL版本匹配的输出。

当*DateStyle* 参数设置为非ISO输出时，*postgres_verbose* 值会产生与3.3之前的Greenplum版本匹配的输出。

值*iso_8601* 将生成与ISO 8601第4.4.3.2节中定义的指示符匹配时间间隔格式的输出。 有关详细信息，请参阅[PostgreSQL 8.4文档](#)。

取值范围	默认值	设置分类
postgres	postgres	master
postgres_verbose		session
sql_standard		reload
iso_8601		

join_collapse_limit

Postgres查询优化器（planner）会在总项不超过配置就显式将内部JOIN结构重写为FROM项列表。 默认情况下，此变量的设置与*fromCollapse_limit* 相同，适用于大多数用途。 将其设置为1可防止对内部JOIN进行任何重新排序。 将此变量设置为介于1和*fromCollapse_limit* 之间的值可能有助于将计划时间与所选计划的质量进行权衡（更高的值会产生更好的计划）。

取值范围	默认值	设置分类
1-n	20	master session reload

keep_wal_segments

对于Greenplum数据库主镜像，如果发生检查点操作，则设置由活动Greenplum数据库主机保存的已处理WAL段文件的最大数量。

段文件用于在standby上同步活跃的master。

取值范围	默认值	设置分类
整数	5	master system reload superuser

krb_caseins_users

设置是否应对区分大小写的Kerberos用户名进行处理。 默认值区分大小写（关闭）。

取值范围	默认值	设置分类
Boolean	off	master system reload

krb_server_keyfile

设置Kerberos服务器密钥文件的位置。

取值范围	默认值	设置分类
路径和文件名	unset	master system restart

lc_collate

报告完成文本数据排序的区域设置。 初始化Greenplum数据库阵列时确定该值。

取值范围	默认值	设置分类
<系统依赖>		read only

lc_ctype

报告确定字符分类的区域设置。 初始化Greenplum数据库阵列时确定该值。

取值范围	默认值	设置分类
<系统依赖>		read only

lc_messages

设置消息的显示语言。 可用的语言环境取决于操作系统安装的内容 - 使用*locale -a* 列出可用的语言环境。 默认值继承自服务器的执行环境。 在某些系统上，此区域设置类别不存在。 设置此变量仍然有效，但不会产生任何影响。 此外，有可能不存在所需语言的翻译消息。 在这种情况下，您将继续看到英文消息。

取值范围	默认值	设置分类
<系统依赖>		local system restart

lc_monetary

设置用于格式化货币金额的区域设置，例如使用*to_char* 系列函数。 可用的语言环境取决于操作系统安装的内容 - 使用*locale -a* 列出可用的语言环境。 默认值继承自服务器的执行环境。

取值范围	默认值	设置分类
<系统依赖>		local system restart

lc_numeric

设置用于格式化数字的语言环境，例如使用`to_char`系列函数。可用的语言环境取决于操作系统安装的内容 - 使用`locale -a`列出可用的语言环境。默认值继承自服务器的执行环境。

取值范围	默认值	设置分类
<系统依赖>		local system restart

lc_time

此参数目前不执行任何操作，但可能在将来执行。

取值范围	默认值	设置分类
<系统依赖>		local system restart

listen_addresses

指定服务器将侦听来自客户端应用程序的连接的TCP/IP地址 - 以逗号分隔的主机名和/或数字IP地址列表。特殊条目*对应于所有可用的IP接口。如果列表为空，则只能连接UNIX域套接字。

取值范围	默认值	设置分类
localhost, host names, IP addresses, * (all available IP interfaces)	*	master system restart

local_reload_libraries

以逗号分隔的共享库文件列表，以便在客户端会话开始时进行预加载。

取值范围	默认值	设置分类
		local
		system
		restart

lock_timeout

尝试获取表，索引，行或其他数据库对象的锁时，中止任何等待时间超过指定毫秒数的语句。时间限制在每次锁定获取尝试是独立的。该限制既适用于显式锁定请求（例如LOCK TABLE，也适用于没有NOWAIT的SELECT FOR UPDATE），也适用于隐式获取的锁定。如果log_min_error_statement设置为ERROR或更低，则Greenplum数据库会记录超时的语句。值为零（默认值）将关闭此锁定等待监视。

与statement_timeout不同，此超时只能在等待锁定时发生。请注意，如果statement_timeout非零，则将lock_timeout设置为相同或更大的值是没有意义的，因为语句超时始终会先触发。

Greenplum数据库使用deadlock_timeout和gp_global_deadlock_detector_period来触发本地和全局死锁检测。请注意，如果打开lock_timeout并将其设置为小于这些死锁检测超时的值，则Greenplum数据库将在该会话中触发死锁检查之前中止语句。

Note: 建议不要在postgresql.conf中设置lock_timeout，因为它会影响所有会话

取值范围	默认值	设置分类
0 - INT_MAX毫秒	0毫秒	master session reload

log_autostats

记录有关与gp_autostats_mode 和gp_autostats_on_change_threshold 相关的自动ANALYZE操作的信息。

取值范围	默认值	设置分类
Boolean	off	master

		session
		reload
		superuser

log_connections

这将向服务器日志输出一行，详细说明每个成功的连接。某些客户端程序（如psql）在确定是否需要密码时尝试连接两次，因此重复的“已接收连接”消息并不总是表示存在问题。

取值范围	默认值	设置分类
Boolean	off	local system restart

log_disconnections

这在客户端会话终止时在服务器日志中输出一行，并包括会话的持续时间。

取值范围	默认值	设置分类
Boolean	off	local system restart

log_dispatch_stats

设置为“on”时，此参数将添加一条日志消息，其中包含有关语句分派的详细信息。

取值范围	默认值	设置分类
Boolean	off	local system

		restart
--	--	---------

log_duration

导致记录满足*log_statement* 的每个已完成语句的持续时间。

取值范围	默认值	设置分类
Boolean	off	master session reload superuser

log_error_verbosity

控制记录的每条消息在服务器日志中写入的详细信息量。

取值范围	默认值	设置分类
TERSE	DEFAULT	master
DEFAULT	DEFAULT	session
VERBOSE	DEFAULT	reload superuser

log_executor_stats

对于每个查询，将查询执行程序的性能统计信息写入服务器日志。这是一种原始的分析工具。无法与*log_statement_stats* 一起启用。

取值范围	默认值	设置分类
Boolean	off	local system restart

log_hostname

默认情况下，连接日志消息仅显示连接主机的IP地址。 打开此选项会导致记录Greenplum数据库主服务器的IP地址和主机名。 请注意，根据您的主机名解析设置，这可能会造成不可忽视的性能损失。

取值范围	默认值	设置分类
Boolean	off	master system restart

log_min_duration_statement

如果语句的持续时间大于或等于指定的毫秒数，则将语句及其持续时间记录在单个日志行上。 将此值设置为0将打印所有语句及其持续时间。 -1禁用该功能。 例如，如果将其设置为250，则将记录运行250毫秒或更长时间的所有SQL语句。 启用此选项可用于跟踪应用程序中未优化的查询。

取值范围	默认值	设置分类
毫秒数, 0, -1	-1	master session reload superuser

log_min_error_statement

控制是否还将在服务器日志中记录导致错误情况的SQL语句。 将记录导致指定级别或更高级别的错误的所有SQL语句。 默认为PANIC（有效关闭此功能以供正常使用）。 启用此选项有助于跟踪服务器日志中出现的任何错误的来源。

取值范围	默认值	设置分类
DEBUG5	ERROR	master
DEBUG4		session
DEBUG3		reload

DEBUG2			superuser
DEBUG1			
INFO			
NOTICE			
WARNING			
ERROR			
FATAL			
PANIC			

log_min_messages

控制将哪些消息级别写入服务器日志。 每个级别包括其后的所有级别。 级别越靠后，发送到日志的消息越少。

如果安装了Greenplum数据库PL/Container扩展。 此参数还控制PL/Container日志级别。 有关扩展的信息，请参阅[Greenplum PL/Container语言扩展语言扩展](#)。

取值范围	默认值	设置分类
DEBUG5	WARNING	master
DEBUG4		session
DEBUG3		reload
DEBUG2		superuser
DEBUG1		
INFO		
NOTICE		
WARNING		
LOG		
ERROR		
FATAL		

PANIC

log_parser_stats

对于每个查询，将查询解析器的性能统计信息写入服务器日志。这是一种原始的分析工具。无法与*log_statement_stats* 一起启用。

取值范围	默认值	设置分类
Boolean	off	master session reload superuser

log_planner_stats

对于每个查询，将Postgres查询优化器（planner）的性能统计信息写入服务器日志。这是一种原始的分析工具。无法与*log_statement_stats* 一起启用。

取值范围	默认值	设置分类
Boolean	off	master session reload superuser

log_rotation_age

确定Greenplum数据库将消息写入活动日志文件的时间。 经过这段时间后，文件将关闭并创建一个新的日志文件。 设置为零以禁用基于时间的新日志文件创建。

取值范围	默认值	设置分类
任何有效的时间表达式（数量和单位）	1d	local system

		restart
--	--	---------

log_rotation_size

确定触发旋转的单个日志文件的大小。当日志文件大小等于或大于此大小时，将关闭该文件并创建新的日志文件。设置为零以禁用基于大小的新日志文件的创建。

最大值为INT_MAX/1024。如果指定了无效值，则使用默认值。INT_MAX是可以作为整数存储在系统中的最大值。

取值范围	默认值	设置分类
kb	1048576	local system restart

log_statement

控制记录哪些SQL语句。DDL记录所有数据定义命令，如CREATE, ALTER和DROP命令。MOD记录所有DDL语句，以及INSERT, UPDATE, DELETE, TRUNCATE和COPY FROM。如果包含的命令属于适当的类型，也会记录PREPARE和EXPLAIN ANALYZE语句。

取值范围	默认值	设置分类
NONE	ALL	master
DDL		session
MOD		reload
ALL		superuser

log_statement_stats

对于每个查询，将查询解析器，优化器和执行器的总体性能统计信息写入服务器日志。这是一种原始的分析工具。

取值范围	默认值	设置分类
Boolean	off	master

		session
		reload
		superuser

log_temp_files

控制临时文件名和大小的记录。可以为排序，哈希，临时查询结果和溢出文件创建临时文件。删除时，每个临时文件都会在pg_log中创建一个日志条目。根据临时文件的来源，可以在master和/或segment上创建日志条目。log_temp_files值为零会记录所有临时文件信息，而正值仅记录大小大于或等于指定KB数的文件。默认设置为-1，禁用日志记录。只有超级用户才能更改此设置。

取值范围	默认值	设置分类
整数	-1	local system restart

log_timezone

设置用于写入日志的时间戳的时区。与[TimeZone](#)不同，此值是系统范围的，因此所有会话都将一致地报告时间戳。默认值是unknown，这意味着使用系统环境指定的任何时区。

取值范围	默认值	设置分类
字符串	unknown	local system restart

log_truncate_on_rotation

截断（覆盖）而不是附加到任何现有的同名日志文件。仅当由于基于时间的旋转而打开新文件时才会发生截断。例如，将此设置与log_filename（例如gpseg#-%H.log）结合使用将导致生成二十四小时的日志文件，然后循环覆盖它们。禁用时，将在所有情况下追加预先存在的文件。

取值范围	默认值	设置分类
Boolean	off	local system restart

maintenance_work_mem

指定要在维护操作中使用的最大内存量，例如VACUUM和CREATE INDEX。默认为16兆字节（16MB）。较大的设置可能会提高清理和恢复数据库转储的性能。

取值范围	默认值	设置分类
整数	16	local system restart

max_appendonly_tables

设置可以写入或更新追加优化表的最大并发事务数。超过最大值的事务将返回错误。

计算的操作是INSERT, UPDATE, COPY和VACUUM操作。限制仅适用于正在进行的交易。一旦交易结束（中止或提交），就不再计入此限制。

对于针对分区表的操作，作为追加优化表并且被更改的每个子分区（子表）计为最大的单个表。例如，分区表p_tbl1被定义为具有三个子分区，这些子分区是追加优化表p_tbl1_ao1, p_tbl1_ao2和p_tbl1_ao3。针对分区表p_tbl1的INSERT或UPDATE命令将更改追加优化表p_tbl1_ao1和p_tbl1_ao2计为两个事务。

增加限制会在服务器启动时在master主机上分配更多共享内存。

取值范围	默认值	设置分类
整数 > 0	10000	master system restart

max_connections

与数据库服务器的最大并发连接数。在Greenplum数据库系统中，用户客户端连接仅通过Greenplum master实例。segment实例应该允许相比master实例5-10倍的数量。增加此参数时，还必须增加[max_prepared_transactions](#)。有关限制并发连接的详细信息，请参阅[Greenplum数据库管理员指南](#)中的“配置客户端身份验证”。

增加此参数可能会导致Greenplum数据库请求更多共享内存。有关Greenplum服务器实例共享内存缓冲区的信息，请参阅[shared_buffers](#)。

取值范围	默认值	设置分类
10- <i>n</i>	250 on master	local
	750 on segments	system
		restart

max_files_per_process

设置允许每个服务器子进程的最大同时打开文件数。如果内核强制执行每个进程的安全限制，则无需担心此设置。一些平台，如BSD，内核将允许单个进程打开比系统真正支持的更多文件。

取值范围	默认值	设置分类
整数	1000	local
		system
		restart

max_function_args

报告函数参数的最大数量。

取值范围	默认值	设置分类
整数	100	read only

max_identifier_length

报告最大标识符长度。

取值范围	默认值	设置分类
整数	63	read only

max_index_keys

报告索引键的最大数量。

取值范围	默认值	设置分类
整数	32	read only

max_locks_per_transaction

创建共享锁定表时，可以在 $max_locks_per_transaction * (max_connections + max_prepared_transactions)$ 对象上描述锁定，因此在任何时候都不能锁定这么多不同的对象。这不是任何一个事务所采用的锁数的硬限制，而是最大平均值。如果您的客户端在单个事务中触及许多不同的表，则可能需要提高此值。

取值范围	默认值	设置分类
整数	128	local system restart

max_prepared_transactions

设置可以同时处于准备状态的最大事务数。Greenplum在内部使用准备好的事务来确保各个segment的数据完整性。该值必须至少与master上的`max_connections` 值一样大。segment实例应设置为与master相同的值。

取值范围	默认值	设置分类
整数	250 on master 250 on segments	local system restart

max_resource_portals_per_transaction

Note: 仅当资源管理设置为资源队列时，`max_resource_portals_per_transaction`服务器配置参数才会生效。

设置每个事务允许的同时打开的用户声明游标的最大数量。请注意，打开的游标将在资源队列中保留活动的查询槽。用于资源管理。

取值范围	默认值	设置分类
整数	64	master system restart

max_resource_queues

Note: 仅当资源管理设置为资源队列时，`max_resource_portals_per_transaction`服务器配置参数才会生效。

设置可以在Greenplum数据库系统中创建的最大资源队列数。请注意，资源队列是系统范围的（角色也是如此），因此它们适用于系统中的所有数据库。

取值范围	默认值	设置分类
整数	9	master system restart

max_stack_depth

指定服务器执行堆栈的最大安全深度。此参数的理想设置是内核强制实际的堆栈大小限制（由

取值范围	默认值	设置分类
KB	2MB	local system restart

max_statement_mem

Note: 仅当资源管理设置为资源队列时, `max_statement_mem`服务器配置参数才会生效。

设置查询的最大内存限制。由于将`statement_mem`设置得太高,有助于在查询处理期间避免segment主机上的内存不足错误。

当`gp_resqueue_memory_policy=auto`时, `statement_mem`和资源队列内存限制控制查询内存使用量。考虑到单个segment主机的配置,请按如下方式计算此设置:

$$(\text{segghost_physical_memory}) / (\text{average_number_concurrent_queries})$$

更改`max_statement_mem`和`statement_mem`时, 必须先更改`max_statement_mem`, 或者先在`postgresql.conf`文件中列出。

取值范围	默认值	设置分类
KB	2000MB	master session reload superuser

memory_spill_ratio

Note: 仅当资源管理设置为资源组时, `memory_spill_ratio`服务器配置参数才会生效。

设置事务中内存密集型运算符的内存使用阈值百分比。当事务达到此阈值时, 它将溢出到磁盘。

默认`memory_spill_ratio`百分比是为分配给当前活动角色的资源组定义的值。您可以在会话级别设置`memory_spill_ratio`,以便在每个查询的基础上有选择地设置此限制。例如,如果您有一个特定的查询溢出到磁盘并需要更多内存,您可以选择设置更大的`memory_spill_ratio`来增加初始内存分配。

在会话级别设置`memory_spill_ratio`时, Greenplum数据库不会对新值执行语义验证,直到您下次执行查询为止。

取值范围	默认值	设置分类
0 - 100	20	master session reload

optimizer

运行SQL查询时启用或禁用GPORCA。 默认on。 如果禁用GPORCA， Greenplum数据库仅使用Postgres查询优化器。

GPORCA与Postgres查询优化器共存。 启用GPORCA后， Greenplum数据库会尽可能使用GPORCA为查询生成执行计划。 如果无法使用GPORCA，则使用Postgres查询优化器。

可以为数据库系统，单个数据库或会话或查询设置optimizer参数。

有关Postgres查询优化器和GPORCA的信息，请参阅[Querying Data](#)。

取值范围	默认值	设置分类
Boolean	on	master session reload

optimizer_analyze_root_partition

对于分区表，控制在表上运行ANALYZE命令时是否需要ROOTPARTITION关键字来收集根分区统计信息。 GPORCA在生成查询计划时使用根分区统计信息。 Postgres查询优化器不使用这些统计信息。

参数的默认设置为on， ANALYZE命令可以在不使用ROOTPARTITION关键字的情况下收集根分区统计信息。 在根分区上运行ANALYZE时，或者在分区表的子叶子分区上运行ANALYZE并且其他子叶子分区具有统计信息时，将收集根分区统计信息。 当该值为off时，您必须运行ANALYZE ROOTPARTITION以收集根分区统计信息。

当服务器配置参数optimizer的值为on（默认值）时，此参数的值也应该为on。 有关收集分区表的表统计信息的信息，请参阅[ANALYZE](#)。

有关Postgres查询优化器和GPORCA的信息，请参阅[查询数据](#)。

取值范围	默认值	设置分类
Boolean	on	master session reload

optimizer_array_expansion_threshold

启用GPORCA（默认值）并处理包含具有常量数组的谓词的查询时， optimizer_array_expansion_threshold参数会根据数组中的常量数限制优化过程。 如果查询谓词中的数组包含多于参数指定的数字元素，则GPORCA会在查询优化期间禁用谓词转换为其析取范式。

默认值是100。

例如，当GPORCA执行包含具有100个以上元素的IN子句的查询时，GPORCA在查询优化期间不会将谓词转换为其析取范式，以减少优化时间消耗更少的内存。查询处理的差异可以在查询EXPLAIN计划的IN子句的过滤条件中看到。

更改此参数的值会更改较短的优化时间和较低的内存消耗之间的权衡，以及查询优化期间约束派生的潜在好处，例如冲突检测和分区消除。

可以为数据库系统，单个数据库或会话或查询设置该参数。

取值范围	默认值	设置分类
整数 > 0	25	master session reload

optimizer_control

控制是否可以使用SET，RESET命令或Greenplum数据库实用程序gpconfig更改服务器配置参数optimizer。如果启用了optimizer_control参数值，则用户可以设置优化程序参数。如果optimizer_control参数值为off，则无法更改优化程序参数。

取值范围	默认值	设置分类
Boolean	on	master system restart superuser

optimizer_cte_inlining_bound

启用GPORCA（默认值）时，此参数控制为公用表表达式（CTE）查询（包含WHERE子句的查询）执行的内联量。默认值0禁用内联。

可以为数据库系统，单个数据库或会话或查询设置该参数。

取值范围	默认值	设置分类
Decimal >= 0	0	master session

reload

optimizer_enable_associativity

启用GPORCA（默认设置）时，此参数控制是否在查询优化期间启用连接关联变换。转换分析连接顺序。对于默认值off，仅启用用于分析连接顺序的GPORCA动态编程算法。连接关联变换很大程度上复制了较新的动态编程算法的功能。

如果值为on，GPORCA可以在查询优化期间使用关联变换。

可以为数据库系统，单个数据库或会话或查询设置该参数。

有关GPORCA的信息，请参阅[关于GPORCA](#)。

取值范围	默认值	设置分类
Boolean	off	master session reload

optimizer_enable_master_only_queries

启用GPORCA（默认值）时，此参数允许GPORCA执行仅在Greenplum数据库master上运行的catalog查询。对于默认值off，只有Postgres查询优化器可以执行仅在Greenplum数据库master上运行的catalog查询。

可以为数据库系统，单个数据库或会话或查询设置该参数。

Note: 启用此参数会降低短期运行的catalog查询的性能。要避免此问题，请仅为会话或查询设置此参数。

有关GPORCA的信息，请参阅[关于GPORCA](#)。

取值范围	默认值	设置分类
Boolean	off	master session reload

optimizer_force_agg_skew_avoidance

启用GPORCA（默认值）时，此参数会影响GPORCA在生成3阶段聚合计划时考虑的查询计划备选方案。当值为`true`时，默认情况下，GPORCA仅考虑3阶段聚合计划，其中中间聚合使用`GROUP BY`和`DISTINCT`列进行分发，以减少处理偏斜的影响。

如果值为`false`，GPORCA还可以考虑使用`GROUP BY`列进行分发的计划。当处理偏斜存在时，这些计划可能表现不佳。

有关GPORCA的信息，请参阅[关于GPORCA](#)。

取值范围	默认值	设置分类
Boolean	true	master session reload

optimizer_force_multistage_agg

对于默认设置，启用GPORCA且此参数为`true`，GPORCA在生成此类计划备选项时为标量不同的合格聚合选择3阶段聚合计划。当值为`false`时，GPORCA会根据成本选择而不是启发式选择。

可以为数据库系统，单个数据库或会话或查询设置该参数。

取值范围	默认值	设置分类
Boolean	true	master session reload

optimizer_force_three_stage_scalar_dqa

对于默认设置，GPORCA已启用且此参数为`true`，GPORCA会在生成此类计划备选项时选择具有多级聚合的计划。当值为`false`时，GPORCA会根据成本选择而不是启发式选择。

可以为数据库系统，单个数据库或会话或查询设置该参数。

取值范围	默认值	设置分类
Boolean	true	master session

reload

optimizer_join_arity_for_associativity_commutativity

该值是一个优化提示，用于限制在查询优化期间探索的连接关联性和连接可交换性转换的数量。该限制控制GPORCA在查询优化期间考虑的备选计划。例如，默认值18是GPORCA在优化期间n-ary连接运算符具有超过18个子节点时停止探索连接关联性和连接交换变换的优化提示。

对于具有大量连接的查询，指定较低的值可通过限制GPORCA评估的备用查询计划的数量来提高查询性能。但是，将值设置得太低可能会导致GPORCA生成次优执行的查询计划。

当optimizer_join_order参数设置为query或greedy时，此参数无效。

可以为数据库系统或会话设置此参数。

取值范围	默认值	设置分类
整数 > 0	18	local system reload

optimizer_join_order

启用GPORCA时，此参数通过指定要评估的连接排序备选类型来设置查询优化期间的连接排序的优化级别。

- query - 使用查询中指定的连接顺序。
- greedy - 根据连接中关系的最小基数，计算查询中指定的连接顺序和备选方案。
- exhaustive - 应用转换规则以查找和评估所有连接排序备选方案。

默认值是exhaustive。将此参数设置为query或greedy可以生成次优查询计划。但是，如果管理员确信使用query或greedy设置生成了令人满意的计划，则可以通过将参数设置为较低的优化级别来提高查询优化时间。

将此参数设置为query或greedy会覆

盖optimizer_join_order_threshold和optimizer_join_arity_for_associativity_commutativity参数。

可以为单个数据库，会话或查询设置此参数。

取值范围	默认值	设置分类
query	exhaustive	master

greedy		session
exhaustive		reload

optimizer_join_order_threshold

启用GPORCA（默认设置）时，此参数设置GPORCA将使用基于动态编程的连接排序算法的最大连接子数。您可以为单个查询或整个会话设置此值。

当optimizer_join_query参数设置为query或greedy时，此参数无效。

取值范围	默认值	设置分类
0 - 12	10	master session reload

optimizer_mdcache_size

设置Greenplum数据库master上GPORCA在查询优化期间用于缓存查询元数据（优化数据）的最大内存量，。基于内存限制会话。GPORCA使用默认设置在查询优化期间缓存查询元数据：启用GPORCA并启用[optimizer_metadata_caching](#)。

默认值为16384（16MB）。这是通过性能分析确定的最佳值。

您可以指定KB，MB或GB的值。默认单位是KB。例如，值16384是16384KB。值1GB与1024MB或1048576KB相同。如果值为0，则不限制高速缓存的大小。

可以为数据库系统，单个数据库或会话或查询设置此参数。

取值范围	默认值	设置分类
整数 >= 0	16384	master session reload

optimizer_metadata_caching

启用GPORCA（默认值）时，此参数指定GPORCA在查询优化期间是否在Greenplum数据库master上的内存中缓存查询元数据（优化数据）。此参数的默认值为on，启用缓存。缓存是基于会话的。会话结束时，缓存将被释放。如果查询元数据的数量超过缓存大小，则从缓存中逐出旧的未使用的元数据。

如果该值为off，则GPORCA在查询优化期间不会缓存元数据。

可以为数据库系统、单个数据库或会话或查询设置此参数。

服务器配置参数[optimizer_mdcache_size](#)控制查询元数据缓存的大小。

取值范围	默认值	设置分类
Boolean	on	master session reload

optimizer_minidump

GPORCA生成minidump文件以描述给定查询的优化上下文。文件中的信息格式不易于调试或故障排除。minidump文件位于主数据目录下，并使用以下命名格式：

`Minidump_date_time.mdp`

minidump文件包含此查询相关信息：

- catalog对象，包括GPORCA所需的数据类型，表，运算符和统计信息
- 查询的内部表示（DXL）
- GPORCA制定的计划的内部代表（DXL）
- 传递给GPORCA的系统配置信息，例如服务器配置参数，成本和统计信息配置以及segment数
- 优化查询时生成的堆栈错误跟踪

将此参数设置为ALWAYS会为所有查询生成一个minidump。将此参数设置为ONERROR以最小化总优化时间。

有关GPORCA的信息，请参阅[关于GPORCA](#)。

取值范围	默认值	设置分类
ONERROR	ONERROR	master session reload
ALWAYS		

optimizer_nestloop_factor

此参数向GPORCA添加成本因子，以在查询优化期间优先考虑哈希连接而不是嵌套循环连接。在使用均匀分布的数据评估大量工作负载之后，选择了默认值1024。应将1024视为此参数的实际上限设置。如果您发现GPORCA选择哈希连接的频率超过预期，则减小该值以转换成本因子，转而使用嵌套循环连接。

The parameter can be set for a database system, an individual database, or a session or query.

取值范围	默认值	设置分类
INT_MAX > 1	1024	master session reload

optimizer_parallel_union

启用GPORCA（默认值）时，optimizer_parallel_union控制包含UNION或UNION ALL子句的查询的并行化数量。

当值为off时，默认GPORCA会生成一个查询计划，其中APPEND（UNION）运算符的每个子项与APPEND运算符位于同一个切片中。在查询执行期间，子节点以顺序方式执行。

当该值打开时，GPORCA会生成一个查询计划，其中重新分配运动节点位于APPEND（UNION）运算符下。在查询执行期间，子项和父APPEND运算符位于不同的切片上，允许APPEND（UNION）运算符的子项在segment实例上并行执行。

可以为数据库系统，单个数据库或会话或查询设置该参数。

取值范围	默认值	设置分类
boolean	off	master session reload

optimizer_print_missing_stats

启用GPORCA（默认设置）时，此参数控制有关查询缺少统计信息的列的表列信息的显示。默认值为true，将列信息显示给客户端。当值为false时，不会将信息发送到客户端。

在查询执行期间或使用EXPLAIN或EXPLAIN ANALYZE命令显示信息。

可以为数据库系统，单个数据库或会话设置该参数。

取值范围	默认值	设置分类
Boolean	true	master session reload

optimizer_print_optimization_stats

启用GPORCA（默认设置）时，此参数可以为查询的各种优化阶段记录GPORCA查询优化统计信息。默认值为off，请勿记录优化统计信息。要记录优化统计信息，必须将此参数设置为on，并且必须将参数client_min_messages设置为log。

- set optimizer_print_optimization_stats = on;
- set client_min_messages = 'log';

在查询执行期间或使用EXPLAIN或EXPLAIN ANALYZE命令记录信息。

可以为数据库系统，单个数据库或会话或查询设置此参数。

取值范围	默认值	设置分类
Boolean	off	master session reload

optimizer_sort_factor

启用GPORCA（默认值）时，optimizer_sort_factor控制在查询优化期间应用于排序操作的成本因子。默认值1指定默认排序成本因子。该值是默认因子的增加或减少的比率。例如，值2.0将成本因子设置为默认值的两倍，值0.5将因子设置为默认值的一半。

可以为数据库系统，单个数据库或会话或查询设置该参数。

取值范围	默认值	设置分类
Decimal > 0	1	master session

reload

password_encryption

如果在CREATE USER 或ALTER USER 中指定了密码而未写入ENCRYPTED 或UNENCRYPTED , 则此选项确定是否要加密密码。

取值范围	默认值	设置分类
Boolean	on	master session reload

password_hash_algorithm

指定存储加密的Greenplum数据库用户密码时使用的加密哈希算法。 默认算法是MD5。

有关设置密码哈希算法以保护用户密码的信息, 请参阅*Greenplum数据库管理员指南*中的“保护Greenplum数据库中的密码”。

取值范围	默认值	设置分类
MD5	MD5	master
SHA-256		session reload superuser

pljava_classpath

包含jar文件或目录的冒号 (:)分隔列表, 其中包含PL/Java函数所需的jar文件。 必须指定jar文件或目录的完整路径, 但\$GPHOME/lib/postgresql/java目录中的jar文件的路径可以省略。 jar文件必须安装在所有Greenplum主机上的相同位置, 并且可由gpadmin用户读取。

pljava_classpath参数用于在每个用户会话开始时组装PL/Java类路径。 会话启动后添加的Jar文件不可用于该会话。

如果在pljava_classpath中指定了jar文件的完整路径, 则会将其添加到PL/Java类路径中。 指定目录时, 该目录包含的任何jar文件都将添加到PL/Java类路径中。 搜索不会下降到指定目录的子目录中。 如果jar文件的名称包含在没有路径的pljava_classpath中, 则jar文件必须位

于\$GPHOME/lib/postgresql/java目录中。

Note: 如果要搜索的目录很多或者有大量的jar文件，性能会受到影响。

如果`pljava_classpath_insecure`为false，则设置`pljava_classpath`参数需要超级用户权限。当没有超级用户权限的用户执行代码时，在SQL代码中设置类路径将失败。必须先由超级用户或`postgresql.conf`文件设置`pljava_classpath`参数。更改`postgresql.conf`文件中的类路径需要重新加载（`gpstop -u`）。

取值范围	默认值	设置分类
string		master session reload superuser

pljava_classpath_insecure

控制是否可以由没有Greenplum数据库超级用户权限的用户设置服务器配置参数`pljava_classpath`。如果为true，则`pljava_classpath`可由常规用户设置。否则，`pljava_classpath`只能由数据库超级用户设置。默认值为false。

Warning: 启用此参数会使非管理员数据库用户能够运行未经授权的Java方法，从而暴露出安全风险。

取值范围	默认值	设置分类
Boolean	false	master session restart superuser

pljava_statement_cache_size

为预准备语句设置JRE MRU（最近使用）缓存的大小（以KB为单位）。

取值范围	默认值	设置分类
KB	10	master system

		restart superuser
--	--	----------------------

pljava_release_lingering_savepoints

如果为true，则PL/Java函数中使用的延迟保存点将在函数退出时释放。如果为false，则将回滚保存点。

取值范围	默认值	设置分类
Boolean	true	master system restart superuser

pljava_vmoptions

定义Java VM的启动选项。默认值为空字符串（“”）。

取值范围	默认值	设置分类
string		master system reload superuser

port

Greenplum实例的数据库侦听器端口。master和每个segment都有自己的端口。还必须在gp_segment_configuration中更改Greenplum系统的端口号。您必须在更改端口号之前关闭Greenplum数据库系统。

取值范围	默认值	设置分类
任何有效的端口号	5432	local

		system restart
--	--	-------------------

random_page_cost

设置Postgres查询优化器 (planner) 的非顺序读取磁盘页面的成本估计。这是连续页面提取的成本的倍数。较高的值使得更有可能使用顺序扫描，较低的值使得更有可能使用索引扫描。

取值范围	默认值	设置分类
浮点数	100	master session reload

readable_external_table_timeout

当SQL查询从外部表读取时，参数值指定当数据停止从外部表返回时，Greenplum数据库在取消查询之前等待的时间（以秒为单位）。

默认值0，指定没有超时。Greenplum数据库不会取消查询。

如果使用gpfdist的查询运行很长时间然后返回错误“间歇性网络连接问题”，则可以为readable_external_table_timeout设定值。如果gpfdist在指定的时间内没有返回任何数据，Greenplum数据库将取消该查询。

取值范围	默认值	设置分类
整数 >= 0	0	master system reload

repl_catchup_within_range

对于Greenplum数据库主镜像，控制活跃从节点的更新。如果walsender尚未处理的WAL段文件数超过此值，Greenplum数据库将更新活跃从节点。

如果段文件的数量不超过该值，则Greenplum数据库会阻止更新，以允许walsender处理文件。如果已处理所有WAL段，则更新活动主站。If the

number of segment files does not exceed the value, Greenplum Database blocks updates to the to allow the `walsender` process the files. If all WAL segments have been processed, the active master is updated.

取值范围	默认值	设置分类
0 - 64	1	master system reload superuser

replication_timeout

对于Greenplum数据库主镜像，设置活动主服务器上的`walsender`进程等待来自备用主服务器上的`walreceiver`进程的状态消息的最长时间（以毫秒为单位）。如果未收到消息，则`walsender`会记录一条错误消息。

The `wal_receiver_status_interval` controls the interval between `walreceiver` status messages.

取值范围	默认值	设置分类
0 - INT_MAX	60000 ms (60 seconds)	master system reload superuser

regex_flavor

“扩展”设置可能对于与PostgreSQL 7.4之前版本的精确向后兼容非常有用。

取值范围	默认值	设置分类
advanced	advanced	master
extended		session
basic		reload

resource_cleanup_gangs_on_wait

Note: 仅当资源管理设置为资源队列时, `resource_cleanup_gangs_on_wait`服务器配置参数才会生效。

如果通过资源队列提交语句, 请在锁定资源队列之前清除所有空闲的查询执行程序工作进程。

取值范围	默认值	设置分类
Boolean	on	master system restart

resource_select_only

Note: 仅当资源管理设置为资源队列时, `resource_select_only`服务器配置参数才会生效。

设置资源队列管理的查询类型。如果设置为on, 则评估SELECT, SELECT INTO, CREATE TABLE AS SELECT和DECLARE CURSOR命令。如果设置为off, 则还将评估INSERT, UPDATE和DELETE命令。

取值范围	默认值	设置分类
Boolean	off	master system restart

runaway_detector_activation_percent

Note: 仅当资源管理设置为资源队列时, `runaway_detector_activation_percent`服务器配置参数才会生效。

设置触发查询终止的Greenplum数据库vmem内存的百分比。如果用于Greenplum数据库segment的vmem内存百分比超过指定值, Greenplum数据库将根据内存使用情况终止查询, 从占用最大内存量的查询开始。查询将终止, 直到使用的vmem百分比低于指定的百分比。

使用服务器配置参数[gp_vmem_protect_limit](#) 指定活动Greenplum数据库segment实例的最大vmem值。

例如, 如果vmem内存设置为10GB, 并且`runaway_detector_activation_percent`的值为90 (90%), 则当使用的vmem内存超过9GB时, Greenplum数据库将开始终止查询。

值0禁用基于所使用的vmem的百分比自动终止查询。

取值范围	默认值	设置分类
百分比(整数)	90	local system restart

search_path

指定在没有schema组件的简单名称引用对象时搜索schema的顺序。当在不同schema中存在具有相同名称的对象时，将使用在搜索路径中首先找到的对象。始终搜索系统catalog schema `pg_catalog`，无论路径中是否提及它。在未指定特定目标schema的情况下创建对象时，它们将被放置在搜索路径中列出的第一个schema中。可以通过SQL函数`current_schemas()` 检查搜索路径的当前有效值。`current_schemas()` 显示了`search_path` 中出现的请求是如何解析的。

取值范围	默认值	设置分类
以逗号分隔的schema名称列表	\$user,public	master session reload

seq_page_cost

对于Postgres查询优化器 (planner)，设置作为一系列连续提取的一部分的磁盘页面提取的成本估计。

取值范围	默认值	设置分类
浮点数	1	master session reload

server_encoding

报告数据库编码（字符集）。确定何时初始化Greenplum数据库阵列。通常，客户只需要关注`client_encoding` 的值。

取值范围	默认值	设置分类
< >	UTF8	read only

server_version

报告此版本的Greenplum数据库所基于的PostgreSQL版本。

取值范围	默认值	设置分类
字符串	9.4.20	read only

server_version_num

报告此版本的Greenplum数据库基于整数的PostgreSQL版本。

取值范围	默认值	设置分类
整数	90420	read only

shared_buffers

设置Greenplum数据库segment实例用于共享内存缓冲区的内存量。此设置必须至少为128KB且至少为[max_connections](#)的16KB倍。

每个Greenplum数据库segment实例根据segment配置计算并尝试分配一定量的共享内存。`shared_buffers`的值是此共享内存计算的重要部分，但并非全部。设置`shared_buffers`时，可能还需要调整操作系统参数SHMMAX或SHMALL的值。

操作系统参数SHMMAX指定单个共享内存分配的最大大小。SHMMAX的值必须大于此值：

```
shared_buffers + other_seg_shmem
```

`other_seg_shmem` 的值是Greenplum数据库共享内存计算的部分，而`shared_buffers`值没有考虑该部分。`other_seg_shmem` 值将根据segment配置而有所不同。

使用默认的Greenplum数据库参数值，对于Greenplum数据库segment，`other_seg_shmem` 的值大约为111MB，对于Greenplum数据库master，大约为79MB。

操作系统参数SHMALL指定主机上的最大共享内存量。SHMALL的值必须大于此值：

```
(num_instances_per_host * ( shared_buffers + other_seg_shmem )) + other_app_shared_mem
```

`other_app_shared_mem` 的值是主机上其他应用程序和进程使用的共享内存量。

发生共享内存分配错误时，解决共享内存分配问题的可能方法是增加SHMMAX或SHMALL，或减少`shared_buffers`或`max_connections`。

有关参数SHMMAX和SHMALL的Greenplum数据库值的信息，请参阅[Greenplum数据库安装指南](#)。

取值范围	默认值	设置分类
整数 > 16K * <i>max_connections</i>	125MB	local system restart

shared_preload_libraries

以逗号分隔的共享库列表，这些共享库将在服务器启动时预加载。PostgreSQL过程语言库可以通过这种方式预加载，通常使用语法\$libdir/plXXX'，其中XXX是pgsql, perl, tcl或python。通过预加载共享库，首次使用库时可以避免库启动时间。如果找不到指定的库，则服务器将无法启动。

取值范围	默认值	设置分类
		local system restart

ssl

启用SSL连接。

取值范围	默认值	设置分类
Boolean	off	master system restart

ssl_ciphers

指定允许在安全连接上使用的SSL密码列表。有关支持的密码列表，请参见[openssl手册页](#)。

取值范围	默认值	设置分类
字符串	ALL	master system restart

standard_conforming_strings

确定普通的字符串文字 ('...') 是否按字面意思处理反斜杠，如SQL标准中所指定的那样。默认值为on。关闭此参数可将字符串文字中的反斜杠视为转义字符而不是文字反斜杠。应用程序可以检查此参数以确定如何处理字符串文字。此参数的存在也可以作为支持转义字符串语法 (E'...') 的指示。

取值范围	默认值	设置分类
Boolean	on	master session reload

statement_mem

Note: 仅当资源管理设置为资源队列时，statement_mem服务器配置参数才会生效。

为每个查询分配segment主机内存。使用此参数分配的内存量不能超过max_statement_mem或提交查询的资源队列的内存限制。
当gp_resqueue_memory_policy =auto时，statement_mem和资源队列内存限制控制查询内存使用量。

如果查询需要额外的内存，则使用磁盘上的临时溢出文件。

该计算可用于估计各种情况的合理值。

```
( gp_vmem_protect_limitGB * .9 ) / max_expected_concurrent_queries
```

将gp_vmem_protect_limit 设置为8192MB (8GB) 并假设最多包含10%缓冲的40个并发查询

```
(8GB * .9) / 40 = .18GB = 184MB
```

更改max_statement_mem和statement_mem时，必须先更改max_statement_mem，或者先在postgresql.conf文件中列出。

取值范围	默认值	设置分类

KB	128MB	master session reload
----	-------	-----------------------------

statement_timeout

中止任何超过指定毫秒数的语句。 0关闭限制。

取值范围	默认值	设置分类
毫秒数	0	master session reload

stats_queue_level

Note: 仅当资源管理设置为资源队列时, stats_queue_level服务器配置参数才会生效。

收集有关数据库活动的资源队列统计信息。

取值范围	默认值	设置分类
Boolean	off	master session reload

superuser_reserved_connections

确定为Greenplum数据库超级用户保留的连接插槽数。

取值范围	默认值	设置分类
整数 < <i>max_connections</i>	3	local system

restart

tcp_keepalives_count

在连接被认为死亡之前，可能会丢失多少个Keepalive。 值0使用系统默认值。 如果不支持TCP_KEEPcnt，则此参数必须为0。

将此参数用于不在primary和mirror之间的所有连接。

取值范围	默认值	设置分类
丢失的Keepalive数量	0	local system restart

tcp_keepalives_idle

在空闲连接上发送Keepalive之间的秒数。 值0使用系统默认值。 如果不支持TCP_KEEPidle，则此参数必须为0。

将此参数用于不在primary和mirror之间的所有连接。

取值范围	默认值	设置分类
秒数	0	local system restart

tcp_keepalives_interval

在重新传输之前等待keepalive响应的秒数。 值0使用系统默认值。 如果不支持TCP_KEEPINTVL，则此参数必须为0。

将此参数用于不在primary和mirror之间的所有连接。

取值范围	默认值	设置分类
number of seconds	0	local system

		restart
--	--	---------

temp_buffers

以块为单位设置最大内存，以允许每个数据库会话使用临时缓冲区。这些是仅用于访问临时表的会话本地缓冲区。可以在单个会话中更改设置，但只能在会话中首次使用临时表之前更改。在实际上不需要大量临时缓冲区的会话中设置大值的成本只是每个块的缓冲区描述符，或每个增量大约64个字节。但是，如果实际使用了缓冲区，则将消耗额外的32768字节。

您可以将此参数设置为32K块的数量（例如，1024以允许32MB用于缓冲区），或指定允许的最大内存量（例如，对于1536块，为'48MB'）。`gpconfig`实用程序和`SHOW`命令报告临时缓冲区允许的最大内存量。

取值范围	默认值	设置分类
整数	1024 (32MB)	master session reload

TimeZone

设置显示和解释时间戳的时区。默认设置是使用系统环境指定的任何时区。请参阅PostgreSQL文档中的[日期/时间关键字](#)。

取值范围	默认值	设置分类
时区缩写		local restart

timezone_abbreviations

设置服务器为日期时间输入接受的时区缩写集合。默认值为Default，这是一个适用于世界大部分地区的集合。可以为特定安装定义Australia和India以及其他集合。可能的值是存储在\$GPHOME/share/postgresql/timezonesets/中的配置文件的名称。

要将Greenplum数据库配置为使用自定义时区集合，请将包含时区定义的文件复制到Greenplum数据库master和segment主机上的\$GPHOME/share/postgresql/timezonesets/目录中。然后将服务器配置参数`timezone_abbreviations`的值设置为该文件。例如，要使用包含默认时区和WIB（Waktu Indonesia Barat）时区的文件custom。

- 从目录\$GPHOME/share/postgresql/timezonesets/复制文件Default到文件custom。将文件Asia.txt中的WIB时区信息添加

到custom。

2. 将文件custom复制到Greenplum数据库master和segment主机上的\$GPHOME/share/postgresql/timezonesets/目录。
3. 将服务器配置参数timezone_abbreviations的值设置为custom。
4. 重新加载服务器配置文件 (gpstop -u)。

取值范围	默认值	设置分类
字符串	默认值	master session reload

track_activity_query_size

设置存储在系统catalog视图pg_stat_activity 的current_query列中的查询文本的最大长度限制。最小长度为1024个字符。

取值范围	默认值	设置分类
整数	1024	local system restart

transaction_isolation

设置当前事务隔离级别。

取值范围	默认值	设置分类
read committed	read committed	master
serializable		session reload

transaction_read_only

设置当前事务的只读状态。

取值范围	默认值	设置分类
Boolean	off	master session reload

transform_null_equals

启用时，表达式expr = NULL（或NULL = expr）被视为expr IS NULL，即，如果expr计算为空值，则返回true，否则返回false。正确的符合SQL规范的expr = NULL行为始终返回null（未知）。

取值范围	默认值	设置分类
Boolean	off	master session reload

unix_socket_directory

指定服务器要侦听来自客户端应用程序的连接的UNIX域套接字的目录。

取值范围	默认值	设置分类
目录路径	unset	local system restart

unix_socket_group

设置UNIX域套接字的拥有组。默认情况下，这是一个空字符串，它使用当前用户的默认组。

取值范围	默认值	设置分类

UNIX组名称	unset	local system restart
---------	-------	----------------------------

unix_socket_permissions

设置UNIX域套接字的访问权限。 UNIX域套接字使用通常的UNIX文件系统权限集。 请注意，对于UNIX域套接字，只有写权限才对。

取值范围	默认值	设置分类
数字UNIX文件权限模式 (由 <i>chmod</i> 或 <i>umask</i> 命令接受)	511	local system restart

update_process_title

每次服务器收到新的SQL命令时，都可以更新进程标题。 通常由ps命令查看进程标题。

取值范围	默认值	设置分类
Boolean	on	local system restart

vacuum_cost_delay

超出成本限制时vacuum进程将休眠的时间长度。 0禁用基于成本的vacuum延迟功能。

取值范围	默认值	设置分类
毫秒 < 0 (以10的倍数表示)	0	local system restart

vacuum_cost_limit

导致vacuum进程进入睡眠状态的累积成本。

取值范围	默认值	设置分类
整数 > 0	200	local system restart

vacuum_cost_page_dirty

vacuum修改先前干净的块时收取的估计成本。 它表示将脏块再次刷新到磁盘所需的额外I/O.

取值范围	默认值	设置分类
整数 > 0	20	local system restart

vacuum_cost_page_hit

vacuum共享缓冲区缓存中找到的缓冲的估计成本。 它表示锁定缓冲池，查找共享哈希表和扫描页面内容的成本。

取值范围	默认值	设置分类
整数 > 0	1	local system restart

vacuum_cost_page_miss

`vacuum`必须从磁盘读取的缓冲的估计成本。这表示锁定缓冲池，查找共享哈希表，从磁盘读取所需块并扫描其内容的代价。

取值范围	默认值	设置分类
整数 > 0	10	local system restart

vacuum_freeze_min_age

指定VACUUM在扫描表时决定是否用FrozenXID 替换事务ID时应使用的截止年龄（在事务中）。

有关VACUUM和事务ID管理的信息，请参阅[Greenplum数据库管理员指南](#)和[PostgreSQL文档](#)中的“管理数据”。

取值范围	默认值	设置分类
整数 0-100000000000	100000000	local system restart

validate_previous_free_tid

启用验证空闲元组ID (TID) 列表的测试。该列表由Greenplum数据库维护和使用。Greenplum数据库通过确保当前空闲元组的先前空闲TID是有效的空闲元组来确定空闲TID列表的有效性。默认值为true，启用测试。

如果Greenplum数据库检测到空闲TID列表中的损坏，则会重建空闲TID列表，并记录警告，并且检查失败的查询会返回警告。Greenplum数据库尝试执行查询。

取值范围	默认值	设置分类
Boolean	true	master session reload

verify_gpfists_cert

使用gpfists协议定义Greenplum数据库外部表以使用SSL安全性时，此参数控制是否启用SSL证书身份验证。默认设置为true，当Greenplum数据库与gpfist实用程序通信以从外部数据源读取数据或将数据写入外部数据源时，将启用SSL身份验证。

值false将禁用SSL证书身份验证。忽略这些SSL异常：

- Greenplum数据库不信任gpfist使用的自签名SSL证书。
- SSL证书中包含的主机名与运行gpfist的主机名不匹配。

您可以将值设置为false以在测试Greenplum数据库外部表与提供外部数据的gpfist实用程序之间的通信时禁用身份验证。

Warning: 通过不验证gpfists SSL证书禁用SSL证书身份验证会暴露安全风险。

有关gpfists协议的信息，请参阅[gpfists:// Protocol](#)。有关运行gpfist实用程序的信息，请参阅[gpfist](#)。

取值范围	默认值	设置分类
Boolean	true	master session reload

vmem_process_interrupt

在Greenplum数据库查询执行期间，为查询预留vmem内存之前，启用检查中断。在为查询保留进一步的vmem之前，请检查查询的当前会话是否具有挂起的查询取消或其他挂起的中断。这确保了更具响应性的中断处理，包括查询取消请求。默认为关闭。

取值范围	默认值	设置分类
Boolean	off	master session reload

wal_receiver_status_interval

对于Greenplum数据库主镜像，设置发送到活动master的walreceiver进程状态消息之间的间隔（以秒为单位）。在高负载下，时间可能会更长。

[replication_timeout](#) 的值控制walsender进程等待walreceiver消息的时间。

取值范围	默认值	设置分类
整数 0- INT_MAX/1000	10 sec	master system reload superuser

writable_external_table_bufsize

Greenplum数据库用于网络通信的缓冲区大小（以KB为单位），例如gpfdist实用程序和外部Web表（使用http）。Greenplum数据库在写入数据之前将数据存储在缓冲区中。有关gpfdist的信息，请参阅Greenplum数据库实用程序指南。

取值范围	默认值	设置分类
整数 32 - 131072 (32KB - 128MB)	64	local session reload

xid_stop_limit

发生事务ID环绕的ID之前的事务ID数。达到此限制时，Greenplum数据库会停止创建新事务，以避免因事务ID环绕而导致数据丢失。

取值范围	默认值	设置分类
整数 10000000 - 2000000000	1000000000	local system restart

xid_warn_limit

[xid_stop_limit](#)指定的限制之前的事务ID数。当Greenplum数据库达到此限制时，它会发出警告以执行VACUUM操作，以避免因事务ID环绕而导致数据丢失。

取值范围	默认值	设置分类

整数 10000000 - 2000000000	500000000	local system restart
--------------------------	-----------	----------------------------

xmlbinary

指定如何在XML数据中编码二进制值。例如，当bytea值转换为XML时。二进制数据可以转换为base64编码或十六进制编码。默认值为base64。

可以为数据库系统，单个数据库或会话设置该参数。

取值范围	默认值	设置分类
base64	base64	master
hex		session reload

xmloption

指定是否将XML数据视为执行隐式解析和序列化的操作的XML文档 (document) 或XML内容片段 (content) 。默认为content。

此参数会影响xml_is_well_formed()执行的验证。如果值是document，则函数将检查格式良好的XML文档。如果值是content，则函数检查格式良好的XML内容片段。

Note: 包含文档类型声明 (DTD) 的XML文档不被视为有效的XML内容片段。如果xmloption设置为content，则包含DTD的XML不被视为有效的XML。

要将包含DTD的字符串转换为xml数据类型，请将xmlparse函数与document关键字一起使用，或将xmloption值更改为document。

可以为数据库系统，单个数据库或会话设置该参数。Greenplum数据库中也提供了为会话设置此选项的SQL命令。

```
SET XML OPTION { DOCUMENT | CONTENT }
```

取值范围	默认值	设置分类
document	content	master
content		session reload

Greenplum数据库® 6.0文档

- 管理员指南
- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象
- 分布与倾斜
- 插入, 更新, 和删除数据
- 查询数据

关于Greenplum的查询
处理

- 关于GPORCA
- 定义查询
- WITH查询 (公用表表达式)
- 使用函数和操作符
- 使用JSON数据
- 使用XML数据
- 使用全文搜索
- 查询性能
- 管理查询生成的溢出文件
- 查询分析
- 使用外部数据

这个主题给出了Greenplum数据库如何处理查询的概述。理解这一处理有助于编写和调优查询。

用户像对任何数据库管理系统那样将查询发送到Greenplum数据库。它们使用`psql`之类的客户端应用连接到Greenplum的Master主机上的数据库实例并且提交SQL语句。

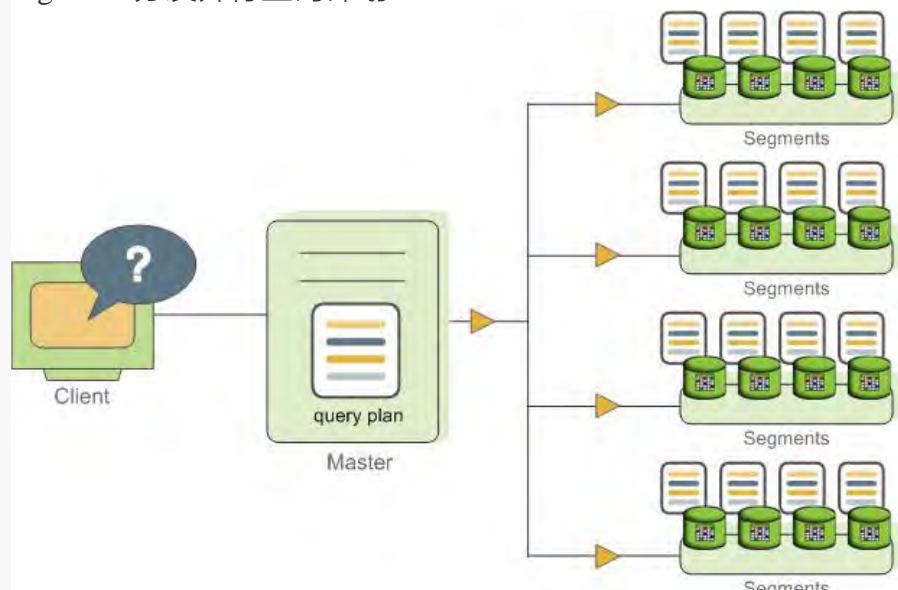
Parent topic: [查询数据](#)

理解查询规划和分发

Master接收、解析并且优化查询。作为结果的查询计划可能是并行的或者定向的。如图 [Figure 1](#) 所示, Master会把并行查询计划分发到所有的Segment。而如图 [Figure 2](#) 所示, Master会把定向查询计划分发到单一的一个segment实例。每个segment实例负责在其自己的数据集上执行本地数据库操作。

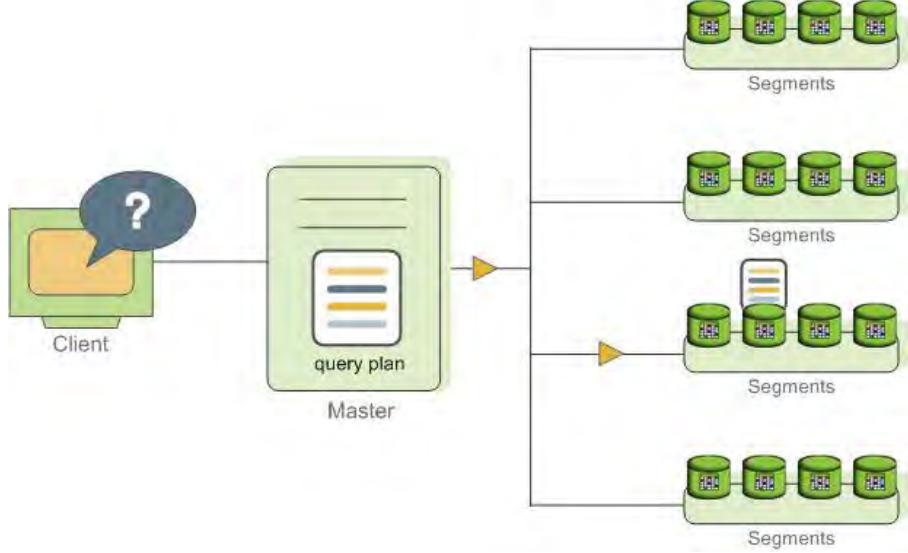
大部分的数据库操作 (例如表扫描、连接、聚集和排序) 都会以并行的方式在所有segment实例上执行。在一个segment实例的数据库上执行的每个操作都独立于存储在其他segment实例数据库中的数据。

Figure 1. 分发并行查询计划



某些查询可能只访问单个Segment上的数据, 例如单行的`INSERT`, `UPDATE`, `DELETE`, 或者 `SELECT`操作或者以表分布键列`partitioned by`的查询。在这些查询中, segment实例, 而是定向给到包含受影响或者相关行的segment实例。

Figure 2. 分发定向查询计划



理解Greenplum的查询计划

查询计划是Greenplum数据库将要执行以产生查询答案的操作集合。计划中的每个节点或者步骤表示一个数据库操作，例如表扫描、连接、聚集或者排序。计划的读取和执行按照从底向上的顺序进行。

除通常的数据库操作（例如表扫描、连接等等）之外，Greenplum数据库还有一种额外的被称为移动的操作类型。移动操作涉及到在查询处理期间在segment实例之间移动元组。注意并非每一个查询都需要移动操作。例如，定向查询计划就不需要通过Interconnect移动数据。

为了在查询执行期间达到最大并行度，Greenplum将查询计划的工作划分成切片。切片是Segment能够在其上独立工作的计划片段。只要有一个移动操作出现在计划中，该查询计划就会被切片，在移动的两端分别有一个切片。

例如，下面涉及两个表之间连接的简单查询：

```
SELECT customer, amount
FROM sales JOIN customer USING (cust_id)
WHERE dateCol = '04-30-2016';
```

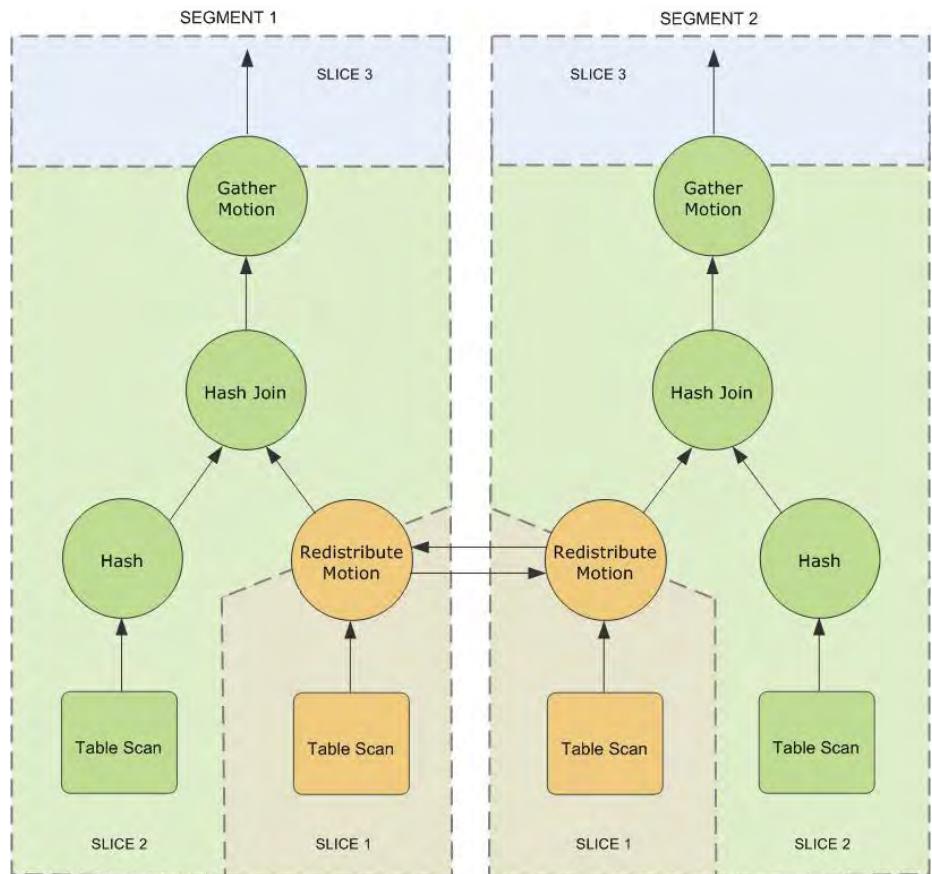
[Figure 3](#) 展示了这个查询计划。每个segment实例接收一份查询计划的拷贝并且并行地根据计划工作。

这个例子的查询计划有一个重分布移动，它在segment实例之间移动元组以完成连接。重分布移动是必要的，因为customer表在Segment上按照cust_id分布，而sales表是按照sale_id分布。为了执行该连接，sales元组必须按照cust_id重新分布。该计划在重分布移动操作的

两边被切换，形成了slice 1和slice 2。

这个查询计划由另一种称为收集移动的移动操作。收集操作表示segment实例何时将结果发回给Master，Master再将结果呈现给客户端。由于只要有移动产生查询计划就会被切片，这个计划在其最顶层也有一个隐式的切片（slice 3）。不是所有的查询计划都涉及收集移动。例如，一个CREATE TABLE x AS SELECT...语句不会有收集移动，因为元组都被发送到新创建的表而不是发给Master。

Figure 3. 查询切片计划



理解并行查询执行

Greenplum会创建若干数据库进程来处理查询的工作。在Master上，查询工作者进程被称作查询分发器（QD）。QD负责创建并且分发查询计划。它也收集并且表达最终的结果。在Segment上，查询工作者进程被称为查询执行器（QE）。QE负责完成它那一部分的工作并且与其他工作者进程交流它的中间结果。

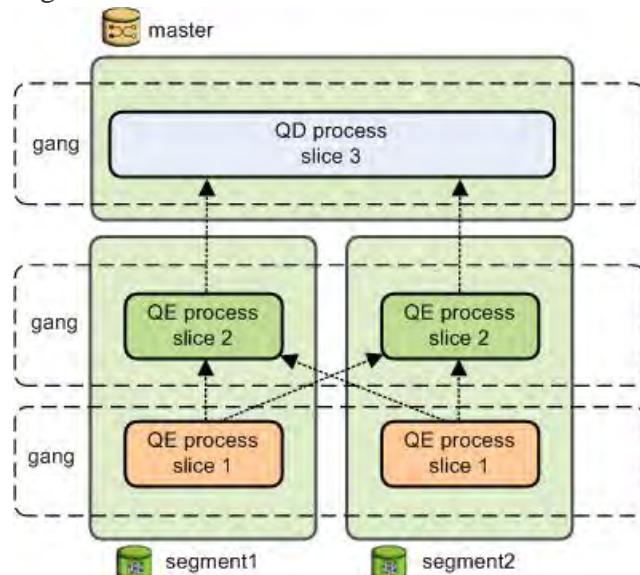
对查询计划的每一个切片至少要分配一个工作者进程。工作者进程独立地工作在分配给它的那部分查询计划上。在查询执行期间，每个Segment将有若干进程并行地为该查询工作。

为查询计划的同一个切片工作但位于不同Segment上的相关进程被称作

团伙。随着部分工作的完成，元组会从一个进程团伙流向查询计划中的下一个团伙。这种Segment之间的进程间通信被称作Greenplum数据库的Interconnect组件。

Figure 4 所示查询计划在Master和两个Segment实例上的查询工作者进行。Figure 3 .

Figure 4. 查询工作者进程



Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统
 - 管理Greenplum数据库访问
 - 定义数据库对象
 - 分布与倾斜
 - 插入, 更新, 和删除数据
 - 查询数据
 - 关于Greenplum的查询处理
 - 关于GPORCA
 - 定义查询
 - WITH查询 (公用表表达式)
 - 使用函数和操作符
 - 使用JSON数据
 - 使用XML数据
 - 使用全文搜索
 - 查询性能
 - 管理查询生成的溢出文件
 - 查询分析
- 使用外部数据

在Greenplum数据库中，默认的GPORCA优化器与传统查询优化器共存。

这些小节描述GPORCA的功能和用法：

- **GPORCA概述**

GPORCA扩展了Greenplum数据库传统优化器的规划和优化能力。

- **启用和禁用GPORCA**

默认情况下，Greenplum数据库使用GPORCA来替代传统查询规划器。服务器配置参数可以启用或者禁用GPORCA。

- **收集根分区统计信息**

对于分区表，GPORCA使用表根分区的统计信息来生成查询计划。这些统计信息用于确定联接顺序、拆分和联接聚合节点以及计算查询步骤的成本。相比之下，Postgres规划器使用每个叶分区的统计信息。

- **使用GPORCA时的考虑**

用GPORCA最优化执行查询需要考虑的查询条件。

- **GPORCA特性和增强**

GPORCA是Greenplum的下一代查询优化器，它包括了对特定类型的查询和操作的增强：

- **GPORCA改变的行为**

相比使用传统规划器，启用了GPORCA优化器（默认启用）的Greenplum数据库的行为有些改变。

- **GPORCA的限制**

在Greenplum数据库中使用默认的GPORCA优化器时有一些限制。GPORCA和传统的查询优化器当前并存于Greenplum数据库中，因为GPORCA不支持所有的Greenplum数据库特性。

- **判断被使用的查询优化器**

当GPORCA被启用（默认启用）时，可以判断Greenplum数据库是在使用GPORCA还是退回到传统查询优化器。

- **关于统一多级分区表**

Parent topic: [查询数据](#)



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

关于Greenplum的查询
处理

□ 关于GPORCA

定义查询

WITH查询 (公用表表达式)

使用函数和操作符

使用JSON数据

使用XML数据

□ 使用全文搜索

查询性能

管理查询生成的溢出文件

查询分析

□ 使用外部数据

Greenplum数据库基于PostgreSQL的SQL标准实现。

这个主题描述如何在Greenplum数据库中构造SQL查询。

- [SQL词汇](#)
- [SQL值表达式](#)

Parent topic: [查询数据](#)

SQL词汇

SQL是一种访问数据库的标准语言。该语言由数据存储、检索、分析、查看、操纵等元素组成。使用者可以使用SQL命令来构造Greenplum数据库引擎能够理解的查询和命令。命令由以正确语法顺序排列的合法符号序列构成，最后以分号(;)终结。

更多有关[SQL命令参考](#)。

Greenplum数据库使用PostgreSQL的结构和语法，但有一些不同。更多有关PostgreSQL中的SQL规则和概念的信息，请见PostgreSQL文档中的“SQL语法”。

SQL值表达式

SQL值表达式由一个或者多个值、符号、操作符、SQL函数及数据组成。值表达式会比较数据或者执行计算并且返回一个值作为结果。计算包括逻辑的、数学的和集合操作。

下列都是值表达式：

- 一个聚集表达式
- 一个数组构造器
- 一个列引用
- 一个常量或者字面值
- 一个相关子查询
- 一个域选择表达式
- 一个函数调用



- INSERT或者UPDATE中的一个新列值
- 一个操作符调用列引用
- 函数定义或者预备语句中的一个位置参数引用
- 一个行构造器
- 一个标量子查询
- WHERE子句中的一个搜索条件
- SELECT命令的一个目标列表
- 一个类型造型
- 圆括号中的一个值表达式，可用于分组子表达式以及覆盖优先级
- 一个窗口表达式

函数和操作符等SQL结构是表达式，但它们不遵循任何一般语法规则。更多有关这些结构的信息，请见使用函数和操作符[使用函数和操作符](#)。

列引用

列引用的形式是：

```
correlation.columnname
```

这里, correlation是一个FROM子句中定义的表的名字（可能有SCHEMA名限定）或者别名，或者是关键词NEW和OLD中的一个。NEW和OLD只能出现在重写规则中，但可以在任何SQL语句中使用其他关系名称。如果列名在该查询的所有表中都唯一，可以省略掉表引用的"correlation."部分。

位置参数

位置参数是SQL语句或者函数的参数，但使用它们在参数序列中的位置来引用。例如，\$1引用第一个参数，\$2是第二个参数，以此类推。位置参数的值从SQL语句的外部设置或者在SQL函数被调用时提供。一些客户端库支持在SQL命令之外单独指定数据值，在这种情况下参数引用的是线外数据值。参数引用的形式是：

```
$number
```

例如：

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

这里，只要该函数被调用，\$1就引用第一个函数参数的值。

下标

如果一个表达式得到的是一个数组类型的值，可以按照下面的方式抽取该数组值中的一个特定元素：

```
expression[subscript]
```

可以按下面的方式（包括方括号）抽取多个相邻的元素，它们被称为一个数组切片：

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

每一个下标都是一个表达式且得到一个整数值。

数组表达式通常必须放在圆括号内，但当被指定下标的表达式是一个列引用或者位置参数时可以省略圆括号。当原始数组是多维时可以串接多个下标。例如（包括圆括号）：

```
mytable.arraycolumn[4]
```

```
mytable.two_d_column[17][34]
```

```
$1[10:42]
```

```
(arrayfunction(a,b))[42]
```

域选择

如果一个表达式得到一个组合类型（行类型）的值，可以按照下面的方式抽取该行的一个特定域：

expression.fieldname

行表达式通常必须放在圆括号中，但当要从中选择的表达式是一个表引用或者位置参数时可以省略这些圆括号。例如：

mytable.mycolumn

\$1.somecolumn

(rowfunction(a,b)).col3

被限定的列引用是域选择语法的一种特殊情况。

操作符调用

操作符调用有下列可能的语法：

*expression operator expression(binary infix operator)**operator expression(unary prefix operator)**expression operator(unary postfix operator)*

其中*operator*是一个操作符符号、关键词AND, OR,或NOT之一，还可以是下面形式的限定操作符名：

OPERATOR(*schema.operatorname*)

可用的操作符以及它们是一元的还是二元的取决于系统或者用户定义的操作符。更多有关内建操作符的信息请见 [内建函数和操作符](#).

函数调用

函数调用的语法是一个函数名（可能由一个模式名限定），后面跟着圆括号中的参数：

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

例如，下面的函数调用计算 2 的平方根：

```
sqrt(2)
```

见[内建函数](#)。用户也可以增加自定义函数。

聚集表达式

聚集表达式在一个查询选择的行上应用一个聚集函数。聚集函数在一组值之上执行一次计算并且返回一个单一值，例如这一组值的总和或者平均。聚集表达式的语法是下列之一：

- *aggregate_name(expression [, ...])* — 在所有输入行上操作，预期的结果值非空。ALL 是默认。
- *aggregate_name(ALL expression [, ...])* — 和第一种形式相同，因为 ALL 是默认
- *aggregate_name(DISTINCT expression [, ...])* — 输入行的所有可区分非空值上操作。
- *aggregate_name(*)* — 在所有值为空以及非空的行上操作。通常这种形式对于 `count(*)` 聚集函数最有用。

其中 *aggregate_name* 是一个之前定义的聚集（可能有模式名限定）而 *expression* 是任何不包含聚集表达式的值表达式。

例如, `count(*)` 会得到输入行的总数, `count(f1)` 得到输入行中 *f1* 为非空的数量, 而 `count(distinct f1)` 会得到 *f1* 中可区分非空值的数量。

预定义的聚集函数, 可见[内建函数和操作符](#)。用户也可以增加自定义聚集函数。

Greenplum 函数提供 MEDIAN 聚集函数，它返回 PERCENTILE_CONT 结果和逆分布函数的特殊聚集表达式的第 50 个百分位数：

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

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

当前只能将这两个表达式与关键词 WITHIN GROUP 一起使用。

聚集函数的限制

下面是聚集表达式的当前限制：

- Greenplum数据库不支持下列关键词：ALL、DISTINCT、FILTER和OVER。详见 [Table 5](#)
- 聚集表达式只能出现在SELECT命令的结果列表或者HAVING子句中。在其他子句（例如WHERE）中不允许聚集表达式，因为那些子句逻辑上会在聚集形式的结果之前计算。这种限制适用于该聚集所属的查询级别。
- 当聚集表达式出现在子查询中时，该聚集通常在子查询的行之上计算。如果该聚集的参数只包含外层变量，该聚集属于最近的那个外层并且在那个外层查询的行上计算。该查询表达式总的来说是其出现的子查询的外部引用，并且该聚集表达式在那个子查询的任何一次计算中都像一个常量。请参考 [标量子查询](#) and [Table 3](#) .
- Greenplum数据库不支持有多个输入表达式的DISTINCT。
- Greenplum数据库不支持指定聚合函数作为另一个聚合函数的参数。
- Greenplum数据库不支持指定窗口函数作为聚合函数的参数。

窗口表达式

窗口表达式允许应用开发者更容易地使用标准SQL命令构造复杂的在线分析处理（OLAP）查询。例如，通过窗口查询，用户可以在多个区间上计算移动平均或者总和、在选择的列值改变时重置聚集和排名以及用简单的术语表达合比。

窗口表达式表示将一个窗口函数应用到一个窗口帧，后者在一个特殊的OVER()子句中定义。窗口分区是一个行的集合，这些行被集合起来应用一个窗口函数。这与使用聚合函数和GROUP BY子句可以完成的计算类型相当。和聚集函数（为每个行分组返回一个结果值）不同，窗口函数为每一行返回一个结果值，但是该结果值是针对一个特定窗口分区中的行计算得来。如果没有指定分区，窗口函数会在完整的中间结果集上计算。

Greenplum数据库不支持将窗口函数制定为另一个窗口函数的参数。

窗口表达式的语法是：

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

```
    window_specification )
```

其中*window_function**window_function*是列在 [Table 4](#) 中的函数之一, *expression*是任何不含窗口表达式的值表达式, 而*window_specification*是:

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

其中*window_frame_bound*可以是下列之一:

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

窗口只能出现在SELECT命令的选择列表中。例如:

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

在窗口表达式中, 一个窗口函数调用必须紧跟OVER子句。OVER子句使得窗口函数与其他聚集或者报表函数相区别。OVER子句定义窗口函数要应用于其上的*window_specification*。

窗口说明具有下列特点:

- PARTITION BY子句定义窗口函数要应用于其上的窗口分区。如果省略, 整个结果集会被当做一个分区。
- ORDER BY子句定义在窗口分区内用来排序行的表达式。窗口说明的ORDER BY子句是独立的并且与一个常规查询表达式的ORDER BY子句不同。计算排名的窗口函数会要求ORDER BY子句, 因为它要确定排名值的排名程度。对于OLAP聚集, 使用窗口帧(ROWS和RANGE子句)也要求ORDER BY子句。

Note: 没有一致排序的数据类型列(如*time*)并不适合在窗口说明的ORDER BY子句中使用。*time*(不管有没有指定的时区)缺少一

种一致的排序，因为加法和减法无法得到预期的效果。例如，下面的式子并不总是为真：`x::time < x::time + '2 hour'::interval`

- `ROWS` 或者 `RANGE` 子句为聚集（非排名）窗口函数定义窗口帧。一个窗口帧在一个窗口分区上定义一个行集合。当一个窗口帧被定义时，窗口函数会在这个移动帧的内容上计算，而不是在整个窗口分区的固定内容上计算。窗口帧可以是基于行的（`ROWS`）或者基于值的（`RANGE`）。

窗口例子

以下示例演示如何使用带有分区和窗口框架的窗口函数

例1 – 分区上的聚合窗口函数

在 `OVER` 子句用 `PARTITION BY` 将行进行分组或分区，在相同值的分组内使用指定表达式。

此例子比较员工工资和部门平均工资：

```
SELECT depname, empno, salary, avg(salary) OVER(PARTITION
BY depname)
FROM empsalary;
      depname | empno | salary |          avg
-----+-----+-----+-----+
    develop |     9 |   4500 | 5020.0000000000000000
    develop |    10 |   5200 | 5020.0000000000000000
    develop |    11 |   5200 | 5020.0000000000000000
    develop |     7 |   4200 | 5020.0000000000000000
    develop |     8 |   6000 | 5020.0000000000000000
  personnel |     5 |   3500 | 3700.0000000000000000
  personnel |     2 |   3900 | 3700.0000000000000000
    sales |     1 |   5000 | 4866.6666666666666667
    sales |     3 |   4800 | 4866.6666666666666667
    sales |     4 |   4800 | 4866.6666666666666667
(10 rows)
```

前三个输出列来自表 `empsalary`，表中每一行都有一个输出行。第四列计算具有相同的 `depname` 值的平均数。相同的 `depname` 行组成一个分区，在这个例子中有三个分区。`avg` 函数与常规的聚合函数 `avg` 相同，但是 `OVER` 子句使它被用作一个窗口函数。

你可以把窗口的命名放在 `WINDOW` 子句中，在 `select` 中引用。这个例子相当于之前的查询：

```
SELECT depname, empno, salary, avg(salary) OVER(mywindow)
FROM empsalary
WINDOW mywindow AS (PARTITION BY depname);
```

当select列表中有多个使用相同功能的窗口函数时，定义一个命名的窗口比较有用。

例2 – 使用ORDER BY子句对窗口函数进行排序

在OVER子句中的ORDER BY，控制窗口函数处理行的顺序。窗口函数的ORDER BY列表不必匹配查询的输出顺序。这个例子使用rank()窗口函数来排序部门内的员工工资：

```
SELECT depname, empno, salary,
       rank() OVER (PARTITION BY depname ORDER BY salary DESC)
FROM empsalary;
      depname | empno | salary | rank
-----+-----+-----+-----
  develop |     8 |   6000 |    1
  develop |    11 |   5200 |    2
  develop |    10 |   5200 |    2
  develop |     9 |   4500 |    4
  develop |     7 |   4200 |    5
personnel |     2 |   3900 |    1
personnel |     5 |   3500 |    2
sales     |     1 |   5000 |    1
sales     |     4 |   4800 |    2
sales     |     3 |   4800 |    2
(10 rows)
```

例3 – 行窗口帧的具体函数

A RANGE 或 ROWS子句定义窗口函数计算的窗口帧(一个分区内的多个行)。ROWS指定要处理的物理行，例如从分区开始到当前行之间的所有行。

此例按部门计算员工工资的连续总额。使用sum()函数计算分区开始到当前行的总行数：

```
SELECT depname, empno, salary,
       sum(salary) OVER (PARTITION BY depname ORDER BY salary
                         ROWS between UNBOUNDED PRECEDING AND CURRENT ROW)
FROM empsalary ORDER BY depname, sum;
      depname | empno | salary | sum
-----+-----+-----+-----
  develop |     7 |   4200 |  4200
```

develop	9	4500	8700
develop	11	5200	13900
develop	10	5200	19100
develop	8	6000	25100
personnel	5	3500	3500
personnel	2	3900	7400
sales	4	4800	4800
sales	3	4800	9600
sales	1	5000	14600
(10 rows)			

例4 – Range窗口帧的聚合函数

RANGE 指定OVER子句中基于ORDER BY 表达式的逻辑值。这个例子展示了 ROWS和RANGE的不同。窗口帧包含小于或等于当前行的所有行的工资值。与之前的例子不同，相同工资的员工，总和是相同的，包括所有这些员工的工资。

SELECT depname, empno, salary,
sum(salary) OVER (PARTITION BY depname ORDER BY salary
RANGE between UNBOUNDED PRECEDING AND CURRENT ROW)
FROM empsalary ORDER BY depname, sum;
depname empno salary sum
-----+-----+-----+-----
develop 7 4200 4200
develop 9 4500 8700
develop 11 5200 19100
develop 10 5200 19100
develop 8 6000 25100
personnel 5 3500 3500
personnel 2 3900 7400
sales 4 4800 9600
sales 3 4800 9600
sales 1 5000 14600
(10 rows)

类型转换

类型转换指定从一种数据类型到另一种数据类型的转换。应用到已知类型的值表达式是运行时的类型转换。仅当定义了适当的类型转换时才成功。这个不同于与带有常量的强制转换，应用于字符串文本的强制转换表示类型对文本常量值的初始赋值，因此如果字符串文字的内容是可接受的数据类型的输入语法，则该转换对任何类型都成功。

Greenplum数据库支持三种类型的值表达式转换：

- 显示转换 - 当显式指定两种数据类型时的强制转换。Greenplum数

数据库接受两种等效的类型造型语法：

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

The `CAST`语法符合SQL，带`::`的语法是PostgreSQL的一种历史用法。

- 赋值转换 - 当被赋值给一个表列时，Greenplum数据库会进行隐式转换。例如，`CREATE CAST` 命令在使用`AS ASSIGNMENT`子句，在赋值语句中创建转换被隐式使用。这个例子赋值转换，假定`tbl1.f1` 是一个`text` 列，`INSERT`命令被允许，因为值从`integer` 隐式转换为`text` 。

```
INSERT INTO tbl1 (f1) VALUES (42);
```

- 隐式转换 - Greenplum数据库在赋值或表达式中隐式进行转换。例如，`CREATE CAST` 命令在 `AS IMPLICIT` 子句中创建一个隐式转换，在赋值和表达式中隐式进行转换。这个例子隐式转换，假定`tbl1.c1` 是一个`int` 列。对于谓词中的计算，`c1` 从`int` 隐式转换为`decimal` 。

```
SELECT * FROM tbl1 WHERE tbl1.c2 = (4.3 + tbl1.c1) ;
```

如果值表达式必须生成的类型没有歧义（例如，当它被分配给表列时），则通常可以省略显式类型转换；系统会自动应用类型转换。Greenplum数据库系统只会自动应用在系统目录中标记为"OK to apply implicitly"的转换。必须使用显式强制转换语法调用其他强制转换，以防止在用户不知情的情况下应用意外的转换。

可以通过`psql meta`命令\dc显示强制转换信息。CAST信息存储在目录表`pg_cast`中，类型信息存储在目录表`pg_type`中。

标量子查询

标量子查询是一个圆括号中的`SELECT`查询，它返回正好一行一列。不要将返回多行或者多列的`SELECT`查询用作一个标量子查询。该查询会运行并且把返回的值用在外围的值表达式中。相关标量子查询包含对于外层查询块的引用。

相关子查询

相关子查询 (CSQ) 是一个SELECT查询，其WHERE子句或者目标列表包含对于外层子句的引用。 CSQ能有效地根据另一个查询的结果来表达结果。 Greenplum数据库支持相关子查询，为很多现有的应用提供了兼容性。 CSQ可以是一个标量或者表子查询，这取决于它返回一行还是多行。 Greenplum数据库不支持跨级关联的相关子查询。

相关子查询例子

例1 – 标量相关子查询

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

例2 – 相关EXISTS子查询

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

Greenplum数据库使用下列方法之一运行CSQ:

- 解除CSQ的嵌套变成连接操作 – 这种方法最高效，并且Greenplum数据库会以这种方法运行大多数CSQ，包括来自TPC-H基准的查询。
- 在外层查询的每一行上运行CSQ – 这种方法相对低效，Greenplum数据库会用这种方法来运行含有在SELECT列表中或者由OR条件连接的CSQ的查询。

下面的例子展示了如何重写这些类型的查询来改进性能。

例3 - 选择列表中的CSQ

原始查询

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

重写这一查询，让它先执行与t1的内连接，然后再执行与t1的左连接。这种重写只适用于相关条件中的等值连接。

重写查询

```
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4 - 由OR子句连接的CSQ

原始查询

```
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)
```

重写这一查询，把它根据OR条件分解成两个部分，然后联合起来。

重写查询

```
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)
```

要查看查询计划，使用EXPLAIN SELECT或者EXPLAIN ANALYZE SELECT。查询计划中的子计划节点表示该查询将在外层查询的每一行上运行，并且该查询是重写的候选。更多有关这些语句的信息，请见[查询分析](#)。

数组构造器

数组构造器是一个从其成员元素的值构造一个数组值的表达式。简单的数组构造器由关键词 ARRAY、一个左方括号[、一个或多个由逗号分隔的表达式（用于数组元素值）以及一个右方括号]组成。例如，

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

数组元素类型是其成员表达式的共同类型，采用和UNION或者CASE结构相同的规则确定。

可以通过嵌套数组构造器构建多维数组值。在内层构造器中，可以省略关键词ARRAY。例如，下面的两个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}}
```

由于多维数组必须是矩形，同一层次上的内层构造器必须产生同维的子数组。

多维数组构造器元素不限于一个子ARRAY结构，它们可以是任何产生正确种类数组的东西。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}}}
```

可以从子查询的结果构造数组。数组构造器写成关键词ARRAY后面跟上圆括号中的一个子查询。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。作为结果的一维数组中每一个元素对应于子查询结果中的每一行，元素类型匹配子查询的输出列。用ARRAY构建的数组值的下标总是从1开始。

行构造器

行构造器是一个从其成员域的值构建一个行值（也被称为组合值）的表达式。例如，

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

行构造器的语法是`rowvalue.*`，当在SELECT列表的顶层使用语法`.*`时，它会扩展成该行值的元素的列表。例如，如果表t有列f1和f2，下列查询是相同的：

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

默认情况下，`ROW`表达式创建的值是一种匿名记录类型。如果有必要，它可以被造型成一种命名的组合类型——一个表的行类型或者用`CREATE TYPE AS`创建的一种组合类型。为了避免歧义，必要时可以对该值进行显式转换。例如：

```
CREATE TABLE mytable(f1 int, f2 float, f3 text);
CREATE FUNCTION getf1(mytable) RETURNS int AS 'SELECT
$1.f1'
LANGUAGE SQL;
```

在下面的查询中，不需要对值转换，因为只有一种`getf1()`函数，所以没有歧义：

```
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;
```

现在我们需要一次转换来指示要调用哪个函数：

```
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
```

可以使用行构造器来构建组合值，该值可以被存储在组合类型表列或者被传递给接受组合类型参数的函数。

表达式计算规则

子表达式的计算顺序未被定义。一个操作符或者函数的输入不必从左向右或者按照任何其他固定顺序计算。

如果通过仅计算表达式的一部分就能确定该表达式的结果，那么其他子表达式可能完全不被计算。例如，在下面的表达式中：

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

`somefunc()` 可能根本不会被调用。在下面的额表达式中也是如此：

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

这和某些编程语言中的布尔操作符实施的从左向右的计算顺序不同。

不要把有副作用的函数用在复杂表达式中，尤其是在**WHERE**和**HAVING**子句中，因为在形成执行计划时会广泛地预处理那些子句。那些子句中的布尔表达式（**AND/OR/NOT**组合）可能会被以布尔代数法允许的任何方式重新组织。

要强制计算顺序，可使用 **CASE**结构。下面的例子是一种在 **WHERE**子句中避免除零的不可靠的方法：

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

下面的例子展示了一种可靠的计算顺序：

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

这种**CASE**结构的用法会使得优化器无法进行优化尝试，因此只有在必要时才使用它。

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

关于Greenplum的查询
处理

□ 关于GPORCA

定义查询

WITH查询 (公用表表达式)

使用函数和操作符

使用JSON数据

使用XML数据

□ 使用全文搜索

查询性能

管理查询生成的溢出文件

查询分析

□ 使用外部数据

WITH子句提供在一个更大的SELECT查询中，使用子查询或执行数据修改操作的方式。你可以在INSERT, UPDATE, 或 DELETE命令中使用WITH子句。

在WITH子句中的查询 在WITH子句中使用SELECT的相关信息

WITH子句中的数据修改语句, 在WITH子句中使用INSERT, UPDATE, or DELETE

Note: 这些是使用WITH子句的限制。

- 对于包含WITH子句的SELECT命令，该子句最多只能包含一个修改表数据的子句 (INSERT, UPDATE, 或 DELETE命令)。
- 对于包含WITH子句的数据修改命令 (INSERT, UPDATE, 或 DELETE)，该子句只能包含SELECT命令，WITH子句不能包含数据修改命令。

默认情况下，将启用WITH子句的RECURSIVE关键字。通过将服务器配置参数gp“recursive”设置为false，可以禁用递归。By default, the RECURSIVE keyword for the WITH clause is enabled. 通过将服务器配置参数gp_recursive_cte 设置为false，WITH子句的RECURSIVE关键字被禁用。

Parent topic: [查询数据](#)

在WITH子句中的查询

子查询通常被称为公共表表达式或CTE，可以认为是为查询定义临时表。这些示例显示了与SELECT命令一起使用的WITH子句。带WITH子句的示例可以以插入、更新或删除的相同方式使用。在每种情况下，WITH子句都有效地提供了可以在主命令中引用的临时表。

WITH子句中的SELECT命令在每次执行父查询时只计算一次，即使父查询或WITH子句的同级多次引用了该命令。因此，需要在多个地方进行的昂贵计算可以放在WITH子句中，以避免重复工作。另一个可能的应用是防止对具有副作用的函数进行不必要的多次计算。然而，这种情况的另一方面是，与普通的子查询相比，优化器无法将来自父查询的限制向下推送到WITH查询中。WITH查询通常将按写入方式进行计算，而不禁止父查询随后可能丢弃的行。但是，如果对查询的引用只需要有限的行数，则计算可能会提前停止。

此功能的一个用途是将复杂的查询分解为简单的部分。此示例查询仅在顶部销售区域中显示每个产品的销售总额：

```
WITH regional_sales AS (
    SELECT region, SUM(amount) AS total_sales
    FROM orders
    GROUP BY region
), top_regions AS (
    SELECT region
    FROM regional_sales
    WHERE total_sales > (SELECT SUM(total_sales)/10 FROM
regional_sales)
)
SELECT region,
       product,
       SUM(quantity) AS product_units,
       SUM(amount) AS product_sales
FROM orders
WHERE region IN (SELECT region FROM top_regions)
GROUP BY region, product;
```

可以不使用WITH子句编写查询，但需要两级嵌套子选择。相比较来说，WITH子句比较容易。

当启用可选RECURSIVE关键字时，WITH子句可以完成标准SQL中不可能完成的事情。使用RECURSIVE，WITH子句中的查询可以引用自己的输出。这是一个简单的例子，它计算从1到100的整数之和：

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
    UNION ALL
    SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

递归WITH子句（使用WITH关键字的WITH子句）的一般形式是一个非递归术语，后跟一个 UNION（或 UNION ALL），然后是一个递归术语，其中只有递归术语可以包含对查询输出的引用。

```
non_recursive_term UNION [ ALL ] recursive_term
```

包含 UNION [ALL]的递归WITH查询执行如下：

1. 计算非递归项。对于UNION（而不是UNION ALL），丢弃重复的行。包括递归查询结果中的所有剩余行，并将它们放在临时工作表中。
2. 只要工作台不是空的，重复以下步骤：

- a. 计算递归项，将工作表的当前内容替换为递归自引用。对于 UNION (而不是UNION ALL) , 放弃重复的行和复制任何先前结果行的行。在递归查询的结果中包含所有剩余的行，并将它们放在临时中间表中。
- b. 将当前工作表的内容替换为中间表的内容，然后清空中间表。

Note: 严格来说，过程是迭代而不是递归的，但是RECURSIVE是由SQL标准委员会选择的术语。

递归WITH查询通常用于处理层次结构或树结构数据。例如，该查询查找产品的所有直接和间接子部分，只给出一个显示直接包含内容的表：

```
WITH RECURSIVE included_parts(sub_part, part, quantity) AS (
    SELECT sub_part, part, quantity FROM parts WHERE part = 'our_product'
    UNION ALL
    SELECT p.sub_part, p.part, p.quantity
    FROM included_parts pr, parts p
    WHERE p.part = pr.sub_part
)
SELECT sub_part, SUM(quantity) as total_quantity
FROM included_parts
GROUP BY sub_part ;
```

使用递归WITH时，必须确保查询的递归部分最终不会返回元组，否则查询将无限期循环。在计算整数和的示例中，工作表在每个步骤中包含一行，并在连续步骤中接受从1到100的值。在第100步中，由于WHERE子句没有输出，查询终止。

对于某些查询，使用 UNION而不是 UNION ALL可以通过丢弃重复以前输出行的行来确保查询的递归部分最终不返回元组。然而，一个循环通常不涉及完全重复的输出行：只检查一个或几个字段就足够了，以查看以前是否达到了相同的点。处理这种情况的标准方法是计算访问值的数组。例如，考虑使用链接字段搜索表图形的以下查询：

```
WITH RECURSIVE search_graph(id, link, data, depth) AS (
    SELECT g.id, g.link, g.data, 1
    FROM graph g
    UNION ALL
    SELECT g.id, g.link, g.data, sg.depth + 1
    FROM graph g, search_graph sg
    WHERE g.id = sg.link
)
SELECT * FROM search_graph;
```

如果链接关系包含循环，则此查询将循环。因为查询需要深度输出，
 UNION ALL UNION

所以将 更改为 不会消除循环。相反，查询需要在遵循特定的链接路径时，识别它是否再次到达同一行。此修改后的查询将path和cycle两列，添加到容易循环的查询中：

```
WITH RECURSIVE search_graph(id, link, data, depth, path,
cycle) AS (
    SELECT g.id, g.link, g.data, 1,
           ARRAY[g.id],
           false
      FROM graph g
UNION ALL
    SELECT g.id, g.link, g.data, sg.depth + 1,
           path || g.id,
           g.id = ANY(path)
      FROM graph g, search_graph sg
     WHERE g.id = sg.link AND NOT cycle
)
SELECT * FROM search_graph;
```

除了检测循环之外，path的数组值本身也很有用，因为它表示到达任何特定行所需的路径。

在一般情况下，需要检查多个字段才能识别循环，可以使用一个行数组。例如，如果需要比较字段 f1 和 f2：

```
WITH RECURSIVE search_graph(id, link, data, depth, path,
cycle) AS (
    SELECT g.id, g.link, g.data, 1,
           ARRAY[ROW(g.f1, g.f2)],
           false
      FROM graph g
UNION ALL
    SELECT g.id, g.link, g.data, sg.depth + 1,
           path || ROW(g.f1, g.f2),
           ROW(g.f1, g.f2) = ANY(path)
      FROM graph g, search_graph sg
     WHERE g.id = sg.link AND NOT cycle
)
SELECT * FROM search_graph;
```

Tip: 如果只需要检查一个字段来识别循环，则忽略ROW()语法。它使用简单的数组而不是复合类型的数组，从而提高了效率。

Tip: 递归查询评估算法按广度优先搜索顺序生成输出。通过按这种方式构造的路径列对外部查询进行ORDER BY，可以按深度优先搜索顺序显示结果。

当您不确定查询是否可能无限循环时，测试查询的一种有用技术是在父查询中设置LIMIT。例如，如果不使用LIMIT子句，则此查询将永远循环：

```
WITH RECURSIVE t(n) AS (
    SELECT 1
    UNION ALL
    SELECT n+1 FROM t
)
SELECT n FROM t LIMIT 100;
```

该技术之所以有效，是因为递归WITH实现只计算WITH查询中实际由父查询提取的行数。不建议在生产中使用此技术，因为其他系统的工作方式可能不同。另外，如果外部查询使用结果对递归进行排序或将结果连接到另一个表，则该技术可能不起作用。

WITH子句中的数据修改语句

对于SELECT命令，可以在 WITH子句中使用数据修改命令 INSERT, UPDATE, 或DELETE。这允许您在同一查询中执行几个不同的操作。

WITH子句中的数据修改语句只执行一次，并且始终执行到完成，这与主查询是否读取所有（或任何）输出无关。这与在WITH子句中使用SELECT时的规则不同，只有在主查询要求输出时，才会继续执行SELECT。

这个简单的CTE查询从products中删除行。WITH子句中的 DELETE从产品中删除指定的行，并通过其返回子句返回其内容。

```
WITH deleted_rows AS (
    DELETE FROM products
    WHERE
        "date" >= '2010-10-01' AND
        "date" < '2010-11-01'
    RETURNING *
)
SELECT * FROM deleted_rows;
```

WITH子句中的数据修改语句必须有RETURNING子句，如前一个示例所示。它是RETURNING子句的输出，而不是数据修改语句的目标表，形成了可以被查询的其余部分引用的临时表。如果 WITH中的数据修改语句缺少RETURNING子句，则返回错误。

如果启用可选的RECURSIVE关键字，则不允许在数据修改语句中进行递归自引用。在某些情况下，可以通过引用递归的输出来绕过这个限制。例如，此查询将删除产品的所有直接和间接子部分。

```
WITH RECURSIVE included_parts(sub_part, part) AS (
    SELECT sub_part, part FROM parts WHERE part =
```

```
'our_product'
UNION ALL
  SELECT p.sub_part, p.part
  FROM included_parts pr, parts p
 WHERE p.part = pr.sub_part
)
DELETE FROM parts
WHERE part IN (SELECT part FROM included_parts);
```

WITH子句中的子语句与主查询同时执行。因此，在WITH中使用数据修改语句时，将在快照中执行该语句。语句的效果在目标表上不可见。RETURNING的数据是在不同子语句和主查询之间传递更改的唯一方法。在本例中，外部SELECT返回WITH子句中UPDATE操作之前的原始价格。

```
WITH t AS (
  UPDATE products SET price = price * 1.05
  RETURNING *
)
SELECT * FROM products;
```

在本例中，外部SELECT返回更新的数据。

```
WITH t AS (
  UPDATE products SET price = price * 1.05
  RETURNING *
)
SELECT * FROM t;
```

不支持在单个语句中更新同一行两次。这种语句的效果是不可预测的。只有一个修改发生了，但是不容易（有时也不可能）预测哪一个发生修改。

在WITH子句中用作数据修改语句目标的任何表都不能有条件规则、也不能有条件规则或扩展为多个语句的INSTEAD规则。

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

关于Greenplum的查询处理

□ 关于GPORCA

定义查询

WITH查询 (公用表表达式)

使用函数和操作符

使用JSON数据

使用XML数据

□ 使用全文搜索

查询性能

管理查询生成的溢出文件

查询分析

□ 使用外部数据

□ 装载和卸载数据

用户定义和内置函数和运算符的说明。

- [在Greenplum数据库中使用函数](#)
- [用户定义的函数](#)
- [内建函数和操作符](#)
- [窗口函数](#)
- [高级聚集函数](#)

Parent topic: [查询数据](#)

在Greenplum数据库中使用函数

在Greenplum数据库中调用函数时, 函数属性控制函数的执行。易变性属性 (IMMUTABLE, STABLE, VOLATILE) 和EXECUTE ON 属性控制函数执行的两个不同方面。一般来说, 易变性指示函数执行时间, EXECUTE ON 指示执行位置。易变性属性是基于PostgreSQL的属性, EXECUTE ON 属性是Greenplum的属性。

例如, 使用IMMUTABLE属性定义的函数可以在查询计划时执行, 而使用VOLATILE属性定义的函数, 必须对查询中每一行执行。使用EXECUTE ON MASTER属性的函数仅在主实例上执行, 而使用EXECUTE ON ALL SEGMENTS属性的函数在所有主segment实例 (而不是主服务器) 上执行。

这些表总结了Greenplum数据库基于这些属性, 对函数执行的假设。

Table 1. Greenplum数据库的函数易变性

函数属性	Greenplum支持	描述	注释
IMMUTABLE	支持	仅直接依赖信息参数列表。如果给定相同的参数值, 则始终返回同样的结果。	
STABLE	大部分情况下支持	在一个单一表扫描中, 相同的参数值返回相同的结果, 但是不同的SQL语句结果会改变。	结果依赖于数据库查找或者参数值。 <code>current_timestamp</code> 函数族是STABLE的。一次执行中值不会改变。
VOLATILE	受限	在一个单一表扫描中函数值可能会改变。例如: <code>random()</code> , <code>timeofday()</code> , 这是默认属性。	任何有副作用的函数都是volatile, 即使它的结果是可预测的也一样。例如: <code>setval()</code> 。

Table 2. Greenplum数据库的EXECUTE ON函数属性

函数属性	描述	注释
EXECUTE ON ANY	表示函数可以在master或任意segment实例上执行, 并且返回相同的结果, 而不管它在何处执行。这是默认属性。	Greenplum数据库决定函数的执行位置。
EXECUTE ON MASTER	表示必须在master实例上执行函数。	如果用户定义函数执行查询以访问表, 请指定此属性。
EXECUTE ON ALL SEGMENTS	表示对于每个调用, 函数必须在所有主segment实例上执行, 而不是在master上执行。	

您可以使用`psql \df+ function`命令显示函数的易变性和`EXECUTE ON`属性。

更多有关Greenplum数据库函数易变性分类的信息，请参考PostgreSQL文档[函数易变性分类](#)

更多关于`EXECUTE ON`属性的信息，参看[创建函数](#)。

在Greenplum数据库中，数据被划分为多个段-每个段都是一个不同的PostgreSQL数据库。为了防止不一致或意外的结果，如果函数包含SQL命令或者任意修改数据库的方式，请不要在segment级别执行被分类为volatile的函数。例如，不允许对Greenplum数据库中的分布式数据`setval()`等函数，因为它们可能导致段实例之间的数据不一致。

函数可以对segment上的复制表（`DISTRIBUTED REPLICATED`）执行只读查询，但修改数据的任何SQL命令都必须在master实例上执行。

为了确保数据的一致性，您可以在主服务器上安全地运行语句来调用使用`VOLATILE`和`STABLE`的函数。例如，下面的语句在master上运行（不带`FROM`子句的语句）：

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

如果语句有一个包含分布式表的`FROM`子句，而`FROM`子句中的函数返回一组行，则该语句可以在以下segment上运行：

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

Greenplum数据库不支持返回表引用（`rangeFuncs`）的函数或使用`refCursor`数据类型的函数。

函数易变性和计划缓冲

对于计划和立即执行的简单交互式查询，`STABLE`和`IMMUTABLE` 的函数易变性类别之间的差异相对较小。无论是在计划期间执行一次函数，还是在查询执行启动期执行一次函数，都无关紧要。但是，当您保存计划并稍后重用它时，会有很大的不同。如果把一个函数误标为`IMMUTABLE`，Greenplum数据库可能在规划时会过早地把它折叠成一个常量，可能会在计划的后续执行期间重用过时的值。当使用`PREPARE`语句或使用PL/pgSQL之类的语言缓存计划时，可能会遇到这种危险。

用户定义的函数

Greenplum数据库支持用户定义的函数。更多信息请见PostgreSQL文档中的[Extending SQL](#)

使用`CREATE FUNCTION`语句注册用户定义的函数，它们可以按在[在Greenplum数据库中使用函数](#)中所述的方式被使用。默认情况下，用户定义的函数被声明为`VOLATILE`，因此如果用户定义的函数是`IMMUTABLE`或者`STABLE`，必须在注册该函数时指定正确的易变性级别。

默认情况下，用户定义的函数被声明为`EXECUTE ON ANY`。只有当函数在master实例上执行时，才支持执行查询以访问表的函数，除非函数可以执行仅访问segment实例上复制表的`SELECT`命令。访问散列分布表或随机分布表的函数必须用`EXECUTE ON MASTER`属性定义。否则，在复杂的查询中使用函数时，函数可能返回不正确的结果。如果没有该属性，规划器优化可能会将函数调用推送到segment实例中执行。

在创建用户定义的函数时，避免使用致命错误或者破坏性的调用。Greenplum数据库可能会用突然的关闭或者重启来应对这些错误。

在Greenplum数据库中，用户定义的函数的共享库必须位于Greenplum数据库阵列的所有主机（Master、Segment以及镜像）上的同一库路径地址。

还可以创建并执行用Greenplum数据库过程语言（例如PL/pgSQL）编写的匿名代码块。匿名块作为短暂的匿名函数运行。有关创建和执行匿名块的信息，请见[DO命令](#)。

内建函数和操作符

下面的表格列出了PostgreSQL支持的内建函数和操作符的分类。Greenplum数据库中支持PostgreSQL中所有的函数和操作符，除了STABLE以及VOLATILE函数之外，这两种服从在Greenplum数据库中使用函数中记录的限制，在[Greenplum数据库中使用函数](#)。更多有关这些内建函数和操作符的信息请见PostgreSQL文档中的[函数和操作符](#)。

Greenplum数据库包括操作json数据类型值的JSON处理函数。有关JSON数据的信息，请见使用JSON数据[使用JSON数据](#)。

Table 3. 内建函数和操作符

操作符/函数分类	VOLATILE函数	STABLE函数	限制
逻辑操作符			
比较操作符			
数学函数和操作符	random setseed		
字符串函数和操作符	所有内置转换操作符	convert pg_client_encoding	
二进制串函数和操作符			
位串函数和操作符			
模式匹配			
数据类型格式化函数		to_char to_timestamp	
日期/时间函数和操作符	timeofday	age current_date current_time current_timestamp localtime localtimestamp now	
枚举支持函数			

几何函数和操作符			
网络地址函数和操作符			
序列操纵函数	nextval() setval()		
条件表达式			
数组函数和操作符		所有数组函数	
聚集函数			
子查询表达式			
行及数组比较			
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 所有系统目录信息函数 所有注释信息函数 所有事务ID和快照	
系统管理函数	set_config pg_cancel_backend pg_terminate_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 所有数据库对象大小函数	Note: pg_column_size 函数显示存储该值所需的字节，可能使用toast压缩。
XML函		cursor_to_xml(cursor refcursor, count int,	

数 □ 和类似函数的表达式	<pre>nulls boolean, tableforest boolean, targetns text)</pre> <pre>cursor_to_xmlschema(cursor refcursor, nulls boolean, tableforest boolean, targetns text)</pre> <pre>database_to_xml(nulls boolean, tableforest boolean, targetns text)</pre> <pre>database_to_xmlschema(nulls boolean, tableforest boolean, targetns text)</pre> <pre>database_to_xml_and_xmlschema(nulls boolean, tableforest boolean, targetns text)</pre> <pre>query_to_xml(query text, nulls boolean, tableforest boolean, targetns text)</pre> <pre>query_to_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)</pre> <pre>query_to_xml_and_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)</pre> <pre>schema_to_xml(schema name, nulls boolean, tableforest boolean, targetns text)</pre> <pre>schema_to_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)</pre> <pre>schema_to_xml_and_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)</pre> <pre>table_to_xml(tbl regclass, nulls boolean, tableforest boolean, targetns text)</pre> <pre>table_to_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text)</pre> <pre>table_to_xml_and_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text)</pre> <pre>xmlagg(xml)</pre> <pre>xmlconcat(xml[, ...])</pre> <pre>xmlelement(name name [, xmlattributes(value [AS attname] [, ...]) [, content, ...]])</pre> <pre>xmlexists(text, xml)</pre> <pre>xmlforest(content [AS name] [, ...])</pre> <pre>xml_is_well_formed(text)</pre> <pre>xml_is_well_formed_document(text)</pre> <pre>xml_is_well_formed_content(text)</pre> <pre>xmlparse ({ DOCUMENT CONTENT } value)</pre> <pre>xpath(text, xml)</pre>
-------------------------------	---

	xpath(text, xml, text[])
	xpath_exists(text, xml)
	xpath_exists(text, xml, text[])
	xmlpi(name target [, content])
	xmlroot(xml, version text no value [, standalone yes no no value])
	xmlserialize ({ DOCUMENT CONTENT } value AS type)
	xml(text)
	text(xml)
	xmlcomment(xml)
	xmlconcat2(xml, xml)

窗口函数

下列内建窗口函数是Greenplum对于PostgreSQL数据库的扩展。所有的窗口函数都是`immutable`。更多有关窗口函数的信息，请见[窗口表达式](#)。

Table 4. 窗口函数

函数	返回类型	完整语法	描述
<code>cume_dist()</code>	<code>double precision</code>	<code>CUME_DIST() OVER ([PARTITION BY expr] ORDER BY expr)</code>	计算一组值中一个值的累积分布。具有相等值的行总是具有相同的累积分布值。。
<code>dense_rank()</code>	<code>bigint</code>	<code>DENSE_RANK () OVER ([PARTITION BY expr] ORDER BY expr)</code>	计算一个有序行组中一行的无跳跃排名值的排名。具有相等值的行会得到相同的排名值。
<code>first_value(expr)</code>	<code>same as input expr type</code>	<code>FIRST_VALUE(expr) OVER ([PARTITION BY expr] ORDER BY expr [ROWS RANGE frame_expr])</code>	返回一个有续值集合中的第一个值。
<code>lag(expr [,offset] [,default])</code>	<code>same as input expr type</code>	<code>LAG(expr [, offset] [, default]) OVER ([PARTITION BY expr] ORDER BY expr)</code>	在不做自连接的情况下，提供对于同一个表中多于一行的访问。给定一个查询返回的一系列行以及该游标的一个位置，LAG提供对位于该位置之前一个给定物理偏移量的行的访问。默认 offset 1

			的为。 default设置当偏移量超出窗口范围之外时要返回的值。如果没有指定default, 默认值是空值。
last_value(expr)	same as input expr type	LAST_VALUE(expr) OVER ([PARTITION BY expr] ORDER BY expr [ROWS RANGE frame_expr])	返回一个有序值 集合中的最后一个 值。
lead(expr [,offset] [,default])	same as input expr type	LEAD(expr [,offset] [,exprdefault]) OVER ([PARTITION BY expr] ORDER BY expr)	在不做自连接的情 况下, 提供对 于同一个表中多 于一行的访问。 给定一个查询返 回的一系列行以 及该游标的一个 位置, lead提供对 位于该位置之后 一个给定物理偏 移量的行的访 问。如果没有指 定offset, 默认偏 移量 是1。default设置 当偏移量超出窗 口范围之外时要 返回的值。如果 没有指 定default, 默认 值是空值。
ntile(expr)	bigint	NTILE(expr) OVER ([PARTITION BY expr] ORDER BY expr)	把一个有序数据 集划分成一些桶 (由expr定义) 并且为每一行分 配一个桶号。
percent_rank()	double precision	PERCENT_RANK () OVER ([PARTITION BY expr] ORDER BY expr)	计算一个假设 行R的排名减1, 然后除以被计算 的行数(在一个 窗口分区)减1。
rank()	bigint	RANK () OVER ([PARTITION BY expr] ORDER BY expr)	计算一行在一个 有序值组中的排 名。根据排名标 准有相等值的行 得到相同的排 名。被占用的行 数被加到排名数 上来计算下一个 排名值。在这样 情况下, 排名可 能不是连续的数 字。
row_number()	bigint	ROW_NUMBER () OVER ([PARTITION BY expr] ORDER BY expr)	为窗口分区中的 每一行或者查询 中的每一行分配 一个唯一的编 号。

高级聚集函数

下列内建高级聚集函数是Greenplum对PostgreSQL数据库的扩展。这些函数都是`immutable`。Greenplum数据库不支持PostgreSQL有序集聚合函数。

Note: 用于分析的Greenplum MADlib扩展提供了额外的高级函数来执行对Greenplum数据库数据的统计分析和机器学习。请见 [Greenplum数据库参考指南中的“Greenplum的MADlib分析扩展”](#)。

Table 5. 高级聚集函数

函数	返回类型	完整语法	描述
<code>MEDIAN(expr)</code>	<code>timestamp</code> , <code>timestamptz</code> , <code>interval</code> , <code>float</code>	<code>MEDIAN(expression)</code> 例子： <pre>SELECT department_id, MEDIAN(salary) FROM employees GROUP BY department_id;</pre>	可以用一个二维数组作为输入。把这些数组当作矩阵。
<code>sum(array[])</code>	<code>smallint[]</code> <code>int[]</code> , <code>bigint[]</code> , <code>float[]</code>	<code>sum(array[[1,2],[3,4]])</code> 例子： <pre>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}}</pre>	执行矩阵求和。可以将被视为矩阵的二维数组作为输入。
<code>pivot_sum(label[], label, expr)</code>	<code>int[]</code> , <code>bigint[]</code> , <code>float[]</code>	<code>pivot_sum(array['A1','A2'], attr, value)</code>	使用 <code>sum</code> 来解决重複项的pivot聚集。
<code>unnest(array[])</code>	<code>anyelement</code> 集合	<code>unnest(array['one', 'row', 'per', 'item'])</code>	把一个一维数组转换成行。返回一个 <code>anyelement</code> （一种多态伪类型，请见PostgreSQL中的伪类型）集合。 pseudo-type

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

关于Greenplum的查询处理

□ 关于GPORCA

定义查询

WITH查询 (公用表表达式)

使用函数和操作符

使用JSON数据

通过XML数据

Greenplum数据库支持json和jsonb数据类型来存储JSON (JavaScript Object Notation)数据。

Greenplum数据库按照文档[RFC 7159](#) 支持JSON, 并且根据JSON规则强制数据有效性。也有一些用于 json 和 jsonb 数据类型的JSON相关的函数和操作符。参见[JSON函数和操作符](#).

这一节包含下列主题:

- [关于JSON数据](#)
- [JSON输入和输出语法](#)
- [设计JSON文档](#)
- [jsonb容器与存在](#)
- [jsonb索引](#)
- [JSON函数和操作符](#)

Parent topic: [查询数据](#)

关于JSON数据

Greenplum数据库支持两种JSON数据类型: json and jsonb. 他们输入值基本一致, 主要区别之一是效率。

- The json数据类型存储输入副本. 这要求JSON处理函数每次执行时处理json 数据。json数据类型不会修改输入文本。
 - 记号之间没有语义意义的空格被保留, JSON对象内的键的顺序也被保留。
 - 即便JSON对象含有重复键, 所有的键/值对也会被保留。对于重复键, JSON处理函数把其中的最后一个值当作有效值。
- jsonb数据类型存储输入文本的二进制格式。输入数据的转换开销使jsonb稍慢于 json数据类型. 但是, json处理函数明显更快, 因为解析jsonb数据不是必须的。 jsonb数据类型修改输入文本。
 - 空格不保留
 - 对象内的键的顺序不保留
 - 重复键对象不保留, 如果输入包含重复键, 仅保留最后一个值。

jsonb数据类型支持索引。参考 [jsonb索引](#).

一般来说, JSON数据应该存储为jsonb数据类型, 除非特殊需要, 例如关于对象键顺序的遗留假设。

关于JSON数据中的Unicode字符

The [RFC 7159](#) 文档允许JSON字符串包含表示为\uXXXX的\uXXXXUnicode转义序列。不过, Greenplum对每一个数据库只允许一种字符集编码。除非数据库编码为UTF8, json类型不可能严格地符合JSON说明。尝试包括数据库编码中无法表示的字符将会失败。允许可在数据库编码中表示但不在UTF8中的字符。

- json数据类型的greenplum数据库输入函数允许unicode转义, 而无论是什么数据库编码, 只检查unicode转义的语法正确性 (a \u后跟四个十六进制数字) 。

- jsonb数据类型的greenplum数据库输入函数更严格。它不允许非ASCII字符（U+007F以上的字符）使用Unicode转义，除非数据库编码是UTF8。它还拒绝\u0000，它不能在greenplum数据库text类型中表示，并且它要求使用unicode代理项对指定unicode基本多语言平面之外的字符是正确的。有效的Unicode转义符（除了\u0000）转换为等效的ASCII或UTF8字符进行存储；这包括将代理项对折叠为单个字符。

Note: [JSON函数和运算符](#)中描述的许多JSON处理函数将Unicode转义转换为常规字符。函数对数据库编码中无法表示的字符抛出错误。如果可能的话，应该避免将JSON中的Unicode转义与非UTF8数据库编码混合在一起。

将JSON数据类型映射到Greenplum数据类型

When converting JSON text input into jsonb data, the primitive data types described by RFC 7159 are effectively mapped onto native Greenplum Database data types, as shown in the following table.

Table 1. 将JSON数据类型映射到Greenplum数据类型

JSON原始数据类型	Greenplum数据库数据类型	注意
string	text	不允许使用\u0000。只有当数据库编码为utf8时，才允许使用非ASCII Unicode转义。
number	numeric	不允许使用NaN和infinity值
boolean	boolean	只接受小写的正误拼写
null	(none)	The JSON null 原始类型不同于SQL NULL。

对于有效的jsonb 数据的构成有一些次要的约束，这些约束既不适用于json 数据类型，也不适用于抽象的JSON，对应于基础数据类型所能表示的限制。值得注意的是，当将数据转换为jsonb数据类型时，超出Greenplum数据库numeric数据类型范围的数字将被拒绝，而json数据类型不会拒绝这些数字。

RFC 7159允许这种实施定义的限制。然而，在实践中，这些问题可能会出现在其他实现中，因为通常将JSON原始类型 number表示为IEEE 754双精度浮点（RFC7159明确预测并允许）。

当使用JSON作为与其他系统的交换格式时，请注意与Greenplum数据库最初存储的数据相比，数字精度可能会降低。

另外，正如上表中所指出的，对于JSON原语类型的输入格式，存在一些小的限制，这些限制不适用于相应的Greenplum数据库数据类型。

JSON输入和输出语法

json数据类型的输入和输出语法如RFC 7159中所述。

下列都是合法的json表达式：

```
-- 简单标量/原始值
-- 原始值可以是数字、带引号的字符串、true、false或者null SELECT '5'::json;

-- 零个或者更多个元素的数组（元素类型可以不同）
SELECT '[1, 2, "foo", null]'::json;

-- 含有键/值对的对象
-- 注意对象的键必须总是带引号的字符串
```

```
SELECT '{"bar": "baz", "balance": 7.77, "active": false}':json;
-- 数组和对象可以任意嵌套
SELECT '{"foo": [true, "bar"], "tags": { "a": 1, "b": null}}':json;
```

如前所述，当输入JSON值，然后在不进行任何附加处理的情况下打印时，`json`数据类型输出的文本与输入的文本相同，而`jsonb`数据类型不保留语义上不重要的细节，如空白。例如，请注意这里的区别：

```
SELECT '{"bar": "baz", "balance": 7.77, "active":false}':json;
-----  
{"bar": "baz", "balance": 7.77, "active":false}  
(1 row)

SELECT '{"bar": "baz", "balance": 7.77, "active":false}':jsonb;
-----  
{"bar": "baz", "active": false, "balance": 7.77}  
(1 row)
```

值得注意的一个语义上不重要的细节是，对于`jsonb`数据类型，将根据基础数字类型的行为打印数字。在实践中，这意味着使用`e`符号输入的数字将不使用`e`符号打印，例如：

```
SELECT '{"reading": 1.230e-5}':json, '{"reading": 1.230e-5}':jsonb;
-----+  
{"reading": 1.230e-5} | {"reading": 0.00001230}  
(1 row)
```

然而，`jsonb`数据类型保留了尾随的小数零，如前一个示例中所示，即使这些小数零在语义上对于诸如相等性检查之类的是无关紧要的。

设计JSON文档

将数据表示为JSON比传统关系数据模型要更灵活，在需求变化的环境中表现得尤其明显。在同一个应用中很可能两种方法会共存并且成为彼此的互补。不过，即便是对于要求最大灵活性的应用中，我们仍然推荐JSON有些许固定的结构。这种结构是非强制的（尽管可能会强制一些业务规则），但是如果可预测的结构会让编写有效汇总表中一组“文档”（数据）的查询更容易。

在表中存储时，JSON数据服从和其他任何数据类型一样的并发控制考虑。尽管存储大型文档格式可行的，但要记住任何更新都要求整个行上的一个行级锁。为了降低更新事务之间的锁竞争，请考虑限制JSON文档为一个可管理的尺寸。理想上，每个JSON文档应该表示业务规则规定的一个原子数据，并且不能进一步地被分解为更小的可以独立修改的数据。

jsonb容器与存在

测试容器是`jsonb`的一项重要功能。`json`类型没有并行的设施集。容器测试一个`jsonb`文档中是否包含了另一个`jsonb`文档。这些例子返回`true`，除非另有标注：

```
--简单标量/原始数值仅包含相同的值:  
SELECT ""foo"":jsonb @> ""foo"":jsonb;
```

```
-- 右边的数组包含在左边的数组中:
SELECT '[1, 2, 3]':jsonb @> '[1, 3]':jsonb;

-- 数组元素的顺序并不重要，因此这也是正确的:
SELECT '[1, 2, 3]':jsonb @> '[3, 1]':jsonb;

-- 重复的数组元素也不重要:
SELECT '[1, 2, 3]':jsonb @> '[1, 2, 2]':jsonb;

-- 右侧只有一对的对象包含在左侧的对象中:
SELECT '{"product": "Greenplum", "version": "6.0.0", "jsonb":true}':jsonb @> '{"version": "6.0.0"}':jsonb;

-- 右侧的数组被认为不包含在左侧的数组中，即使其中嵌套了类似的数组:
SELECT '[1, 2, [1, 3]]':jsonb @> '[1, 3]':jsonb; -- yields false

-- 但是，通过一层嵌套，它包含:
SELECT '[1, 2, [1, 3]]':jsonb @> '[[1, 3]]':jsonb;

-- 同样，此处未报告遏制:
SELECT '{"foo": {"bar": "baz", "zig": "zag"}}':jsonb @> '{"bar": "baz"}':jsonb; -- yields false

-- 但是，通过一层嵌套，它包含:
SELECT '{"foo": {"bar": "baz", "zig": "zag"}}':jsonb @> '{"foo": {"bar": "baz"}}':jsonb;
```

一般原则是，所包含的对象必须在结构和数据内容方面与包含的对象匹配，可能是在从包含的对象中丢弃一些不匹配的数组元素或对象键/值对之后。对于容器，在进行容器匹配时，数组元素的顺序并不重要，重复的数组元素只被有效地考虑一次。

作为结构必须匹配的一般原则的例外，数组可以包含原始值：

```
-- 此数组包含原始字符串值:
SELECT ["foo", "bar"]':jsonb @> ""bar"":jsonb;

-- 这个异常不是相互的——这里报告了非包容:
SELECT ""bar"":jsonb @> ["bar"]':jsonb; -- yields false
```

`jsonb` 还有一个`existence`操作符，它是容器主题的变体：它测试字符串（作为文本值给出）是否作为对象键或数组元素出现在`jsonb`值的顶层。这些示例返回`true`，，除非另有标注：

```
-- 字符串作为数组元素存在:
SELECT ["foo", "bar", "baz"]':jsonb ? 'bar';

-- 字符串作为对象键存在:
SELECT '{"foo": "bar"}':jsonb ? 'foo';

-- 不考虑对象值:
SELECT '{"foo": "bar"}':jsonb ? 'bar'; -- yields false

-- 与容器一样，存在必须在顶层匹配:
SELECT '{"foo": {"bar": "baz"} }':jsonb ? 'bar'; -- yields false

-- 如果字符串与原始JSON字符串匹配，则认为该字符串存在:
SELECT ""foo"":jsonb ? 'foo';
```

当涉及到许多键或元素时，JSON对象比数组更适合测试包含性或存在性，因为与数组不同，JSON对象在内部针对搜索进行了优化，不需要进行线性搜索。

各种容器和存在操作符以及所有其他JSON操作符和函数都记录在JSON函数和操作符[JSON函数和操作符](#)中。

因为JSON容器是嵌套的，所以适当的查询可以跳过子对象的显式选择。例如，假设我们有一个包含顶级对象的doc列，其中大多数对象包含子对象数组的标记字段。此查询查找包含“term”：“paris”和“term”：“food”的子对象出现的条目，同时忽略标记数组之外的任何此类键：

```
SELECT doc->'site_name' FROM websites
WHERE doc @> '{"tags": [{"term": "paris"}, {"term": "food"}]}';
```

使用这个谓词的查询可以完成相同的事情。

```
SELECT doc->'site_name' FROM websites
WHERE doc->'tags' @> [{"term": "paris"}, {"term": "food"}];
```

然而，第二种方法的灵活性较低，而且效率通常也较低。

另一方面，json存在操作符不是嵌套的：它只在json值的顶层查找指定的键或数组元素。

jsonb索引

Greenplum数据库jsonb数据类型，支持GIN, btree, 和hash索引。

- [jsonb数据上的GIN索引](#)
- [jsonb数据上的树索引和哈希索引](#)

jsonb数据上的GIN索引

可以使用GIN索引有效地搜索出现在大量jsonb文档（基准）中的键或键/值对。两个GIN操作符类，提供不同的性能和灵活性权衡。

*jsonb*的默认GIN操作符类支持带@>, ?, ?& 和 ?|操作符的查询。（有关这些运算符实现的语义的详细信息，请参见表[Table 3](#)）。使用此操作符类创建索引的示例如下：

```
CREATE INDEX idxgin ON api USING gin (jdoc);
```

非默认的GIN运算符类*jsonb_path_ops*仅支持为@>运算符编制索引。使用此运算符类创建索引的示例如下：

```
CREATE INDEX idxginp ON api USING gin (jdoc jsonb_path_ops);
```

考虑一个表的示例，该表存储从第三方Web服务检索到的JSON文档，并且具有文档化的模式定义。这是一个典型的文档：

```
{
  "guid": "9c36adc1-7fb5-4d5b-83b4-90356a46061a",
  "name": "Angela Barton",
  "is_active": true,
  "company": "MagnaFone",
  "address": "178 Howard Place, Gulf, Washington, 702",
```

```

"registered": "2009-11-07T08:53:22 +08:00",
"latitude": 19.793713,
"longitude": 86.513373,
"tags": [
    "enim",
    "aliquip",
    "qui"
]
}

```

JSON文档存储在jsonb列中的一个名为API的表中。如果在此列上创建了GIN索引，则以下查询可以使用该索引：

```
-- 查找关键词“公司”具有“magnafone”价值的文档:
SELECT jdoc->'guid', jdoc->'name' FROM api WHERE jdoc @> '{"company": "Magnafone"}';
```

但是，索引不能用于以下查询。操作符?是可索引的，但是，比较不会直接应用于索引列jdoc：

```
-- Find documents in which the key "tags" contains key or array element "qui"
SELECT jdoc->'guid', jdoc->'name' FROM api WHERE jdoc -> 'tags' ? 'qui';
```

通过适当地使用表达式索引，上述查询可以使用索引。如果在tags键中查询特定项是常见的，那么定义这样的索引可能是值得的：

```
CREATE INDEX idxgintags ON api USING gin ((jdoc -> 'tags'));
```

现在，WHERE子句 jdoc -> 'tags' ? 'qui' 被认为是可索引运算符的应用？到索引表达式 jdoc -> 'tags'。有关表达式索引的信息，请参阅表达式索引 [表达式索引](#)。

查询JSON文档的另一种方法是利用包含性，例如：

```
-- 查找键“tags”包含数组元素“qui”的文档
SELECT jdoc->'guid', jdoc->'name' FROM api WHERE jdoc @> '{"tags": ["qui"]}';
```

jdoc列上的一个简单的GIN索引可以支持这个查询。但是，索引将在jdoc列中存储每个键和值的副本，而前一个示例的表达式索引只存储标记键下的数据。虽然简单索引方法要灵活得多（因为它支持关于任何键的查询），但是目标表达式索引可能比简单索引更小，搜索速度更快。

尽管 jsonb_path_ops类只支持带@> 运算符的查询，但它比默认的jsonb_operator类具有性能优势。对于相同的数据， jsonb_path_ops索引通常比jsonb_ops索引小得多，搜索的特殊性更好，尤其是当查询包含频繁出现在数据中的键时。因此，搜索操作通常比默认的operator类执行得更好。

jsonb_ops 和 jsonb-path-ops-gin 索引的技术区别在于前者为数据中的每个键和值创建独立的索引项，而后者仅为数据中的每个值创建索引项。

Note: 对于这个讨论，术语 *value* 包括数组元素，尽管JSON术语有时考虑数组元素不同于对象中的值。

基本上，每个jsonb_path_ops索引项都是值的散列和指向该值的键；例如，要索引{"foo": {"bar": "baz"} }，将创建一个索引项，将foo, bar, 和baz 中的三个都合并到散列值中。因此，查找此结构的包含查询将导致极其具体的索引搜索；但根本无法确定foo是否显示为键。另一方面，jsonb_ops 索引将分别创建三个表示foo, bar, 和baz 的索引项；然后，为了执行包含性查询，它将查找包含这三个项的行。虽然GIN索引可以相当有效地执行这样的搜索，但是它仍然比同等的jsonb_path_ops搜索更不具体和更慢，特别是如果有大量的行包含三个索引项中的任何一个。

`jsonb_path_ops`方法的一个缺点是它不会为不包含任何值的JSON结构生成索引项，例如`{"a":{}}`。如果请求搜索包含此类结构的文档，则需要进行完全索引扫描，这非常慢。`jsonb_path_ops`不适合经常执行这种搜索的应用程序。

jsonb数据上的树索引和哈希索引

jsonb还支持btree和hash索引。只有在检查完整JSON文档的相等性很重要时，这些方法才有用。

为完整起见，`jsonb`基准的btree排序是：

Object > Array > Boolean > Number > String > Null

Object with n pairs > object with n - 1 pairs

Array with n elements > array with n - 1 elements

相等键值数的对象按以下顺序进行比较：

key-1, value-1, key-2 ...

对象键按存储顺序进行比较。特别是，由于较短的键存储在较长的键之前，这可能导致顺序不直观，例如：

{ "aa": 1, "c": 1 } > { "b": 1, "d": 1 }

同样，元素数目相等的数组按以下顺序进行比较：

element-1, element-2 ...

使用与底层Greenplum数据库数据类型相同的比较规则来比较原始JSON值。使用默认数据库排序规则比较字符串。

JSON函数和操作符

Greenplum数据库包含创建和操作JSON数据的内置函数和运算符。

- [JSON操作符](#)
- [JSON创建函数](#)
- [JSON聚合函数](#)
- [JSON处理函数](#)

Note: 对于`json`数据类型值，即使JSON对象包含重复的键，也会保留所有键/值对。对于重复的键，JSON处理函数将最后一个值视为可操作的值。对于`JSONB`数据类型，不保留重复的对象键。如果输入包含重复键，则只保留最后一个值。请参见[关于JSON数据](#)。

JSON操作符

这个表格描述了可以用于 json 和 jsonb 数据类型的操作符。

Table 2. json 和 jsonb 操作符

操作符	右操作数类型	描述	例子	例子结果
->	int	获得JSON数组元素（索引从零开始）。	'[{"a": "foo"}, {"b": "bar"}, {"c": "baz"}] :: json->2	{"c": "baz"}
->	text	根据键获得JSON对象的域。	'{"a": {"b": "foo"}}' :: json->'a'	{"b": "foo"}
->>	int	获得JSON数组元素的text形式。	'[1, 2, 3]' :: json->>2	3
->>	text	获得JSON对象域的text形式。	'{"a":1, "b":2}' :: json->>'b'	2
#>	text[]	获得在指定路径上的JSON对象。	'{"a": {"b": {"c": "foo"}}}' :: json#>'{a,b}'	{"c": "foo"}
#>>	text[]	获得在指定路径上的JSON对象的text形式。	'{"a": [1, 2, 3], "b": [4, 5, 6]}' :: json#>>'{a,2}'	3

Note: 对于 json 和 jsonb 数据类型，这些运算符都有并行变体。字段、元素和路径提取运算符返回的数据类型与其左侧输入（json 和 jsonb）相同，但指定为返回 text 的数据类型除外，后者将值强制为 text。如果 JSON 输入没有与请求匹配的正确结构，则字段、元素和路径提取运算符返回 NULL，而不是失败；例如，如果不存在这样的元素。

下表描述了需要 jsonb 数据类型作为左操作数的运算符。这些运算符中的许多都可以通过 jsonb 运算符类进行索引。有关 jsonb 包含和存在语义的完整描述，请参见 [jsonb 容器与存在](#)。有关如何使用这些运算符有效地索引 JSONB 数据的信息，请参阅 [jsonb 索引](#)。

Table 3. jsonb 操作符

操作符	右操作数类型	描述	例子
@>	jsonb	左边的 json 值包含右边的值吗？	'{"a":1, "b":2}' :: jsonb @> '{"b":2}' :: jsonb
<@	jsonb	左 JSON 值是否包含在右值中？	'{"b":2}' :: jsonb <@ '{"a":1, "b":2}' :: jsonb
?	text	键/元素字符串是否存在 JSON 值中？	'{"a":1, "b":2}' :: jsonb ? 'b'
?	text[]	是否存在这些键/元素字符串？	'{"a":1, "b":2, "c":3}' :: jsonb ? array['b', 'c']
?&	text[]	所有这些键/元素字符串都存在吗？	'["a", "b"]' :: jsonb ?& array['a', 'b']

下表中的标准比较运算符仅适用于 jsonb 数据类型，而不适用于 json 数据类型。它们遵循 jsonb 索引 [jsonb 索引](#) 中描述的 B 树操作的排序规则。

Table 4. jsonb 比较运算符

操作符	描述
<	小于

>	大于
<=	小于或等于
>=	大于或等于
=	等于
<>或者 !=	不等于

Note: 这个!=运算符在解析阶段转换为<>。无法执行!=和<>来执行不同操作的运算符。

JSON创建函数

此表描述了创建 json 数据类型值的函数。（目前， jsonb 没有等价的函数，但是您可以将其中一个函数的结果强制转换为 jsonb。）

Table 5. JSON 创建函数

函数	描述	例子	例子结果
<code>to_json(anyelement)</code>	返回该值作为一个合法的 JSON 对象。数组和组合会被递归处理并且转换成数组和对象。如果输入包含一个从该类型到 json 的转换，会使用该 cast 函数来执行转换，否则将会产生一个 JSON 标量值。对于任何非数字、布尔值或空值的标量类型，会使用其文本表示，并且加上适当的引号和转义让它变成一个合法的 JSON 字符串。	<code>to_json('Fred said "Hi."')::text</code>	"Fred said \"Hi.\""
<code>array_to_json(anyarray [, pretty_bool])</code>	返回该数组为一个 JSON 数组。一个 Greenplum 数据库多维数组会变成一个 JSON 数组的数组。 如果 <code>pretty_bool</code> 为 <code>true</code> ，在第一维元素之间会增加换行。	<code>array_to_json('{{1,5},{99,100}}')::int[]</code>	[[1,5],[99,100]]
<code>row_to_json(record [, pretty_bool])</code>	返回该行为一个 JSON 对象。 如果 <code>pretty_bool</code> 为 <code>true</code> ，在第一级别元素之间会增加换行。	<code>row_to_json(row(1,'foo'))</code>	{"f1":1,"f2":"foo"}
<code>json_build_array(VARIADIC</code>	从 VARIADIC 参数列表构建	<code>json_build_array(1,2,'3',4,5)</code>	[1, 2, "3", 4, 5]

"any")	一个可能是异种类型的JSON数组。		
json_build_object(VARIADIC "any")	从VARIADIC参数列表构建JSON对象。参数列表按顺序获取，并转换为一组键/值对。	json_build_object('foo',1,'bar',2)	{"foo": 1, "bar": 2}
json_object(text[])	从文本数组构建JSON对象。数组必须是一维或二维数组。 一维数组必须有偶数个元素。元素作为键/值对。 对于二维数组，每个内部数组必须恰好有两个元素，作为键/值对。	json_object('{a, 1, b, "def", c, 3.5}') json_object('{{a, 1}, {b, "def"}, {c, 3.5}}')	{"a": "1", "b": "def", "c": "3.5"}
json_object(keys text[], values text[])	从文本阵列中创建JSON对象。这一形式的json_object对象从两个分隔阵列中选择了键和值对。在所有其他方面，它与一个论据形式相同。	json_object('{a, b}', '{1,2}')	{"a": "1", "b": "2"}

Note: `array_to_json`和`row_to_json`的行为与`to_json`相同，只是提供了一个打印漂亮的选项。`to_json`描述的行为同样适用于其他JSON创建函数转换的每个单独的值。

Note: `hstore` `hstore`模块包含从`hstore`转换为`json`的函数，因此通过`json`创建函数转换的`hstore`值将被表示为`json`对象，而不是原始字符串值。

JSON聚合函数

此表显示函数将记录聚合到一个JSON对象数组和一对JSON对象的值。

Table 6. JSON聚合函数

函数	参数类型	返回类型	描述
<code>json_agg(record)</code>	<code>record</code>	<code>json</code>	将记录聚合为对象的JSON数组。
<code>json_object_agg(name, value)</code>	("any", "any")	<code>json</code>	将名称/值对聚合为JSON对象。

JSON处理函数

这个表描述处理`json` 和 `jsonb`值的函数。

许多处理函数和运算符将JSON字符串中的Unicode转义符转换为适当的单个字符。如果输入数据类型是`jsonb`，这不是问题，因为转换已经完成。但是，对于`json`数据类型输入，这可能导致抛出错误。请参见关于 [JSON](#)。

Table 7. JSON处理函数

函数	返回类型	描述	例子	例子结果
json_array_length(json) jsonb_array_length(jsonb)	int	返回最外层JSON数组中的元素数。	json_array_length('[1,2,3,{"f1":1,"f2":[5,6]},4]')	5
json_each(json) jsonb_each(jsonb)	setof key text, value json setof key text, value jsonb	将最外层的JSON对象展开为一组键/值对。	select * from json_each('{"a":"foo", "b":"bar"}')	<pre>key value -----+----- a "foo" b "bar"</pre>
json_each_text(json) jsonb_each_text(jsonb)	setof key text, value text	把最外层的JSON对象展开成键/值对的集合。返回值的类型是text。	select * from json_each_text('{"a":"foo", "b":"bar"}')	<pre>key value -----+----- a foo b bar</pre>
json_extract_path(from_json json, VARIADIC path_elems text[]) jsonb_extract_path(from_json jsonb, VARIADIC path_elems text[])	json jsonb	返回path_elems指定的JSON值。等效于#>操作符。	json_extract_path('{"f2":{"f3":1}, "f4": {"f5":99, "f6":"foo"} }', 'f4')	{"f5":99, "f6":"foo"}
json_extract_path_text(from_json json, VARIADIC path_elems text[]) jsonb_extract_path_text(from_json jsonb, VARIADIC path_elems text[])	text	以文本形式返回path_elems指向的JSON值。相当于>>运算符。	json_extract_path_text('{"f2": {"f3":1}, "f4": {"f5":99, "f6":"foo"} }', 'f4', 'f6')	foo
json_object_keys(json) jsonb_object_keys(jsonb)	setof text	返回最外层JSON对象中的键集合。	json_object_keys('{"f1":"abc", "f2": {"f3":"a", "f4":"b"} }')	<pre>json_object_keys ----- f1 f2</pre>
json_populate_record(base anyelement, from_json json) jsonb_populate_record(base	anyelement	把from_json的对象展开成一行，其中的列匹配由base定义的记录	select * from json_populate_record(null::myrowtype, '{"a":1, "b":2}')	<pre>a b ---+--- 1 2</pre>

anyelement, from_json jsonb)		类型。请见注解 Note 1 。		
json_populate_recordset(base anyelement, from_json json) jsonb_populate_recordset(base anyelement, from_json jsonb)	setof anyelement	将from_json的最外层数组从\json扩展到一组行，这些行的列与基定义的记录类型匹配。见 Note 1 。	select * from json_populate_recordset(null::myrowtype, '[{"a":1,"b":2}, {"a":3,"b":4}]')	a b ---+--- 1 2 3 4
json_array_elements(json) jsonb_array_elements(jsonb)	setof json setof jsonb	Expands a JSON array to a set of JSON values.	select * from json_array_elements('[1,true, [2,false]]')	value ----- 1 true [2,false]
json_array_elements_text(json) jsonb_array_elements_text(jsonb)	setof text	Expands a JSON array to a set of text values.	select * from json_array_elements_text('["foo", "bar"]')	value ----- foo bar
json_typeof(json) jsonb_typeof(jsonb)	text	以文本字符串形式返回最外面的JSON值的类型。可能的类型包括object, array, string, number, boolean, 和 null. See Note 2 .	json_typeof('-123.4')	number
json_to_record(json) jsonb_to_record(jsonb)	record	从JSON对象生成任意记录，可见 Note 1 。 与所有返回记录的函数一样，调用方必须使用AS子句显式定义记录的结构。	select * from json_to_record('{"a":1,"b": [1,2,3],"c":"bar"}') as x(a int, b text, d text)	a b d ---+---+--- 1 [1,2,3]
json_to_recordset(json) jsonb_to_recordset(jsonb)	setof record	从JSON对象数组生成任意记录集，可见 Note 1 。 与所有返回记录的函数一样，调用方必须使用AS	select * from json_to_recordset('[{"a":1,"b":"foo"}, {"a":2,"c":"bar"}]') as x(a int, b text);	a b ---+--- 1 foo 2

clause子句显式定义记录的结构。

Note:

1. 函数`json_populate_record()`, `json_populate_recordset()`, `json_to_record()` 和 `json_to_recordset()` 的示例使用常量。但是，典型的用法是引用`FROM`子句中的表，并使用它的`json`或`jsonb`列之一作为函数的参数。然后可以在查询的其他部分中引用提取的键值。例如，可以在`WHERE`子句和目标列表中引用该值。以这种方式提取多个值可以提高性能，而不是使用每个键操作符分别提取多个值。JSON键与目标行类型中相同的列名匹配。这些函数的JSON类型强制可能不会产生某些类型所需的值。目标行类型中未出现的JSON字段将从输出中省略，不匹配任何JSON字段的目标列将为空。
2. `json_typeof`函数NULL返回值`null`不应与`SQLNULL`混淆。当调用`json_typeof('null'::json)`将返回`null`时，调用`json_typeof(NULL::json)`将返回`SQLNULL`。

Greenplum数据库® 6.0文档

□ 管理员指南

Greenplum数据库支持存储XML数据的xml数据类型。

xml数据类型会检查输入值的结构良好性，提供了在文本域中简单地存储XML数据的便利。此外，支持函数允许用户在这种数据上执行类型安全的操作，请参考下文的XML函数参考[XML函数参考](#)。

使用这种数据类型要求编译时指定`configure --with-libxml`。

xml类型可以存储结构良好（按照XML标准的定义）的“文档”，以及“内容”片段（按照XML标准中生产`XMLDecl?`内容定义）。大致上，这意味着内容片段可以有多于一个顶层元素或者字符节点。表达式`xmlvalue IS DOCUMENT`可以被用来评估一个特定的 xml值是一个完整文档或者只是一个内容片段。

此张包含以下主题：

- [创建XML值](#)
- [编码处理](#)
- [访问XML值](#)
- [处理XML](#)
- [将表映射到XML](#)
- [使用XML函数和表达式](#)

Parent topic: [查询数据](#)

创建XML值

要从字符数据产生一个xml类型的值，使用函数`xmlparse`：

```
xmlparse ( { DOCUMENT | CONTENT } value)
```

例如：

```
XMLPARSE (DOCUMENT '<?xml version="1.0"?><book>
<title>Manual</title><chapter>...</chapter></book>')
XMLPARSE (CONTENT 'abc<foo>bar</foo><bar>foo</bar>')
```

上面的方法把字符串根据SQL标准转换成XML值，但是也可以使用如下的Greenplum数据库语法：

```
xml '<foo>bar</foo>'  
'<foo>bar</foo>'::xml
```

即便输入值指定了一个文档类型声明（DTD），`xml`类型也不会按DTD验证输入值。当前也没有内建的支持来根据其他XML模式语言（例如XML模式）来进行验证。

从`xml`产生字符串值的逆操作，可以使用函数`xmlserialize`：

```
xmlserialize ( { DOCUMENT | CONTENT } value AS type )
```

`type`可以是`character`, `character varying`, 或`text`（或者这些类型的一种别名）。同样，根据SQL标准，这是唯一一种在`xml`类型和字符类型见转换的方法，但是Greenplum数据库也允许用户简单地造型该值。

当一个字符串值与类型`xml`之间的转换没有使`XMLPARSE`或者`XMLSERIALIZE`时，选择`DOCUMENT`还是`CONTENT`由`XML OPTION`会话配置参数决定，该参数可以使用标准命令设置：

```
SET XML OPTION { DOCUMENT | CONTENT };
```

或者用Greenplum数据库的风格：

```
SET XML OPTION TO { DOCUMENT | CONTENT };
```

默认值是`CONTENT`，这样所有形式的XML数据都被允许。

Note:

在设置了默认的`xml`选项后，如果字符串中含有一个文档类型声明，则用户不能直接将字符串转换成类型`xml`，因为XML内容片段的定义不接受文档类型声明。如果真的需要做这样的转换，可以使用`XMLPARSE`或者更改`XML`选项。

编码处理

在处理客户端、服务器以及流经它们的XML数据上的多种字符编码时要多加小心。在使用文本模式向服务器传递查询以及向客户端传递查询结果（这是通常的模式）时，Greenplum数据库会把转换所有在客户端和服务器之间传递的字符数据为相应端点的字符编码，请见[字符集支持](#)。这包括XML值的字符串表示，如上面的例子所示。通常，这意味着XML数据中包含的编码声明可能会变得无效，由于字符数据会在

客户端和服务器之间进行传输时被转换成其他编码，而内嵌的编码声明却不会改变。为了应对这种行为，提供给`xml`类型输入的字符串中包含的编码声明会被忽略，而内容会被假定为当前的服务器编码。随后，为了正确的处理，XML数据的字符串必须被从客户端以当前的客户端编码发出。客户端应该负责在把文档发送到服务器之前将它们转换成当前的客户端编码，或者适当地调整客户端编码。在输出上，类型`xml`的值将不会有编码声明，并且客户端应该假定所有数据都处于当前的客户端编码。

在使用二进制模式向服务器传递查询参数并且向客户端传回查询结果时，不会执行字符集转换，因此情况有所不同。在这种情况下，XML数据中的编码声明会被注意到，并且在缺少编码声明时假定数据是UTF-8（按XML标准的要求，注意Greenplum数据库不支持UTF-16）。在输出上，数据将有编码声明来指定客户端编码，除非客户端编码为UTF-8，那种情况下编码声明将被省略。

Note:

如果XML数据编码、客户端编码和服务器编码都相同，用Greenplum数据库处理XML数据会很少碰到错误并且更加高效。由于XML数据在内部是以UTF-8的形式处理，所以在服务器编码也是UTF-8时计算效率最高。

访问XML值

`xml`数据类型是与众不同的，在其中没有提供任何比较操作符。这是因为对于XML数据没有良好定义的并且通用的比较算法。其后果之一就是无法通过比较一个`xml`列和一个搜索值来检索行。因此XML值应该通常伴随着一个独立的键域，例如ID。比较XML值的一种可选方案是将它们先转换成字符串，但是要注意字符串比较并不是一种有用的XML比较方法。

正因为没有用于`xml`数据类型的比较操作符，所以不能直接在这种类型的列上创建索引。如果想要在XML数据中的快速搜索，可能的解决方案包括将表达式转换成字符串类型并且建立索引，或者索引一个XPath表达式。当然，要用被索引的表达式进行搜索，实际的查询也必须被调整。

处理XML

为了处理数据类型`xml`的值，Greenplum数据库提供了函数`xpath`和`xpath_exists`，它们可以计算XPath 1.0表达式。

```
xpath(, xml [, nsarray])
```

函数 `xpath` 对 XML 值 `xml` 计算 XPath 表达式 `xpath` (一个文本值)。它返回一个 XML 值的数组，它对应于 XPath 表达式产生的节点集合。

第二个参数必须是一个结构良好的 XML 文档。特别是它必须有一个单一的根节点元素。

该函数可选的第三个参数是一个名字空间映射的数组。这个数组应该是一个二维文本数组，其第二轴线的长度等于 2 (即，它应该是一个数组的数组，每一个元素数组由刚好两个元素组成)。每个数组项的第一个元素是名字空间名称 (别名)，第二个元素是名字空间的 URI。不要求这个数组中提供的别名与 XML 文档本身中使用的别名相同 (换句话说，在 XML 文档和 `xpath` 函数上下文中，别名都是本地的)。

例子：

```
SELECT xpath('/my:a/text()', '<my:a  
              xmlns:my="http://example.com">test</my:a>',  
              ARRAY[ARRAY[ 'my', 'http://example.com' ]]);  
  
xpath  
-----  
{test}  
(1 row)
```

为了处理默认 (匿名) 名字空间，可以这样做：

```
SELECT xpath('//mydefns:b/text()', '<a  
              xmlns="http://example.com"><b>test</b></a>',  
              ARRAY[ARRAY[ 'mydefns',  
                  'http://example.com' ]]);  
  
xpath  
-----  
{test}  
(1 row)
```

```
xpath_exists(xpath, xml [, nsarray])
```

函数 `xpath_exists` 是 `xpath` 函数的一种特殊形式。与返回满足该 XPath 的个体 XML 值不同，这个函数返回一个布尔值来表示该查询是否被满足。这个函数等效于标准的 `XMLEXISTS` 谓词，不过它还提供了对名字空间映射参数的支持。

例子：

```

SELECT xpath_exists('/my:a/text()', '<my:a
                     xmlns:my="http://example.com">test</my:a>',
                     ARRAY[ARRAY['my',
                     'http://example.com']]));

xpath_exists
-----
t
(1 row)

```

将表映射到XML

下列函数把关系表的内容映射到XML值。它们可以被认为是XML导出功能：

```

table_to_xml(tbl regclass, nulls boolean, tableforest
boolean, targetns text)
query_to_xml(query text, nulls boolean, tableforest
boolean, targetns text)
cursor_to_xml(cursor refcursor, count int, nulls boolean,
              tableforest boolean, targetns text)

```

每个函数的返回类型都是xml。

`table_to_xml`映射由参数`tbl`指定的表的内容。`regclass`类型接受使用通常记法标识表的字符串，包括可选的方案限定和双引号。`query_to_xml`执行文本作为参数`query`传入的查询并且映射结果集。`cursor_to_xml`从参数`cursor`指定的游标中取出指定数目的行。如果不得不映射大型表，推荐使用这种变体，因为每个函数都会在内存中构建结果值。

如果`tableforest`为假，那么得到的结果XML文档看起来像这样：

```

<tablename>
  <row>
    <columnname1>data</columnname1>
    <columnname2>data</columnname2>
  </row>

  <row>
    ...
  </row>

  ...
</tablename>

```

如果`tableforest`为真，结果会是一个这样的XML内容片段：

```

<tablename>
  <columnname1>data</columnname1>
  <columnname2>data</columnname2>
</tablename>

<tablename>
  ...
</tablename>

...

```

如果没有表名可用，也就是说映射一个查询或者游标时，会以第一种格式使用table，而以第二种格式使用row。

对这些格式的选择取决于用户。第一种格式是一种正确的XML文档，它在很多应用中都很重要。如果结果值后面会被重新组装到一个文档中，第二种格式在cursor_to_xml函数中更有用。上面讨论的产生XML内容的函数（尤其是xmlelement）可以被用来按照要求修改结果。

数据值会按照函数xmlelement所描述的相同方式来映射。

参数nulls决定空值是否应该被包括在输出中。如果为真，列中的空值会被表示为：

```
<columnname xsi:nil="true" />
```

其中xsi是XML模式实例的XML名字空间前缀。一个合适的名字空间声明将被加入到结果值中。如果为假，包含空值的列会被从输出中省略。

参数targetns指定想要的结果XML名字空间。如果没有特定的名字空间想要，应该传递一个空字符串。

下列函数返回XML模式文档，它描述由上述对应函数执行的映射：

```

able_to_xmlschema(tbl regclass, nulls boolean, tableforest
boolean, targetns text)
query_to_xmlschema(query text, nulls boolean, tableforest
boolean, targetns text)
cursor_to_xmlschema(cursor refcursor, nulls boolean,
tableforest boolean, targetns text)

```

关键的一点是，应该传递相同的参数来获得相匹配的XML数据映射以及XML模式文档。

下面的函数在一个文档（或者forest）中产生XML数据映射和相应XML

的 模式，它们会被链接在一起。如果想要自包含和自描述的结果，它们会很有用：

```
table_to_xml_and_xmleschema(tbl regclass, nulls boolean,
tableforest boolean, targetns text)
query_to_xml_and_xmleschema(query text, nulls boolean,
tableforest boolean, targetns text)
```

此外，下面的函数可以用来产生整个方案或者整个当前数据库的类似映射：

```
schema_to_xml(schema name, nulls boolean, tableforest
boolean, targetns text)
schema_to_xmleschema(schema name, nulls boolean, tableforest
boolean, targetns text)
schema_to_xml_and_xmleschema(schema name, nulls boolean,
tableforest boolean, targetns text)

database_to_xml(nulls boolean, tableforest boolean,
targetns text)
database_to_xmleschema(nulls boolean, tableforest boolean,
targetns text)
database_to_xml_and_xmleschema(nulls boolean, tableforest
boolean, targetns text)
```

注意这些函数可能会产生大量需要在内存中构建的数据。在请求大型方案或者数据库的内容映射时，应考虑分别映射表，甚至可能通过游标来映射。

模式内容映射的结果可能像这样：

```
<schemaname>
  table1-mapping
  table2-mapping
  ...
</schemaname>
```

其中表映射的格式取决于tableforest参数，如前所释。

数据库内容映射的结果可能像这样：

```
<dbname>
  <schemaname>
    ...
  </schemaname>
```

```

<schema2name>
  ...
</schema2name>

  ...

</dbname>

```

其中模式映射和前面一样。

下面的例子展示了对这些函数产生的输出的使用。该例子展示了一个XLST样式表，它把table_to_xml_and_xmlschema的输出转换成一个含有表数据表格化呈现的HTML文档。以类似的方式，来自这些函数的结果可以被转换成其基于XML的格式。

```

<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.w3.org/1999/xhtml"
>

  <xsl:output method="xml"
    doctype-
    system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
    doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
    indent="yes" />

  <xsl:template match="/" />
  <xsl:variable name="schema" select="//xsd:schema" />
  <xsl:variable name="tabletypename" select="$schema/xsd:element[@name=name(current())]/@type" />
  <xsl:variable name="rowtypename" select="$schema/xsd:complexType[@name=$tabletypename]/xsd:seq

  <html>
    <head>
      <title><xsl:value-of select="name(current())" />
    </title>
    </head>
    <body>
      <table>
        <tr>
          <xsl:for-each select="$schema/xsd:complexType[@name=$rowtypename]/xsd:sequence">
            <th><xsl:value-of select="." /></th>
          </xsl:for-each>
        </tr>
        <xsl:for-each select="row" >
          <tr>

```

```

<xsl:for-each select="*">
    <td><xsl:value-of select=". . . /></td>
</xsl:for-each>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>

```

XML函数参考

这一节中描述的函数在类型`xml`的值上操作。小节[XML谓词](#)也包含有关`xml`函数和类函数表达式的信息。

Function:

`xmlcomment`

Synopsis:

```
xmlcomment(text)
```

函数`xmlcomment`创建一个XML值，它含有一个以指定文本作为内容的XML注释。文本中不能包含"--"或者以一个"--"结尾，这样得到的结果结构才是一个合法的XML注释。如果该参数是空，结果也是空。

例子:

```

SELECT xmlcomment('hello');

xmlcomment
-----
<!--hello-->

```

Function:

`xmlconcat`

Synopsis:

```
xmlconcat(xml[ , ...])
```

函数`xmlconcat`把一个列表中的个体XML值串接以创建一个包含XML内容片段的单值。空值会被忽略，只有在没有非空参数的情况下结果才为空。

Example:

```
SELECT xmlconcat('<abc/>', '<bar>foo</bar>');

xmlconcat
-----
<abc/><bar>foo</bar>
```

XML声明（如果存在）会按下列方式组合：

- 如果所有参数值都有相同的XML版本声明，结果中也会用那个版本，不会使用别的版本。
- 如果所有参数值都有单独的声明值"yes"，那么该值会被用在结果中。
- 如果所有参数值都有单独的声明值且至少有一个为"no"，那么那一个会被用在结果中。否则，结果将没有单独的声明。
- 如果结果被判定为有一个单独声明但没有版本声明，那么将使用版本1.0的版本声明，因为XML要求XML声明包含版本声明。

在所有情况下编码声明都会被忽略并且移除。

例子：

```
SELECT xmlconcat('<?xml version="1.1"?><foo/>', '<?xml
version="1.1" standalone="no"?><bar/>');

xmlconcat
-----
<?xml version="1.1"?><foo/><bar/>
```

函数：

`xmlelement`

概要：

```
xmlelement(name name [, xmлатtributes(value [AS attname] [, ...
...])] [, content, ...])
```

The `xmlelement`表达式产生一个有给定名称、属性和内容的XML元素。

例子：

```

SELECT xmlelement(name foo);

xmlelement
-----
<foo/>

SELECT xmlelement(name foo, xmlattributes('xyz' as bar));

xmlelement
-----
<foo bar="xyz"/>

SELECT xmlelement(name foo, xmlattributes(current_date as
bar), 'cont', 'ent');

xmlelement
-----
<foo bar="2017-01-26">content</foo>

```

没有合法XML名称的元素和属性名会被转义，转义的方式是将不合法的字符用序列_xHHHH_替换，其中HHHH是该字符以16进制表达的Unicode代码点。例如：

```

SELECT xmlelement(name "foo$bar", xmlattributes('xyz' as
"a&b"));

xmlelement
-----
<foo_x0024_bar a_x0026_b="xyz"/>

```

如果属性值是一个列引用，则不需要指定显式的属性名，这种情况下默认将把列名用作该属性的名字。在其他情况下，必须给该属性一个显式的命名。因此这个例子是合法的：

```

CREATE TABLE test (a xml, b xml);
SELECT xmlelement(name test, xmlattributes(a, b)) FROM
test;

```

但这些就不合法：

```

SELECT xmlelement(name test, xmlattributes('constant'), a,
b) FROM test;
SELECT xmlelement(name test, xmlattributes(func(a, b)))
FROM test;

```

元素内容（如果指定）将被根据其数据类型进行格式化。如果内容本身是类型xml，则会构建复杂的XML文档。例如：

```

SELECT xmlelement(name foo, xmlattributes('xyz' as bar),
xmlelement(name abc),

```

```

xmlcomment('test'),
xmlelement(name xyz));

xmlelement
-----
<foo bar="xyz"><abc/><!--test--><xyz/></foo>

```

其他类型的内容将被格式化为合法的XML字符数据。这意味着特别的字符 <, > 以及 & 将会被转换成实体。二进制数据（数据类型 `bytea`）将被表示为 `base64` 或者十六进制编码，具体取决于配置参数 [xmlbinary](#)。为了将SQL和Greenplum数据库数据类型与XML模式说明一致，用于个别数据类型的特殊行为也预计会逐步出现，届时会有更精确的描述。

函数：

`xmlforest`

概要：

```
xmlforest(content [AS name] [, ...])
```

`xmlforest` 表达式产生一个使用给定名称和内容的元素的 XML 森林（序列）。

例子：

```

SELECT xmlforest('abc' AS foo, 123 AS bar);

xmlforest
-----
<foo>abc</foo><bar>123</bar>

SELECT xmlforest(table_name, column_name)
FROM information_schema.columns
WHERE table_schema = 'pg_catalog';

xmlforest
-----
-----
<table_name>pg_authid</table_name>
<column_name>rolname</column_name>
<table_name>pg_authid</table_name>
<column_name>rolsuper</column_name>

```

如在第二个例子中所见，如果内容值是一个列引用，则列名可以被省略，那种情况下默认会使用列名。否则，必须指定一个列名。

不是合法XML名称的元素名会按照前述对`xmlelement`的方式转义。类似地，内容数据也会被转义来得到合法的`xml`内容，除非它已经是类型`xml`。

注意如果XML森林由多于一个元素组成，它们就不是合法的XML文档，因此将`xmlforest`表达式包裹在`xmlelement`中可能会有用。 Note that XML forests are not valid XML documents if they consist of more than one element,

函数：

`xmlpi`

概要：

```
xmlpi(name target [, content])
```

`xmlpi`表达式创建一个XML处理指令。内容（如果存在）不能包含字符序列`?>`。

例如：

```
SELECT xmlpi(name php, 'echo "hello world";');

xmlpi
-----
<?php echo "hello world";?>
```

Function:

`xmlroot`

概要：

```
xmlroot(xml, version text | no value [, standalone yes|no|no value])
```

`xmlroot`表达式修改一个XML值的根节点属性。如果指定了一个版本，它会替换掉根节点版本声明中的值；如果指定了一个单独的设置，它会替换掉根节点的单独声明的值。

```
SELECT xmlroot(xmlopse(document '<?xml version="1.1"?>
<content>abc</content>'),
                version '1.0', standalone yes);
```

`xmlroot`

```
<?xml version="1.0" standalone="yes"?>
<content>abc</content>
```

函数：

`xmlagg`

```
xmlagg (xml)
```

和这里描述的其他函数不同，函数`xmlagg`是一个聚集函数。它把输入值串接到一个聚集函数调用中（很像`xmlconcat`的做法），不过串接发生在行之间而不是单个行的表达式之间。有关聚集函数的更多信息，请见[使用函数和操作符](#)。

例子：

```
CREATE TABLE test (y int, x xml);
INSERT INTO test VALUES (1, '<foo>abc</foo>');
INSERT INTO test VALUES (2, '<bar/>');
SELECT xmlagg(x) FROM test;
xmlagg
-----
<foo>abc</foo><bar/>
```

为了决定串接的顺序，`ORDER BY`子句可能会被增加到聚集调用上。例如：

```
SELECT xmlagg(x ORDER BY y DESC) FROM test;
xmlagg
-----
<bar/><foo>abc</foo>
```

下面的非标准方法在之前的版本中常常会被推荐，它们在特定情况下可能仍然能发挥作用：

```
SELECT xmlagg(x) FROM (SELECT * FROM test ORDER BY y DESC)
AS tab;
xmlagg
-----
<bar/><foo>abc</foo>
```

XML谓词

这一节中描述的表达式会检查`xml`值的属性。

表达式:

`IS DOCUMENT`

摘要:

```
xml IS DOCUMENT
```

如果参数`XML`值是一个正确的XML文档，表达式`IS DOCUMENT`会返回真，如果不是正确的XML文档（即是一个内容片段）就会返回假，如果参数为空则返回空。

表达式:

`XMLEXISTS`

摘要:

```
XMLEXISTS(text PASSING [BY REF] xml [BY REF])
```

如果第一个参数中的XPath表达式返回任何节点，则函数`xmlexists`返回真，否则返回假（只要有一个参数为空，结果也为空）。

例子:

```
SELECT xmlexists(''//town[text() = ''Toronto'']'' PASSING BY
REF
      '<towns><town>Toronto</town><town>Ottawa</town>
</towns>' );
xmlexists
-----
t
(1 row)
```

`BY REF`子句在Greenplum数据库中没有效果，但是为了与其他实现的SQL符合性和兼容性还是允许使用它。每一种SQL标准都要求第一个`BY REF`，而第二个是可选的。还要注意SQL标准指定`xmlexists`结构需要一个XQuery表达式作为第一个参数，但Greenplum数据库当前只支持XPath，它只是XQuery的一个子集。

表达式:

`xml_is_well_formed`

Synopsis:

```
xml_is_well_formed(text)
xml_is_well_formed_document(text)
xml_is_well_formed_content(text)
```

这些函数检查一个文本字符串是否为结构良好的XML，返回一个布尔结果。`xml_is_well_formed_document`检查一个结构良好的文档，而`xml_is_well_formed_content`检查结构良好的内容。如果`xmloption`配置参数被设置为`DOCUMENT`，`xml_is_well_formed`会做前者；如果设置为`CONTENT`，则`xml_is_well_formed`会做后者。这意味着`xml_is_well_formed`对查看一个到类型`xml`的简单造型是否会成功很有用，而其他两个函数则可用于查看`XMLPARSE`的相应变体是否会成功。

例子：

```
SET xmloption TO DOCUMENT;
SELECT xml_is_well_formed('<>');
xml_is_well_formed
-----
f
(1 row)

SELECT xml_is_well_formed('<abc/>');
xml_is_well_formed
-----
t
(1 row)

SET xmloption TO CONTENT;
SELECT xml_is_well_formed('abc');
xml_is_well_formed
-----
t
(1 row)

SELECT xml_is_well_formed_document('<pg:foo
xmlns:pg="http://postgresql.org/stuff">bar</pg:foo>');
xml_is_well_formed_document
-----
t
(1 row)

SELECT xml_is_well_formed_document('<pg:foo
xmlns:pg="http://postgresql.org/stuff">bar</my:foo>');
xml_is_well_formed_document
-----
f
(1 row)
```

最后一个例子展示了包括名字空间是否正确匹配在内的检查。

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

关于Greenplum的查询
处理

□ 关于GPORCA

定义查询

WITH查询 (公用表表
达式)

使用函数和操作符

使用JSON数据

使用XML数据

□ 使用全文搜索

查询性能

管理查询生成的溢出文
件

查询分析

□ 使用外部数据

Greenplum数据库提供用于查询自然语言文档的数据类型, 函数, 运算符, 索引类型和配置。

• [关于全文搜索](#)

本主题概述了Greenplum数据库全文搜索, 基本文本搜索表达式, 配置和自定义文本搜索。以及Greenplum数据库全文搜索与Pivotal GPText的比较。

• [在数据库表中搜索文本](#)

本主题说明如何使用文本搜索运算符搜索数据库表以及如何创建索引以加快文本搜索。

• [控制文本搜索](#)

本主题说明如何创建搜索和查询向量, 如何对搜索结果进行排名, 以及如何在文本搜索查询的结果中突出显示搜索词。

• [文本搜索附加功能](#)

Greenplum数据库具有其他附加功能和操作符, 您可以使用它们来操作搜索和查询向量, 以及重写搜索查询。

• [文本搜索解析器](#)

本主题描述了Greenplum数据库文本搜索解析器从原始文本生成的token类型。

• [文本搜索词典](#)

由Greenplum数据库全文搜索解析器生成的token通过一系列字典传递以产生标准化术语或“词位”。不同种类的词典可用于以不同方式和不同语言过滤和转换token。

• [文本搜索配置示例](#)

本主题说明如何创建自定义文本搜索配置以处理文档和查询文本。

• [测试和调试文本搜索](#)

本主题介绍可用于测试和调试搜索配置或配置中指定的单个解析器和词典的Greenplum数据库函数。

• [文本搜索的GiST和GIN索引](#)

本主题描述并比较用于全文搜索的Greenplum数据库索引类型。

• [psql支持](#)

psql命令行实用程序提供了一个元命令, 用于显示有关Greenplum数据库全文搜索配置的信息。

• [限制](#)

本主题列出了Greenplum数据库全文搜索对象的限制和值。

Parent topic: [查询数据](#)

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

关于Greenplum的查询
处理

□ 关于GPORCA

定义查询

WITH查询 (公用表表
达式)

使用函数和操作符

使用JSON数据

使用XML数据

□ 使用全文搜索

查询性能

管理查询生成的溢出文
件

查询分析

□ 使用外部数据

Greenplum数据库动态地消除一个表中的不相关分区并且为查询中的不同操作符最优地分配内存。这些增强可以让查询扫描更少的数据、加速查询处理以及支持更大的并发度。

● 动态分区消除

在Greenplum数据库中，只有在查询运行时才可用的值被用来动态地剪枝分区，这会改进查询的处理速度。通过将服务器配置参数`gp_dynamic_partition_pruning`设置为ON或者OFF可启用或者禁用动态分区消除，该参数默认是ON。

● 内存优化

Greenplum数据库为一个查询中的不同操作符最优地分配内存以及在处理查询的各阶段中释放并重新分配内存。

Note: Greenplum数据库默认使用下一代查询优化器GPORCA。GPORCA扩展了Greenplum数据库传统优化器的规划和优化能力。有关GPORCA的特性和限制信息，请参看 [GPORCA概述](#)

Parent topic: [查询数据](#)



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

关于Greenplum的查询
处理

□ 关于GPORCA

定义查询

WITH查询 (公用表表达式)

使用函数和操作符

使用JSON数据

使用XML数据

□ 使用全文搜索

查询性能

管理查询生成的溢出文件

查询分析

□ 使用外部数据

如果没有足够的内存让一个SQL查询在内存中运行, Greenplum数据库会在磁盘上创建溢出文件 (也被称为工作文件)。默认的100,000个溢出文件对于大部分查询来说够用了。但是, 如果一个查询创建的溢出文件超过指定个数, Greenplum数据库会返回这个错误:

```
ERROR: number of workfiles per query limit exceeded
```

导致生成大量溢出文件的原因包括:

- 在被查询的数据中存在数据倾斜。
- 为该查询分配的内存量太小。

通过更改该查询、更改数据分布或者更改系统内存配置可能可以让查询成功运行。可以使用 *gp_workfile_** *gp_workfile_**视图查看溢出文件使用信息。用Greenplum数据库的服务器配置参数*max_statement_mem*, *statement_mem*, 或者资源队列可以控制查询所使用的最大内存量。

[监控Greenplum系统](#) 包含下列信息:

- 有关倾斜的信息以及如何检查数据倾斜
- 有关使用 *gp_workfile_* views* 视图的信息

有关服务器配置参数的信息, 请见 *Greenplum*数据库参考指南。有关资源队列的信息, 请见 [使用资源队列](#).

如果已经确定查询必须创建比服务器配置参数*gp_workfile_limit_files_per_query*所允许的更多的溢出文件, 可以增加该参数的值。

Parent topic: [查询数据](#)



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

关于Greenplum的查询
处理

□ 关于GPORCA

定义查询

WITH查询 (公用表表达式)

使用函数和操作符

使用JSON数据

使用XML数据

□ 使用全文搜索

查询性能

管理查询生成的溢出文件

查询分析

□ 使用外部数据

检查性能不好的查询的查询计划,来确定可能的性能调优机会。

Greenplum数据库为每个查询设计一个 查询计划。选择正确的查询计划来匹配查询和数据结构对好的性能是必要的。一个查询计划定义Greenplum数据库将如何在并行执行环境中运行查询。

查询优化器使用数据库维护的数据统计信息来选择具有最低可能代价的查询计划。代价以磁盘I/O来度量, 磁盘I/O用取得的磁盘页面为单位。目标是最小化计划的总执行代价。

可以用EXPLAIN命令查看一个给定查询的计划。EXPLAIN展示查询规划器对该查询计划估计的代价。例如:

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

EXPLAIN ANALYZE不仅运行该语句, 还会显示它的计划。这有助于判断优化器的估计与现实有多接近。例如:

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

Note: 在Greenplum数据库中, 默认的GPORCA优化器与传统查询优化器共存。GPORCA生成的EXPLAIN输出与传统查询优化器生成的输出不同。

默认情况下, Greenplum数据库会在可能时使用GPORCA来为查询生成执行计划。

当EXPLAIN ANALYZE命令使用GPORCA时, EXPLAIN计划只显示被排除的分区数。被扫描的分区不会被显示。要在segment实例 日志中显示被扫描分区的名称, 可以把服务器配置参数gp_log_dynamic_partition_pruning设置为on。这个SET命令的例子启用了该参数。

```
SET gp_log_dynamic_partition_pruning = on;
```

有关GPORCA的信息, 请见[查询数据](#)**Parent topic:** [查询数据](#)

阅读EXPLAIN输出

一个查询计划是一棵节点的树。计划中的每个节点表示一个操作，例如表扫描、连接、聚集或者排序。

应该从底向上阅读计划：每个节点会把行交给直接在它上面的节点。一个计划中的底层节点通常是表扫描操作：顺序的、索引的或者位图索引扫描。如果该查询要求那些行上的连接、聚集、排序或者其他操作，就会有额外的节点在扫描节点上面负责执行这些操作。最顶层的计划节点通常是Greenplum数据库的移动节点：重新分布、显式重新分布、广播或者收集移动。这些操作在查询处理时在segment实例之间移动行。

`EXPLAIN` 的输出对计划树中的每个节点都有一行并且为该计划节点显示基本的节点类型和下列执行代价估计：

- **cost** —以磁盘页面获取为单位度量。1.0等于一次顺序磁盘页面读取。第一个估计是得到第一行的启动代价，第二个估计是得到所有行的总代价。总代价总是假定所有的行都将被检索，但并不总是这样。例如，如果查询使用了`LIMIT`，并非所有的行都会被检索。
- **rows** —这个计划节点输出的总行数。这个数字通常小于被该计划节点处理或者扫描的行数，它反映了任意`WHERE`子句条件的估计选择度。理想情况下，最顶层节点的估计近似于该查询实际返回、更新或者删除的行数。
- **width** —这个计划节点输出的所有行的总字节数。

注意以下几点：

- 一个节点的代价包括其子节点的代价。最顶层计划节点有对于该计划估计的总执行代价。这就是优化器想要最小化的数字。
- 代价只反映了被查询优化器加以考虑的计划执行的某些方面。例如，代价不反映将结果行传送到客户端花费的时间。

EXPLAIN实例

下面的例子描述了如何阅读一个查询的`EXPLAIN`查询代价：

```
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

```

从底向上阅读这个计划。一开始，查询优化器顺序地扫描`names`表。注意`WHERE`子句被应用为一个`filter`条件。这意味着扫描操作会对它扫描的每个行检查该条件并且只输出满足该条件的行。

扫描操作的结果被传递给一个 *gather motion* 操作。在Greenplum数据库中，收集移动是segment实例何时把行发送给Master。在这个例子中，我们有两个Segment实例会向一个Master实例发送。这个操作工作在并行执行计划的`slices`上。查询计划会被划分成`slices`，这样segment实例可以并行工作在查询计划的片段上。

为这个计划估计的启动代价是00.00（没有代价）而总代价是20.88次磁盘页面获取。优化器估计这个查询将返回一行。

阅读EXPLAIN ANALYZE输出

`EXPLAIN ANALYZE`规划并且运行语句。`EXPLAIN ANALYZE`计划会把实际执行代价和优化器的估计一起显示。这允许用户查看优化器的估计是否接近于实际。`EXPLAIN ANALYZE`也展示下列信息：

- 查询执行的总运行时间（以毫秒为单位）。
- 查询计划每个切片使用的内存，以及为整个查询语句保留的内存。
- 一个计划节点操作中涉及的工作者（segment实例）数量。其中只统计返回行的segment实例。
- 为该操作产生最多行的segment实例返回的行最大数量。如果多个segment实例产生了相等的行数，`EXPLAIN ANALYZE`会显示那个用了运行最长`<time>`的segment实例。
- 为一个操作产生最多行的segment实例的ID。
- 相关操作使用的内存量（`work_mem`）。如果`work_mem`不足以在内存中执行该操作，计划会显示溢出到磁盘的数据量最少的segment实例的溢出数据量。例如：

```

Work_mem used: 64K bytes avg, 64K bytes max (seg0).
Work_mem wanted: 90K bytes avg, 90K bytes max (seg0) to
lessen
workfile I/O affecting 2 workers.

```

- 产生最多行的segment实例检索到第一行的时间（以毫秒为单位）以及该segment实例检索到所有行花掉的时间。如果到获取第1行

的`<time>`与运行到结束的`<time>`相同，结果中可能会省略前者。

EXPLAIN ANALYZE实例

这个例子用同一个查询描述了如何阅读一个EXPLAIN ANALYZE查询计划。这个计划中粗体部分展示了每一个计划节点的实际计时和返回行，以及整个查询的内存和时间统计信息。

```
EXPLAIN ANALYZE SELECT * FROM names WHERE name = 'Joelle';
QUERY PLAN
-----
-
Gather Motion 2:1 (slice1; segments: 2) (cost=0.00..20.88
rows=1 width=13)
  Rows out: 1 rows at destination with 0.305 ms to first
row, 0.537 ms to end, start offset by 0.289 ms.
    -> Seq Scan on names (cost=0.00..20.88 rows=1
width=13)
      Rows out: Avg 1 rows x 2 workers. Max 1 rows
(seg0) with 0.255 ms to first row, 0.486 ms to end, start
offset by 0.968 ms.
        Filter: name = 'Joelle'::text
        Slice statistics:
          (slice0) Executor memory: 135K bytes.

          (slice1) Executor memory: 151K bytes avg x 2 workers,
151K bytes max (seg0).

Statement statistics:
  Memory used: 128000K bytes
  Total runtime: 22.548 ms
```

从底向上阅读这个查询。运行这个查询花掉的总时间是22.548毫秒。

sequential scan 操作只有一个返回行的segment实例（`seg0`），并且它只返回 1 行。它用了 0.255 毫秒找到第一行且用了 0.486 毫秒来扫描所有的行。这个结果接近于优化器的估计：查询优化器估计这个查询将会返回一行。收集移动（segment实例向Master发送数据）接收到了 1 行。这个操作的总消耗时间是 0.537 毫秒。

判断查询优化器

用户可以查看EXPLAIN输出来判断是否启用了GPORCA以及这个解释计划是由GPORCA还是传统查询优化器生成。这一信息出现在EXPLAIN输出的末尾。Settings行显示服务器配置参

数OPTIMIZER的设置。Optimizer status行显示该解释计划是由GPORCA还是传统查询优化器生成。

对于这两个查询计划的例子，GPORCA被启用，所以服务器配置参数OPTIMIZER为on。对于第一个计划，GPORCA生成了该EXPLAIN计划。对于第二个计划，Greenplum数据库回退到传统查询优化器来生成该查询计划。

```
QUERY PLAN
-----
-----
Aggregate (cost=0.00..296.14 rows=1 width=8)
    -> Gather Motion 2:1 (slice1; segments: 2)
(cost=0.00..295.10 rows=1 width=8)
        -> Aggregate (cost=0.00..294.10 rows=1 width=8)
            -> Seq Scan on part (cost=0.00..97.69
rows=100040 width=1)
Settings: optimizer=on
Optimizer status: Pivotal Optimizer (GPORCA) version 1.584
(5 rows)
```

```
explain select count(*) from part;
```

```
QUERY PLAN
-----
-----
Aggregate (cost=3519.05..3519.06 rows=1 width=8)
    -> Gather Motion 2:1 (slice1; segments: 2)
(cost=3518.99..3519.03 rows=1 width=8)
        -> Aggregate (cost=3518.99..3519.00 rows=1
width=8)
            -> Seq Scan on part (cost=0.00..3018.79
rows=100040 width=1)
Settings: optimizer=on
Optimizer status: Postgres query optimizer
(5 rows)
```

对于这个查询，服务器配置参数OPTIMIZER是off。

```
explain select count(*) from part;

QUERY PLAN
-----
-----
Aggregate (cost=3519.05..3519.06 rows=1 width=8)
    -> Gather Motion 2:1 (slice1; segments: 2)
(cost=3518.99..3519.03 rows=1 width=8)
        -> Aggregate (cost=3518.99..3519.00 rows=1
width=8)
            -> Seq Scan on part (cost=0.00..3018.79
rows=100040 width=1)
Settings: optimizer=off
```

Optimizer status: Postgres query optimizer

(5 rows)

检查查询计划来解决问题

如果一个查询执行得不好，检查它的查询并且提出以下问题：

- 该计划中的操作花费了特别长的时间吗？查找消耗了多数查询执行时间的操作。例如，如果一个索引扫描花费了比预期长的时间，该索引可能过期并且需要重建索引。或者，调整enable_<operator>参数来看看是否能够强制传统查询优化器（规划器）来为该查询选择一个不同的计划。
- 优化器的估计是否接近实际？运行EXPLAIN ANALYZE并且查看规划器估计的行数是否接近查询操作实际返回的行数。如果有很大的差别，应在相关列上收集更多统计信息。更多 EXPLAIN ANALYZE和ANALYZE命令的信息请见Greenplum数据库参考指南
- 在该计划中是否很早就应用了选择性谓词？在计划中早些应用最具选择性的过滤条件，这样会有较少的行在计划树中向上移动。如果查询计划没有正确地估计查询谓词的选择度，应在相关列上收集更多统计信息。更多有关收集统计信息的内容请见Greenplum数据库参考指南中的ANALYZE命令。用户还可以尝试重新排序SQL语句中的WHERE子句。
- 优化器是否选择了最好的连接顺序？当查询连接多个表时，确保优化器选择了最具选择性的连接顺序。计划中应该尽早做消除最多行的连接，这样会有较少的行在计划树中向上移动。如果计划没有选择最优的连接顺序，可以设置joinCollapse_limit=1并且在SQL语句中使用显式的JOIN语法来强制传统查询优化器（规划器）用指定的连接顺序。还可以在相关的连接列上收集更多的统计信息。更多有关收集统计信息的内容请见Greenplum数据库参考指南中的ANALYZE命令。
- 优化器是否有选择地扫描分区表？如果在使用表分区，优化器是否有选择地只扫描满足查询谓词所需的子表？对父表的扫描应该会返回0行，因为父表中不包含任何数据。一个显示选择性分区扫描的查询计划例子，可见[验证分区策略](#) 验证分区策略
- 优化器是否在适用时选择了哈希聚集和哈希连接操作？哈希操作通常比其他类型的连接或者聚集快很多。行比较和排序在内存中完成而不需要读写磁盘。为了让查询优化器选择哈希操作，必须由足够多的可用内存来保存估计数量的行。尝试增加工作内存来改进查询的性能。如果可能，未查询运行一次EXPLAIN ANALYZE来显示那些计划操作会溢出到磁盘、它们使用了多少工作内存以及需要多少内存来避免溢出到磁盘。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.

来自于 EXPLAIN ANALYZE的"bytes wanted"消息是基于被写出到工作文件的数据量的而且并不准确。所需的最小work_mem可以与建议值不同。

Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统
 - 管理Greenplum数据库访问
 - 定义数据库对象
 - 分布与倾斜
 - 插入, 更新, 和删除数据
 - 查询数据
 - 使用外部数据
 - 定义外部表
 - 使用PXF访问外部数据
 - 使用外部表访问外部数据
 - 使用Greenplum的并行文件服务器 (gpfdist)
 - 装载和卸载数据
 - 性能管理

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

E外部表允许把外部文件当作常规数据库表来访问。它们常常被用来把数据移进或者移出Greenplum数据库。

要创建一个外部表定义，用户需要指定输入文件的格式以及外部数据源的位置。有关输入文件格式的信息，请参考 [格式化数据文件](#)。

使用下列协议之一来访问外部表数据源。用户不能在CREATE EXTERNAL TABLE语句中混用协议：

- file:// 访问Segment主机上Greenplum数据超级用户 (gpadmin)能访问的外部数据文件。见 [file:// 协议](#)。
- gpfdist:// 指向一个位于文件主机上的目录并且向Greenplum数据库所有的Segment并行提供外部数据文件。见 [gpfdist:// 协议](#)。
- gpfdists:// 是gpfdist的一个安全版本。见 [gpfdists:// 协议](#)。
- s3:// 访问Amazon S3桶中的文件。见 [s3:// 协议](#)。
- The pxf:// 协议使用Greenplum平台扩展框架 (PXF) 访问外部HDFS文件和HBase和Hive表。见 [pxf:// 协议](#)。

Note:

s3:// 和 pxf:// 协议是自定义数据访问协议，file://, gpfdist://, 和 gpfdists:// 协议是实现在Greenplum数据库内部的。自定义和内部协议有以下不同：

- 自定义协议必须使用CREATE PROTOCOL 命令注册。在数据库中注册PXF扩展会创建 pxf:// 协议。(见 [使用PXF访问外部数据](#))您可以选择性地注册 s3:// 协议。(见 [配置和使用S3外部表](#))。内部协议始终存在且无法取消注册。
- 注册自定义协议时，会在pg_extprotocol catalog表中添加一行，以指定实现该协议的处理函数。协议的共享库必须安装在所有Greenplum数据库主机上。内部协议在pg_extprotocol表中没有记录，并且没有其他库需要安装。
- 要授予用户对自定义协议的权限，请使用GRANT [SELECT | INSERT | ALL] ON PROTOCOL。要允许(或拒绝)内部协议的用户权限，需要使用CREATE ROLE 或 ALTER ROLE将CREATEEXTTABLE (或NOCREATEEXTTABLE) 属性添加到每个用户的角色。

外部表从数据库内访问外部文件，就好像它们是常规的  表一样。用 gpfdist/gpfdists, pxf, 和 s3 协议定义的外部表通过使用所有 Greenplum 数据库的 Segment 资源装载或卸载数据来利

用Greenplum的并行机制。 pxf协议利用Hadoop分布式文件系统的并行体系结构来访问该系统上的文件。 s3协议利用了Amazon Web服务(AWS)的功能。

用户可以使用SELECT、JOIN或SORT EXTERNAL TABLE DATA等SQL命令直接并行地查询外部表数据，并且用户可以为外部表创建视图。

使用外部表的步骤是：

1. 定义外部表。

使用s3协议，用户还必须配置Greenplum数据库并且启用该协议。见[s3:// 协议](#)。

2. 做下面的事情之一：

- 在使用gpfdist或者gpdists协议时，启动Greenplum数据库文件服务器。
- 验证为s3协议做好了所需的一次性配置。

3. 把数据文件放置在正确位置。

4. 用SQL命令查询外部表。

Greenplum数据库提供可读和可写的外部表：

- 用于数据加载的可读外部表。可读外部表支持：

- 数据仓库中常见的基本提取，转换和加载(ETL)任务。
- 从多个Greenplum数据库段实例并行读取外部表数据，以优化大型加载操作。
- 过滤下推。如果查询包含WHERE子句，则可以将其传递给外部数据源。有关更多信息，请参考[gp_external_enable_filter_pushdown](#)服务器配置参数讨论。请注意，此功能目前仅支持pxf协议(参阅[pxf:// 协议](#))。

可读外部表仅允许SELECT操作。

- 用于数据卸载的可写外部表。可写外部表支持：

- 从数据库表中选择数据以插入可写外部表。
- 将数据以数据流形式发送到应用程序。例如，从Greenplum数据库卸载数据并将其发送到连接到另一个数据库或ETL工具的应用程序以在其他地方加载数据。
- 从Greenplum并行MapReduce计算接收输出。

可写外部表仅允许INSERT操作。

外部表可以是基于文件的或者是基于Web的。使用file://协议的外

部表示只读表。

- 普通（基于文件的）外部表访问静态平面文件。普通外部表是可以重新扫描的：在查询运行时数据是静态的。
- Web（基于Web的）外部表访问动态数据源，或者在一个用http://协议的Web服务器上，或者通过执行OS命令或脚本得到。外部Web表不是可以重新扫描的：在查询运行时数据可以改变。

转储和操作只针对外部表和外部Web表的定义而非数据源。

- file:// 协议**

file:// 协议被用在一个指定操作系统文件的URI中。

- gpfdist:// 协议**

gpfdist:// 协议被用在一个URI中引用一个正在运行的gpfdist实例。

- gpfdists:// 协议**

gpfdists:// 协议是gpfdist:// 协议的一个安全版本。

- pxf:// 协议**

您可以使用Greenplum平台扩展框架（PXF）pxf:// 协议访问驻留在外部Hadoop系统（HDFS，Hive，HBase），对象存储系统（Azure，Google云端存储，Minio，S3）和SQL数据库上的数据。

- s3:// 协议**

s3协议使用一个URL指定Amazon S3桶的位置和一个用来在桶里读写文件的前缀。

- 使用自定义协议**

一种自定义协议允许用户连接到一个不能用file://，gpfdist://，或者pxf:// 协议的Greenplum数据库。

- 处理外部表数据中的错误**

默认情况下，如果外部表数据中包含有一个错误，命令就会失败并且不会有数据被载入到目标数据库表中。

- 创建和使用外部Web表**

外部Web表允许Greenplum Database像处理常规数据表一样处理动态数据源。由于Web表数据可以在查询运行时更改，因此数据无法重新扫描。

- Examples for Creating External Tables**

These examples show how to define external data with different protocols. Each CREATE EXTERNAL TABLE command can contain only one protocol.

Parent topic: 使用外部数据

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 定义外部表

使用PXF访问外部数据

□ 使用外部表访问外部数据

使用Greenplum的并行文件
服务器 (gpfldst)

□ 装载和卸载数据

□ 性能管理

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

Greenplum数据库实现了SQL / MED规范的一部分，允许您使用常规SQL查询访问驻留在Greenplum之外的数据。这些数据称为外部数据。

您可以在外部数据包装器的帮助下访问外部数据。外部数据包装器是与远程数据源通信的库。该库隐藏了特定源的连接和数据访问详细信息。一些外部数据包装器将作为Greenplum Database contrib模块提供，包括file_fdw和postgres_fdw。如果现有的外部数据包装器都不适合您的需求，您可以编写如编写外部数据包装器中所述的自己的外部数据包装器。[写一个外部数据的包装器](#)

要访问外部数据，请创建外部服务器对象，该对象根据其支持的外部数据包装器使用的选项集定义如何连接到特定的远程数据源。然后创建一个或多个外部表，这些表定义远程数据的结构。外表可以像普通表一样在查询中使用，但外表在Greenplum数据库服务器中没有存储任何数据。每当访问外表时，Greenplum数据库都会要求外部数据包装器从远程源中获取数据或更新数据（如果包装器支持）。

访问远程数据可能需要对远程数据源进行身份验证。该信息可以由用户映射提供，该映射可以基于当前的Greenplum数据库角色提供诸如用户名和密码之类的附加数据。

有关其他信息，请参阅[CREATE FOREIGN DATA WRAPPER](#),
[CREATE SERVER](#), [CREATE USER MAPPING](#), 和 [CREATE FOREIGN TABLE](#).

- [写一个外部数据的包装器](#)

Parent topic: [使用外部数据](#)



Greenplum数据库® 6.0文档

□ 管理员指南

- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象
- 分布与倾斜
 - 插入, 更新, 和删除数据
- 查询数据
- 使用外部数据
- 定义外部表
 - file://协议
 - gpfdist://协议
 - gpfdists:// 协议
 - pxf:// 协议
 - s3:// 协议
- 使用自定义协议
- 处理外部表数据中的错误

创建和使用外部Web表

- Examples for Creating External Tables

外部Web表允许Greenplum Database像处理常规数据表一样处理动态数据源。由于Web表数据可以在查询运行时更改，因此数据无法重新扫描。

`CREATE EXTERNAL WEB TABLE`建立一个外部表的定义。你可以定义基于命令或基于URL的外部Web表。定义形式是不同的：不能将基于命令的和基于URL的定义混在一起。

Parent topic: [定义外部表](#)

基于命令的外部Web表

一个shell命令或者脚本的输出可定义基于命令的Web表数据。将命令放在`CREATE EXTERNAL WEB TABLE`的`EXECUTE`子句里。数据是命令运行时的最新数据。子句`EXECUTE`在指定的master或segment节点上运行shell命令或脚本。命令和脚本必须存在于`EXECUTE`子句中定义的host上。

默认，命令会在当活跃节点上有输出的行要处理时，运行在节点上。例如，如果每个宿主机器上运行4个有输出行待处理的节点实例，命令会在每个宿主机器上运行4次。可以限制运行web表命令的节点实例数量。所有在web表定义`ON`子句中的节点会并行运行命令。

指定在外部表定义中的命令会从数据库运行，但不会从`.bashrc`或`.profile`中加载环境变量。在`EXECUTE`子句中设置环境变量。例如：

```
=# CREATE EXTERNAL WEB TABLE output (output text)
  EXECUTE 'PATH=/home/gpadmin/programs; export PATH;
myprogram.sh'
  FORMAT 'TEXT';
```

脚本必须对于`gpadmin`用户是可执行的，并且在master和segment节点上路径相同。

下面命令定义了一个运行脚本的web表。脚本运行在有数据需要处理的节点上。

```
=# CREATE EXTERNAL WEB TABLE log_output
  (linenum int, message text)
  EXECUTE '/var/load_scripts/get_log_data.sh' ON HOST
```

```
FORMAT 'TEXT' (DELIMITER '|') ;
```

基于URL的外部Web表

基于URL的web表从HTTP协议的服务器访问数据。 Web表数据是动态的，不可重新扫描的。

在LOCATION里定义基于http://协议的web服务文件路径。 web数据文件必须存在于Greenplum节点可访问的web服务器上。 指定的URL数量与并行访问web表的节点实例数量相关。 例如，如果为一个有8个节点的Greenplum数据库指定两个外部文件， 8个节点中的两个会在查询运行时并行访问web表。

下面的例子定义了一个从多个URL获取数据的web表。

```
=# 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 );
```

外部文件服务器 (gpfdist)

Greenplum数据库® 6.0文档

- 管理员指南
- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象
 - 分布与倾斜
 - 插入、更新、和删除数据
- 查询数据
- 使用外部数据
 - 定义外部表
 - 使用PXF访问外部数据
 - 使用外部表访问外部数据
 - 使用Greenplum的并行文件服务器 (gpfdist)**
- 装载和卸载数据
- 性能管理

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

gpfdist 协议用于CREATE EXTERNAL TABLE SQL命令，以访问Greenplum Database gpfdist文件服务器实用程序提供的外部数据。当外部数据由gpfdist 提供时，Greenplum数据库系统中的所有节点都可以并行读取或写入外部表数据。

本主题介绍将gpfdist 与外部表一起使用的设置和管理任务。

- [关于gpfdist和外部表](#)
- [关于gpfdist设置和性能](#)
- [控制Segment并行](#)
- [安装gpfdist](#)
- [启动和停止gpfdist](#)
- [解决gpfdist的问题](#)

Parent topic: [使用外部数据](#)

关于gpfdist和外部表

gpfdist文件服务器实用程序位于Greenplum数据库主节点和每个节点上的\$GPHOME/bin目录中。启动gpfdist实例时，您指定一个侦听端口以及一个包含要读取的文件或要写入文件的目录的路径。例如，此命令在后台运行gpfdist，侦听端口8801，并在/home/gpadmin/external_files目录中提供文件：

```
$ gpfdist -p 8801 -d /home/gpadmin/external_files &
```

CREATE EXTERNAL TABLE命令LOCATION子句将外部表定义连接到一个或多个gpfdist实例。如果外部表是可读的，则gpfdist服务器从指定目录中的文件读取数据记录，将它们打包到块中，并在响应Greenplum数据库节点的请求时发送该块。这些节点解压缩它们接收的行并根据外部表的分发策略分发它们。如果外部表是可写表，则节点将请求中的行块发送到gpfdist，gpfdist将它们写入外部文件。

外部数据文件可以包含CSV格式的行或CREATE EXTERNAL TABLE命令的FORMAT子句支持的任何分隔文本格式。此外，gpfdist可以配置YAML格式的文件，以在支持的文本格式和另一种格式（例如XML或JSON）之间转换外部数据文件。有关如何使用gpfdist将外部XML文件读入Greenplum数据库可读外部表的示例，请参阅。

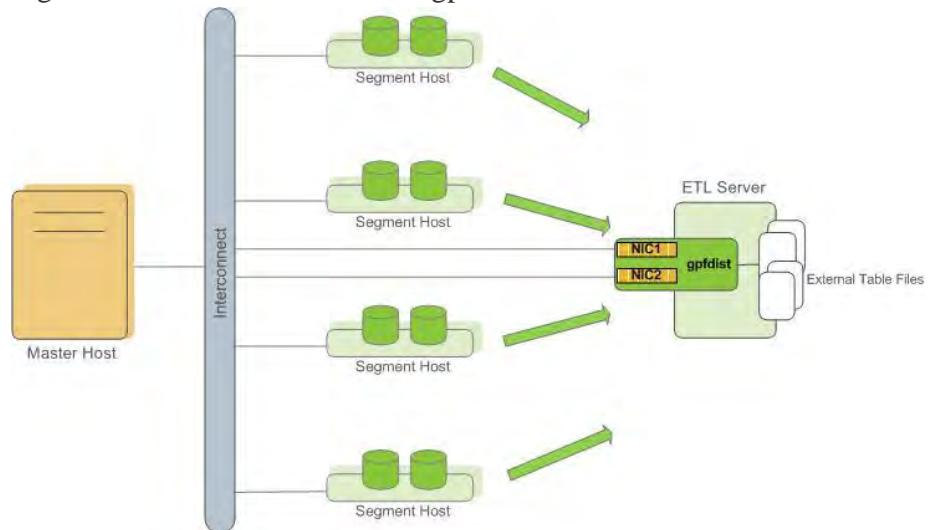
对于可读的外部表，`gpfdist`会自动解压缩gzip (.gz) 和bzip2 (.bz2) 文件。您可以使用通配符 (*) 或其他C样式模式匹配来表示要读取的多个文件。外部文件路径都假定相对于启动`gpfdist`实例时指定的路径。

关于gpfdist设置和性能

您可以在多个主机上运行`gpfdist`实例，也可以在每个主机上运行多个`gpfdist`实例。这允许您策略性地部署`gpfdist`服务器，以便通过利用所有可用的网络带宽和Greenplum数据库的并行性来获得快速的数据加载和卸载速率。

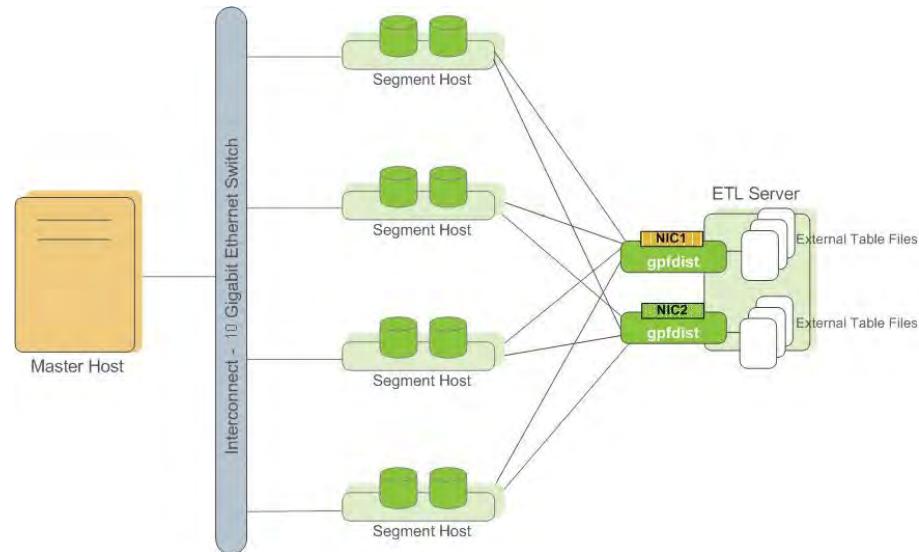
- 允许网络流量同时使用所有的ETL主机网络接口卡（NIC）。在ETL主机上运行一个`gpfdist`实例，然后在 on the ETL host, then declare the host name of each NIC in the 用户的外部表定义（见[Examples for Creating External Tables](#)）。）的LOCATION子句中声明每一个NIC的主机名。

Figure 1. 使用带多个NIC的单一gpfdist实例的外部表



- 在ETL主机上的多个`gpfdist`实例之间平均划分外部表数据。例如，在一个有两个NIC的ETL系统上，运行两个`gpfdist`实例（每个NIC上一个）来优化数据装载性能并且在这两个`gpfdist`之间平均划分外部表数据文件。

Figure 2. 使用带多个NIC的多个gpfdist实例的外部表



Note: 当用户提交文件给gpfdist时，用竖线 (|) 来分隔格式化好的文本。Greenplum数据库将逗号分隔的文本字符串包括在单引号或者双引号中。gpfdist必须移除这些引号来解析字符串。使用竖线分隔格式化好的文本避免了这种额外的步骤并且可以改进性能。

控制Segment并行

`gp_external_max_segs`服务器配置参数控制能同时访问单一gpfdist实例的Segment实例数量。64是默认值。用户可以设置Segment的数目为一些Segment处理外部数据文件并且一些执行其他数据库处理。在用户的Master实例的`postgresql.conf`文件中设置这个参数。

安装gpfdist

gpfdist被装载用户的Greenplum数据库Master主机安装的`$GPHOME/bin`中。在一个不同于Greenplum数据库Master或者后备Master的机器上运行gpfdist，例如在专用于ETL处理的机器。在Master或者后备Master上运行gpfdist可能会对查询执行造成性能影响。要在用户的ETL服务器上安装gpfdist，请从*Greenplum Load Tools*包中得到它并且按照它的安装指导。

启动和停止gpfdist

用户可以在用户的当前目录中或者其他指定的任何目录中启动gpfdist。默认端口是8080。

从用户的当前目录中运行，输入：

```
gpfdist &
```

要从一个不同的目录启动，应指定提供文件的目录，还可以有选择地指定运行的HTTP端口。

要在后台启动gpfdist 并且把输出的消息和错误记录在一个日志文件中：

```
$ gpfdist -d /var/load_files -p 8081 -l /home/gpadmin/log &
```

对于同一个ETL主机上的多个gpfdist实例（见[Figure 1](#)），为每一个实例使用一个不同的基础目录和端口。例如：

```
$ gpfdist -d /var/load_files1 -p 8081 -l /home/gpadmin/log1 &
$ gpfdist -d /var/load_files2 -p 8082 -l /home/gpadmin/log2 &
```

在gpfdist运行在后台时停止它：

首先找到它的进程ID：

```
$ ps -ef | grep gpfdist
```

然后杀死进程，例如（这个例子中的进程ID是3456）：

```
$ kill 3456
```

解决gpfdist的问题

Segment在运行时访问gpfdist。 确保Greenplum的Segment主机具有到gpfdist的网络访问。 gpfdist是一个Web服务器：可以通过从Greenplum阵列的每一个主机（Segment和Master）运行下列命令来测试连接：

```
$ wget http://gpfdist_hostname:port/filename
```

CREATE EXTERNAL TABLE定义必须有用于gpfdist的正确的主机名、端口以及文件名、以及相对于gpfdist提供文件的目录（gpfdist启动时指定的目录路径）的方式指定文件和路径。请见[Examples for Creating External Tables](#)。

如果用户在其系统上启动gpfdist且IPv6网络被禁用，测试一个IPv6端口时，gpfdist会显示下列警告消息。

```
[WRN gpfdist.c:2050] Creating the socket failed
```

如果对应的IPv4端口可用，gpfdist会使用该端口并且忽略对于IPv6端口的警告。要查看gpfdist测试的端口的信息，使用-v选项。

有关IPv6和IPv4网络的信息，请见操作系统的文档。

当用gpfdist或者gfdists协议读写数据时，gpfdist工具拒绝头部不包括X-GP-PROTO的HTTP请求。如果在头部没有检测到X-GP-PROTO，gpfdist会在HTTP响应头部的状态行中返回一个400错误：400 invalid request (no gp-proto)。

Greenplum数据库会在HTTP请求头部包括X-GP-PROTO以表示该请求是来自于Greenplum数据库。

如果gpfdist实用程序挂起而没有发生读取或写入活动，则可以在下次发生挂起时生成core文件以帮助调试问题。将环境变量GPFDIST_WATCHDOG_TIMER设置为gpfdist无活动等待多少秒后强制退出。设置环境变量并且gpfdist挂起后，实用程序将在指定的秒数后中止，创建core文件，并将中止信息发送到日志文件。

此示例在Linux系统上设置环境变量，以便gpfdist在300秒（5分钟）无活动后退出。

```
export GPFDIST_WATCHDOG_TIMER=300
```

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

使用外部表装载数据

□ 装载和写入非HDFS自
定义数据

□ 处理装载错误

用gupload装载数据

使用PXF访问外部数据

使用gpfdist和gupload转
换外部数据

用COPY装载数据

在单行错误隔离模式中
运行COPY优化数据装载和查询性
能

使用如INSERT和SELECT这样的SQL命令查询一个可读外部表, 和查
询一个普通数据库表是同样的方式。例如, 要从一个外部
表ext_expenses装载差旅费数据到一个数据库表
expenses_travel:

```
=# INSERT INTO expenses_travel
      SELECT * from ext_expenses where category='travel';
```

要载入所有数据到一个新的数据库表:

```
=# CREATE TABLE expenses AS SELECT * from ext_expenses;
```

Parent topic: [装载和卸载数据](#)



数据

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

使用外部表装载数据

□ 装载和写入非HDFS自定义数据

□ 处理装载错误

用gupload装载数据

使用PXF访问外部数据

使用gpfdist和gupload转换外部数据

用COPY装载数据

在单行错误隔离模式中运行COPY

优化数据装载和查询性能

Greenplum



Greenplum Database Documentation

[返回greenplum.cn](#)

[Wiki](#)

Doc Index

- [Greenplum数据库® 6.0文档](#)
- [管理员指南](#)
 - [Greenplum数据库概念](#)
 - [关于Greenplum的架构](#)
 - [关于管理和监控工具](#)
 - [关于Greenplum数据库中的并发控制](#)
 - [管理事务ID的例子](#)
 - [关于并行数据装载](#)
 - [关于Greenplum数据库中的冗余和故障切换](#)
 - [关于Greenplum数据库中的数据库统计信息](#)
 - [管理一个Greenplum系统](#)
 - [关于Greenplum数据库发布版本号](#)
 - [启动和停止Greenplum数据库](#)
 - [访问数据库](#)
 - [建立一个数据库会话](#)
 - [支持的客户端应用](#)
 - [Greenplum数据库客户端应用](#)
 - [用psql连接](#)
 - [使用PgBouncer连接池](#)
 - [数据库应用接口](#)
 - [连接问题的发现及解决](#)
 - [配置Greenplum数据库系统](#)
 - [关于Greenplum数据库的Master参数和本地参数](#)
 - [设置配置参数](#)
 - [设置本地配置参数](#)
 - [设置Master配置参数](#)
 - [设置系统级别参数](#)
 - [设置数据库级别参数](#)
 - [设置角色级别参数](#)
 - [设置会话级别参数](#)
 - [查看服务器配置参数设置](#)
 - [配置参数种类](#)
 - [启用压缩](#)
 - [启用高可用和数据持久化特征](#)
 - [Greenplum数据库高可用性概述](#)
 - [Segment镜像概述](#)
 - [Master镜像概述](#)
 - [在Greenplum数据库中启用镜像](#)
 - [启用Segment镜像](#)
 - [启用Master镜像](#)
 - [检测故障的Segment](#)
 - [检测故障Segment](#)
 - [检查日志文件](#)
 - [恢复故障Segment](#)
 - [从Segment故障中恢复](#)
 - [当一台Segment主机不可恢复时](#)
 - [恢复故障的Master](#)
 - [在恢复后还原Master镜像](#)
 - [备份和恢复数据库](#)

- [备份和恢复概述](#)
- [使用gpbackup和gprestore并行备份](#)
 - [需求和限制](#)
 - [备份或还原中包含的对象](#)
 - [执行基本备份和还原操作](#)
 - [过滤备份或恢复的内容](#)
 - [配置邮件通知](#)
 - [理解备份文件](#)
 - [使用gpbackup和gprestore创建增量备份](#)
 - [关于增量备份集](#)
 - [使用增量备份](#)
 - [将gpbackup和gprestore与BoostFS一起使用](#)
 - [安装BoostFS](#)
 - [使用BoostFS备份和恢复](#)
 - [使用gpbackup存储插件](#)
 - [使用带有gpbackup和gprestore的S3存储插件](#)
 - [备份/恢复存储插件API \(Beta版\)](#)
 - [backup_data](#)
 - [backup_file](#)
 - [cleanup_plugin_for_backup](#)
 - [cleanup_plugin_for_restore](#)
 - [plugin_api_version](#)
 - [restore_data](#)
 - [restore_file](#)
 - [setup_plugin_for_backup](#)
 - [setup_plugin_for_restore](#)
- [扩容Greenplum系统](#)
 - [系统扩容概述](#)
 - [规划Greenplum系统扩容](#)
 - [准备并增加节点](#)
 - [初始化新节点](#)
 - [重分布表](#)
 - [移除扩容Schema](#)
- [监控Greenplum系统](#)
- [日常系统维护任务](#)
- [Recommended Monitoring and Maintenance Tasks](#)
- [管理Greenplum数据库访问](#)
 - [配置客户端认证](#)
 - [使用带TLS/SSL的LDAP认证](#)
 - [使用Kerberos认证](#)
 - [为Linux客户端进行Kerberos配置](#)
 - [为Windows客户端进行Kerberos配置](#)
 - [管理角色与权限](#)
- [定义数据库对象](#)
 - [创建和管理数据库](#)
 - [创建和管理表空间](#)
 - [创建和管理SCHEMA](#)
 - [创建和管理表](#)
 - [选择表存储模型](#)
 - [对大型表分区](#)
 - [创建和使用序列](#)
 - [在Greenplum数据库中使用索引](#)
 - [创建和管理视图](#)
- [分布与倾斜](#)
- [插入、更新、和删除数据](#)
- [查询数据](#)
 - [关于Greenplum的查询处理](#)

- [关于GPORCA](#)
 - [GPORCA概述](#)
 - [启用和禁用GPORCA](#)
 - [收集根分区统计信息](#)
 - [使用GPORCA时的考虑](#)
 - [GPORCA特性和增强](#)
 - [GPORCA改变的行为](#)
 - [GPORCA的限制](#)
 - [判断被使用的查询优化器](#)
 - [关于统一多级分区表](#)
- [定义查询](#)
- [WITH查询 \(公用表表达式\)](#)
- [使用函数和操作符](#)
- [使用JSON数据](#)
- [使用XML数据](#)
- [使用全文搜索](#)
 - [关于全文搜索](#)
 - [在数据库表中搜索文本](#)
 - [控制文本搜索](#)
 - [文本搜索附加功能](#)
 - [文本搜索解析器](#)
 - [文本搜索词典](#)
 - [文本搜索配置示例](#)
 - [测试和调试文本搜索](#)
 - [文本搜索的GiST和GIN索引](#)
 - [psql支持](#)
 - [限制](#)
- [查询性能](#)
- [管理查询生成的溢出文件](#)
- [查询分析](#)
- [使用外部数据](#)
 - [定义外部表](#)
 - [file://协议](#)
 - [gpfdist://协议](#)
 - [gpfdists:// 协议](#)
 - [pxf:// 协议](#)
 - [s3:// 协议](#)
 - [使用自定义协议](#)
 - [处理外部表数据中的错误](#)
 - [创建和使用外部Web表](#)
 - [Examples for Creating External Tables](#)
 - [Example 1—Single gpfdist instance on single-NIC machine](#)
 - [Example 2—Multiple gpfdist instances](#)
 - [Example 3—Multiple gpfdists instances](#)
 - [Example 4—Single gpfdist instance with error logging](#)
 - [Example 5—TEXT Format on a Hadoop Distributed File Server](#)
 - [Example 6—Multiple files in CSV format with header rows](#)
 - [Example 7—Readable External Web Table with Script](#)
 - [Example 8—Writable External Table with gpfdist](#)
 - [Example 9—Writable External Web Table with Script](#)
 - [Example 10—Readable and Writable External Tables with XML Transformations](#)
 - [使用PXF访问外部数据](#)
 - [使用外部表访问外部数据](#)
 - [写一个外部数据的包装器](#)
 - [使用Greenplum的并行文件服务器 \(gpfdist\)](#)
 - [装载和卸载数据](#)
 - [使用外部表装载数据](#)

- [装载和写入非HDFS自定义数据](#)
 - [使用一种自定义格式](#)
 - 导入和导出固定宽度的数据
 - 例子：读取宽度固定的数据
 - [使用一种自定协议](#)
 - [处理装载错误](#)
 - 定义一个带有单行错误隔离的外部表
 - 捕捉行格式化错误并且声明拒绝极限
 - 在错误日志中查看不正确的行
 - 在表之间移动数据
 - [用gupload装载数据](#)
 - [使用PXF访问外部数据](#)
 - [使用gpfdist和gupload转换外部数据](#)
 - [用COPY装载数据](#)
 - [在单行错误隔离模式中运行COPY](#)
 - [优化数据装载和查询性能](#)
 - [从Greenplum数据库卸载数据](#)
 - 定义基于文件的可写外部表
 - 例1—Greenplum文件服务器 (gpfdist)
 - 例2—Hadoop文件系统(pxf)
 - 定义基于命令的可写外部Web表
 - 为Web或者可写外部表禁用EXECUTE
 - 使用可写外部表卸载数据
 - 使用COPY卸载数据
 - [格式化数据文件](#)
 - 格式化行
 - 格式化列
 - 表示NULL值
 - 转义
 - 在文本格式的文件中转义
 - 在CSV格式的文件中转义
 - 字符编码
 - [自定义数据访问协议实例](#)
 - 安装外部表协议
 - `gpextprotocol.c`
- [性能管理](#)
 - 管理性能
 - 性能问题的常见原因
 - [Greenplum数据库内存总览](#)
 - 管理资源
 - 用资源组进行工作负载管理
 - 使用资源队列
 - 检修性能问题

[Greenplum database 5管理员指南](#)[Greenplum database 4管理员指南](#)[FAQ](#)

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

处理装载错误

可读外部表最常被用来选择数据装载到普通数据库表中。用户使用`CREATE TABLE AS SELECT`或者`INSERT INTO`命令来查询外部表数据。默认情况下，如果数据包含错误，整个命令会失败并且数据不会被装载到目标数据库表中。

`SEGMENT REJECT LIMIT`子句允许用户隔离外部表数据中的格式错误并且继续装载正确格式的行。使用`SEGMENT REJECT LIMIT`设置一个错误阈值，就可以指定拒绝限制计数为`ROWS`的数量（默认）或者全部行的一个`PERCENT`（1-100）。

如果错误行的数量达到`SEGMENT REJECT LIMIT`, 整个外部表操作会被中止并且不会有行被处理。错误行的限制是针对每个Segment的, 而不针对整个操作。如果错误行的数量没有达到`SEGMENT REJECT LIMIT`, 该操作处理所有好的行并且抛弃错误的行, 也可以选择把错误行的格式错误记录下来。

`LOG ERRORS`子句允许用户保留错误行以便进一步的检查。关于`LOG ERRORS`子句的信息, 请见*Greenplum*数据库参考指南中的`CREATE EXTERNAL TABLE`命令。

若用户设置了`SEGMENT REJECT LIMIT`, Greenplum以单行错误隔离模式扫描外部数据。单行错误隔离模式适用于外部数据行有如多出或者缺少属性、属性数据类型错误或者非法客户端编码序列等格式错误的情况。Greenplum不会检查约束错误, 但是用户可以通过在运行时限制外部表的`SELECT`来过滤约束错误。例如, 要消除重复键错误:

```
=# INSERT INTO table_with_pkeys
    SELECT DISTINCT * FROM external_table;
```

Note: 在用`COPY`命令或者外部表装载数据时, 服务器配置参数`gp_initial_bad_row_limit`的值限制最开始的没有被正确格式化的行的数量。默认是如果前1000行都包含格式化错误, 则停止处理。关于该参数的信息请见*Greenplum*数据库参考指南。

- [定义一个带有单行错误隔离的外部表](#)
- [捕捉行格式化错误并且声明拒绝极限](#)
- [在错误日志中查看不正确的行](#)
- [在表之间移动数据](#)

Parent topic: [装载和卸载数据](#)

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

使用外部表装载数据

□ 装载和写入非HDFS自定义数据

□ 处理装载错误

用gupload装载数据

使用PXF访问外部数据

使用gpfldist和gupload转换外部数据

用COPY装载数据

在单行错误隔离模式中运行COPY

优化数据装载和查询性能

Greenplum

Greenplum的gupload工具使用可读外部表和Greenplum并行文件服务器 (gpfldist 或者gpfldists) 来装载数据。它处理并行的基于文件的外部表设置并且允许用户在一个单一配置文件中配置他们的数据格式、外部表定义以及gpfldist 或者gpfldists设置。

Note: gpfldist和gupload仅与发布它们的Greenplum数据库主要版本兼容。例如，与Greenplum Database 4.x一起安装的gpfldist实用程序不能与Greenplum Database 5.x或6.x一起使用。

Note: 如果目标表列名是保留关键字，即包含大写字母，或者包含需要引号 ("") 来标识列的任何字符，则不支持MERGE和UPDATE操作。

要使用gupload

- 确保环境已经设置好来运行gupload。一些来自于Greenplum数据库安装的依赖文件是必需的，例如gpfldist 和Python，还有访问Greenplum的Segment主机的网络。
详见*Greenplum*数据库参考指南。

- 创建装载控制文件。这是一个YAML格式的文件，它指定Greenplum数据库连接信息、gpfldist 的配置信息、外表选项和数据格式。
详见*Greenplum*数据库参考指南。

例如：

```
---
VERSION: 1.0.0.1
DATABASE: ops
USER: gpadmin
HOST: mdw-1
PORT: 5432
GUPLOAD:
  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
        - descr: text
```



从

数据库卸

```
- date: date
- FORMAT: text
- DELIMITER: '|'
- ERROR_LIMIT: 25
- LOG_ERRORS: true
OUTPUT:
- TABLE: payables.expenses
- MODE: INSERT
PRELOAD:
- REUSE_TABLES: true
SQL:
- BEFORE: "INSERT INTO audit VALUES('start',
current_timestamp)"
- AFTER: "INSERT INTO audit VALUES('end',
current_timestamp)"
```

3. 运行gupload，传入该装载控制文件。例如：

```
gupload -f my_load.yml
```

Parent topic: [装载和卸载数据](#)

数据

Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统
 - 管理Greenplum数据库访问
 - 定义数据库对象
 - 分布与倾斜
 - 插入, 更新, 和删除数据
 - 查询数据
 - 使用外部数据
 - 装载和卸载数据
 - 使用外部表装载数据
 - 装载和写入非HDFS自定义数据
 - 处理装载错误
 - 用gload装载数据
 - 使用PXF访问外部数据
- 使用gpfdist和gload转换外部数据**
- 用COPY装载数据
- 在单行错误隔离模式中运行COPY
- 优化数据装载和查询性能

gpfdist并行文件服务器允许用户设置转换，使Greenplum数据库外部表能够以CREATE EXTERNAL TABLE命令的FORMAT子句不支持的格式读取和写入文件。*input*转换以外部数据格式读取文件，并以CSV或外部表的FORMAT子句中指定的其他文本格式将行输出到gpfdist。*output*转换以文本格式从gpfdist接收行，并将它们转换为外部数据格式。

Note: gpfdist和gload仅与发布它们的Greenplum数据库主要版本兼容。例如，与Greenplum Database 4.x一起安装的gpfdist工具不能与Greenplum Database 5.x或6.x一起使用。

本节介绍了设置数据转换的任务，以便gpfdist使用Greenplum数据库不支持的格式读取或写入外部数据文件。

- [关于gpfdist转换](#)
- [确定转换方案](#)
- [编写转换](#)
- [编写gpfdist配置文件](#)
- [传输数据](#)
- [配置文件格式](#)
- [XML转换示例](#)

Parent topic: [装载和卸载数据](#)

关于gpfdist转换

要设置数据格式转换，用户需要提供一个可执行命令，该命令可以被gpfdist以包含数据的文件名调用。例如，用户可以编写一个shell脚本，该脚本在XML文件上运行XSLT转换，以输出具有以竖线(|)字符分隔的列和使用换行符分隔的行的行。

转换是在以命令行方式传递给gpfdist的YAML格式的配置文件中配置的。

如果要将外部数据加载到Greenplum数据库的表中，可以使用gload工具自动执行创建外部表的任务，运行gpfdist并将转换后的数据加载到数据库表中。

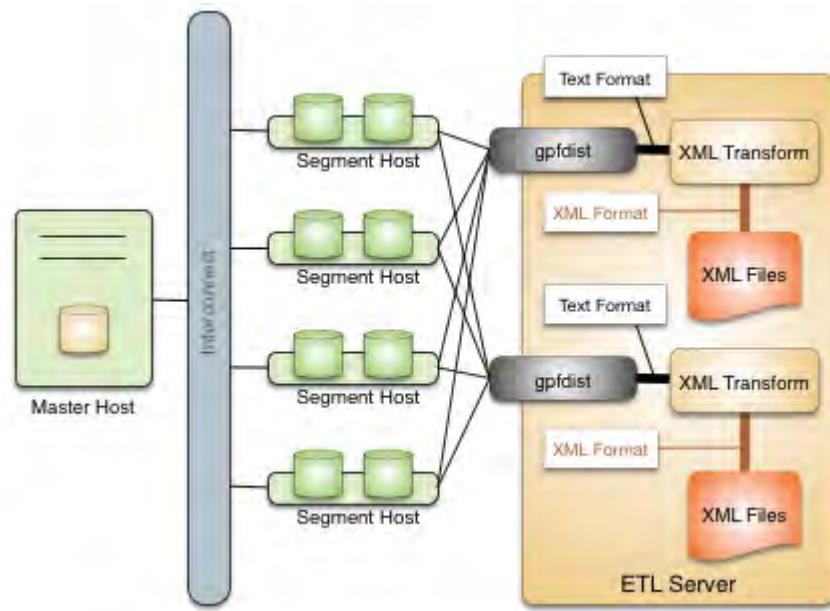


□ 从

数据库卸

从数据库中访问外部XML文件中的数据是常见的需要转换的示例。下图展示了对ETL服务器上的XML文件执行转换的`gpfdist`。

Figure 1. 使用XML转换的外部表



以下是为外部数据文件设置`gpfdist`转换的高级步骤。该过程用XML示例说明。

1. [Determine the transformation schema.](#)
2. [Write a transformation .](#)
3. [Write the gpfdist configuration file.](#)
4. [Transfer the data .](#)

确定转换方案

要为转换项目做准备：

1. 确定该项目的目标，例如索引数据、分析数据、组合数据等等。
2. 检查源文件并标记文件结构和元素名称。
3. 选择要导入的元素并且决定是否需要其他适当的限制。

例如，下面的XML文件`prices.xml` 是一个简短的包含价格记录的文件。每个价格记录包含两个字段：项目编号和价格。

```
<?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>
```

目的是把所有的数据导入到一个具有一个整数列itemnumber和一个小数列price的Greenplum数据库表中。

编写转换

转换指定从数据中抽取什么。用户可以使用适合其项目的任何创作环境和语言。对于XML转换，基于项目的目标和范围选择一种如XSLT、Joost (STX)、Java、Python或者Perl的技术。

在price的例子中，下一步是转换XML数据成一个简单的分隔成两列的格式。

```
708421|19.99
708466|59.25
711121|24.99
```

下面的称为*input_transform.stx* 的STX转换完成了这种数据转换。

```
<?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:template>
</stx:transform>
```

```

</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>

```

此STX转换声明两个临时变量itemnumber 和price，以及以下规则。

1. 当找到满足XPath表达式/prices/pricerecord的元素时，检查子元素并生成包含itemnumber变量值，一个|字符， price变量值和一个新行的输出。
2. 当找到<itemnumber>元素时，将该元素的内容存储在变量itemnumber中。
3. 当找到<price>元素后，将该元素的内容存储在变量price中。

编写gpfdist配置文件

gpfdist 配置是一个YAML 1.1文档。它指定在装载或者抽取数据时gpfdist 用来选择一种转换并应用的规则。

这个gpfdist 配置的例子包含下列项：

- 定义TRANSFORMATIONS的config.yaml文件
- config.yaml文件中引用的input_transform.sh包装器脚本
- input_transform.sh中调用的joost转换input_transform.stx

除普通的YAML规则（如文档用三个破折号开始（---））之外， gpfdist 配置必须符合下列限制：

1. VERSION设置必须出现并且值为1.0.0.1。
2. TRANSFORMATIONS设置必须出现并且包含一个或者更多个映射。
3. TRANSFORMATION中的每一个映射必须包含：
 - 一个值为'input'或者'output'的TYPE
 - 一个表明转换如何运行的COMMAND。
4. TRANSFORMATION中的每一个映射可以包含可选

的CONTENT、SAFE以及STDERR设置。

下面这个称为config.yaml的gpfdist配置适用于prices的例子。每一行开始的缩进是有意义的，它们反映了该说明的层级性。下面例子中的名称prices_input将在后面用SQL创建该表时引用。

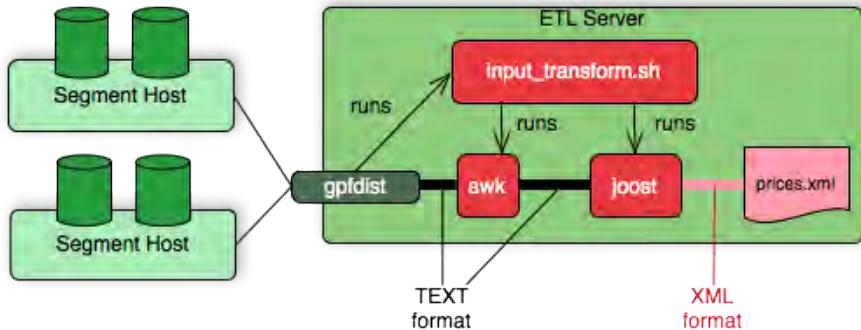
```
---
VERSION: 1.0.0.1
TRANSFORMATIONS:
  prices_input:
    TYPE:      input
    COMMAND:   /bin/bash input_transform.sh %filename%
```

COMMAND设置使用了一个名为input_transform.sh的包装器脚本，它带有一个%filename%占位符。当gpfdist运行该prices_input转换时，它用/bin/bash调用input_transform.sh，并且将%filename%占位符替换为要转换的输入文件的路径。名为input_transform.sh的包装器脚本包含调用STX转换并返回其输出的逻辑。

如果使用了Joost，则必须安装Joost STX引擎。

```
#!/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文件使用带有AWK解释器的Joost STX引擎。下面的图表展示了gpfdist运行该转换的处理流程。



传输数据

使用基于相应模式的SQL语句创建目标数据库表。

对于保存已加载数据的Greenplum数据库表没有特殊要求。在prices示例中，以下命令创建要加载数据的prices表。

```
CREATE TABLE prices (
    itemnumber integer,
    price      decimal
)
DISTRIBUTED BY (itemnumber);
```

接下来，使用其中一种方法用gpfldist转换数据。

- GPOLOAD只支持输入转换，但是在很多情况下更容易实现。
- INSERT INTO SELECT FROM支持输入和输出转换，但是会暴露更多细节。

用GPOLOAD转换

Greenplum数据库gupload工具使用gpfldist并行文件服务器和YAML格式的配置文件来编排数据加载操作。gupload自动执行以下任务：

- 在数据库中创建可读的外部表。
- 使用包含转换的配置文件启动gpfldist实例。
- 运行INSERT INTO *table_name* SELECT FROM *external_table*以加载数据。
- 删除外部表定义。

Transforming data with `gload` 转换数据要求出现在`gload`控制文件的INPUT节中的设置TRANSFORM和TRANSFORM_CONFIG。

有关在`gload`控制文件中这些设置的语法和布置，请见*Greenplum数据库参考指南*。

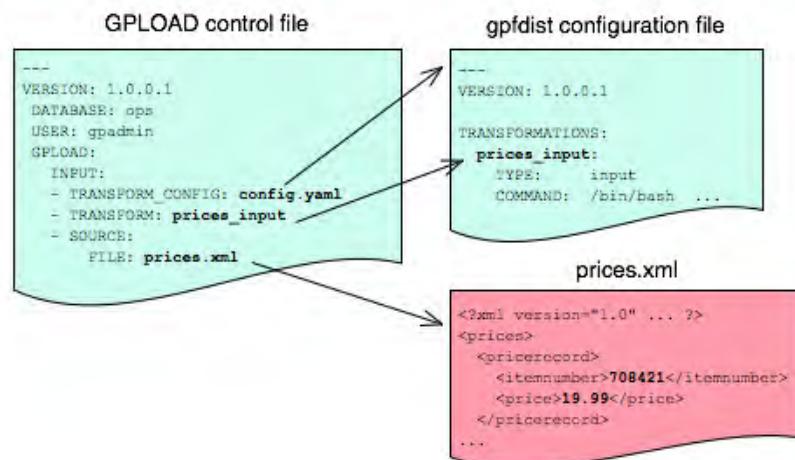
- TRANSFORM_CONFIG指定gpfdist配置文件的名称。
- The TRANSFORM设置表示TRANSFORM_CONFIG中提到的文件中描述的转换名称。

```
---
VERSION: 1.0.0.1
DATABASE: ops
USER: gpadmin
GLOAD:
  INPUT:
    - TRANSFORM_CONFIG: config.yaml
    - TRANSFORM: prices_input
    - SOURCE:
      FILE: prices.xml
```

转换名称必须出现在两个位置：在gpfdist配置文件的TRANSFORM设置中以及在TRANSFORM_CONFIG节提到的文件的TRANSFORMATIONS节中。

在`gload`控制文件中，可选的参数MAX_LINE_LENGTH指定被传递给`gload`的XML转换数据中一行的最大长度。

下面的图表展示了`gload`控制文件、gpfdist配置文件以及XML数据文件之间的关系。



用INSERT INTO SELECT FROM转换

使用此加载方法，用户可以执行gload自动执行的每个任务。用户启动gpfdist，创建一个外部表，加载数据，然后通过删除表并停止gpfdist来清理。

在CREATE EXTERNAL TABLE定义的LOCATION子句中指定转换。例如，在下面的命令中转换被显示为粗体（使用命令gpfdist -c config.yaml先运行gpfdist）。

```
CREATE READABLE EXTERNAL TABLE prices_readable (LIKE
prices)
  LOCATION
('gpfdist://hostname:8080/prices.xml#transform=prices_input')
  FORMAT 'TEXT' (DELIMITER '|')
  LOG ERRORS SEGMENT REJECT LIMIT 10;
```

在上面的命令中，把hostname改成实际的主机名。prices_input来自于gpfdist配置文件。

下面的查询装载数据到prices表中。

```
INSERT INTO prices SELECT * FROM prices_readable;
```

配置文件格式

gpfdist配置文件使用YAML 1.1文档格式并且实现了一种定义转换参数的方案。配置文件必须是一个合法的YAML文档。

gpfdist程序会按照顺序处理该文档并且使用缩进（空格）来判断文档的层次以及小节之间的关系。空白的使用很重要。不要使用空白来进行格式化也不要使用制表符。

下面是一个配置文件的基本结构。

```
---
VERSION: 1.0.0.1
TRANSFORMATIONS:
  transformation_name1:
    TYPE:      input | output
    COMMAND:   command
    CONTENT:   data | paths
    SAFE:      posix-regex
    STDERR:    server | console
```

```

transformation_name2:
  TYPE:      input | output
  COMMAND:   command
...

```

VERSION

必需。gpfdist配置文件方案的版本。当前版本是1.0.0.1。

TRANSFORMATIONS

必需。开始转换说明小节。一个配置文件必须有至少一个转换。

当gpfdist收到一个转换请求时，它会在这个小节查找具有匹配的转换名称的项。

TYPE

必须。指定转换的方向。值是input或者output。值为 `input` 或者 `output`.

- `input`: gpfdist 把转换处理的标准输出当做是要载入到Greenplum数据库的一个记录流。
- `output` : gpfdist 把转换处理的标准输入当作是来自于Greenplum数据库的一个记录流并且写出到适当的输出。

COMMAND

必须。指定将被gpfdist执行来做转换的命令。

对于输入转换，gpfdist调用CONTENT设置中指定的命令。该命令应该会以适当的方式打开底层文件并且为每一行产生一个TEXT行以载入到Greenplum数据库。输入转换决定整个内容是应该被转换为一个行还是多个行。

对于输出转换，gpfdist调用在CONTENT设置中指定的命令。输出命令应该会以适当的方式打开底层文件并且向其中写入。输出转换决定转换好的输出的最终安置。

CONTENT

可选。值是`data` 和 `paths`。默认值是`data`。

- 当CONTENT指定`data`时，COMMAND小节中的文本`%filename%`会被替换为要读写的文件路径。
- 当CONTENT指定`paths`时，COMMAND小节中的文本`%filename%`会被替换为包含要读写的文件列表的临时文件路径。

下面是一个COMMAND小节的例子，它展示了被替换掉的文本`%filename%`。

```
COMMAND: /bin/bash input_transform.sh %filename%
```

SAFE

可选。一个POSIX 正则表达式，路径必须匹配它才能被传递给转换。在担心传递给命令的路径被注入或者被不正确解读时指定SAFE。默认对路径没有限制。

STDERR

可选。值是server 和 console。

这个设置指定如何处理来自于转换的标准错误输出。默认值server，指定 gpfdist 将捕捉来自于转换的标准错误输出放在一个临时文件中，并且把该文件的前8k字节发送到Greenplum数据库作为一个错误消息。该错误消息将作为一个SQL错误出现。Console指定gpfdist 不会重定向或者转换来自于转换的标准错误输出。

XML转换示例

下面的例子展示了对不同类型的XML数据以及STX转换的完整处理。与这些例子相关的文件和详细用法在GitHub仓库github.com:greenplum-db/gpdb [gpMgmt/demo/gpfdist_transform](#) 目录中。在运行这些例子之前，请阅读开始之前小节中的README文件。README文件解释了如何下载这些例子中用到的示例数据文件。

基于命令的外部Web表

一个shell命令或者脚本的输出可定义基于命令的Web表数据。在CREATE EXTERNAL WEB TABLE的EXECUTE子句中指定该命令。EXECUTE子句会在指定的Master以及（一个或者多个）Segment主机上运行该shell命令或脚本。该命令或脚本必须位于EXECUTE子句中定义的主机上。

默认情况下，当活动Segment有输出行要处理时，该命令运行在Segment主机上。例如，如果每一个Segment主机运行四个有输出行要处理的主Segment实例，该命令在每个Segment主机上会运行四次。用户可以有选择地限制执行Web表命令的Segment实例的数量。ON子句中Web表定义里包括的所有Segment会并行运行该命令。

用户在外部表定义中指定的命令从数据库中执行并且不能访问来

自.bashrc 或者 .profile的环境变量。可在EXECUTE子句中设置环境变量。例如：

```
=# CREATE EXTERNAL WEB TABLE output (output text)
  EXECUTE 'PATH=/home/gpadmin/programs; export PATH;
myprogram.sh'
  FORMAT 'TEXT';
```

脚本必须对gpadmin用户可执行并且在Master或者Segment主机上位于同一位置。

下列命令定义一个运行脚本的Web表。该脚本在每一台有需要处理输出行的Segment的Segment主机上运行。

```
=# CREATE EXTERNAL WEB TABLE log_output
  (linenum int, message text)
  EXECUTE '/var/load_scripts/get_log_data.sh' ON HOST
  FORMAT 'TEXT' (DELIMITER '|');
```

IRS MeF XML 文件 (在demo目录中)

这个例子展示了使用一种Joost STX转换装载一份IRS Modernized eFile tax return数据样例。该数据是一种复杂的XML文件形式。

美国国内税务署 (IRS) 在对XML进行了可观的投资并且指定在其现代化电子文件 (MeF) 系统中使用它。在MeF中，每一份纳税申报表都是一个具有深层次结构的XML文档，该层次结构仔细地反映了底层纳税代码的特定形式。

XML、XML Schema和样式表在他们的数据表达和业务工作流中扮演了重要角色。实际的XML数据被从一个附加了MIME “transimission file”消息的ZIP文件中抽取。更多有关MeF的信息，请见IRS网站上的[Modernized e-File \(Overview\)](#)。

样例XML文档*RET990EZ_2006.xml* 大约有350KB大小，它有两个元素：

- ReturnHeader
- ReturnData

<ReturnHeader> 元素包含了有关纳税申报表的一般细节，例如纳税人的姓名、纳税年份以及preparer。<ReturnData> 元素包含了多个小节，它们含有关于纳税申报表及相关计划表的特定细节。

下面是该XML文件的一个被删节过的例子。

```

<?xml version="1.0" encoding="UTF-8"?>
<Return returnVersion="2006v2.0"
  xmlns="https://www.irs.gov/efile"
  xmlns:efile="https://www.irs.gov/efile"
  xsi:schemaLocation="https://www.irs.gov/efile"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ReturnHeader binaryAttachmentCount="1">
    <ReturnId>AAAAAAAAAAAAAAA</ReturnId>
    <Timestamp>1999-05-30T12:01:01+05:01</Timestamp>
    <ReturnType>990EZ</ReturnType>
    <TaxPeriodBeginDate>2005-01-01</TaxPeriodBeginDate>
    <TaxPeriodEndDate>2005-12-31</TaxPeriodEndDate>
    <Filer>
      <EIN>011248772</EIN>
      ... more data ...
    </Filer>
    <Preparer>
      <Name>Percy Polar</Name>
      ... more data ...
    </Preparer>
    <TaxYear>2005</TaxYear>
  </ReturnHeader>
  ... more data ..

```

目标是把所有数据导入到一个Greenplum数据库。首先，将该XML文档转换为文本形式，其中的新行已被转义过并且有两个列：ReturnId和末尾的一个用于整个Mef纳税申报表的列。例如：

```
AAAAAAAAAAAAAAA | <Return returnVersion="2006v2.0" ...
```

载入数据到Greenplum数据库。

WITSML™ 文件 (在demo目录中)

这个例子展示了使用一种Joost STX转换载入描述一个石油钻塔的样例数据。该数据是一种从energistics.org下载的复杂的XML文件格式。

Wellsite Information Transfer Standard Markup Language (WITSML™)是一种石油工业发起的为技术和软件提供开放的、非所有权的、标准的接口，以便在石油公司、服务公司、钻探承包商和监管代理之间分享信息。更多有关WITSML™的信息请见<http://www.energistics.org/>。

石油钻塔的信息由一个顶层的<rigs>元素和多个子元素（例如<documentInfo>, <rig>,等）构成。下面从该文件中摘录的片

段展示了`<rig>`标签中的信息的类型。

```

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="../stylesheets/rgi.xsl"
type="text/xsl" media="screen"?>
<rigs
  xmlns="http://www.energistics.org/schemas/131"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.energistics.org/schemas/131
  ../obj_rig.xsd"
  version="1.3.1.1">
  <documentInfo>
    ... misc data ...
  </documentInfo>
  <rig uidWell="W-12" uidWellbore="B-01" uid="xr31">
    <nameWell>6507/7-A-42</nameWell>
    <nameWellbore>A-42</nameWellbore>
    <name>Deep Drill #5</name>
    <owner>Deep Drilling Co.</owner>
    <typeRig>floater</typeRig>
    <manufacturer>Fitsui Engineering</manufacturer>
    <yearEntService>1980</yearEntService>
    <classRig>ABS Class A1 M CSDU AMS ACCU</classRig>
    <approvals>DNV</approvals>
    ... more data ...
  
```

目标是把这个钻塔的信息导入到Greenplum数据库。

样例文档*rig.xml*的尺寸大约是11KB。输入不包含制表符，因此相关信息可以被转换成用竖线 (|) 分隔的记录。

W-12|6507/7-A-42|xr31|Deep Drill #5|Deep Drilling
Co.|John Doe|John.Doe@example.com|

有这些列：

- well_uid text, -- e.g. W-12
- well_name text, -- e.g. 6507/7-A-42
- rig_uid text, -- e.g. xr31
- rig_name text, -- e.g. Deep Drill #5
- rig_owner text, -- e.g. Deep Drilling Co.
- rig_contact text, -- e.g. John Doe
- rig_email text, -- e.g. John.Doe@example.com
- doc xml

然后，载入该数据到Greenplum数据库中。

Greenplum数据库® 6.0文档

 管理员指南

COPY FROM将文件或标准输入中的数据复制到表中，并将数据附加到表内容中。COPY是非并行的：使用Greenplum master实例在单个进程中加载数据。建议仅对非常小的数据文件使用COPY。

Master主机上的postgres进程必须可以访问COPY源文件。指定相对于Master主机上的数据目录的COPY源文件名，或指定绝对路径。

Greenplum使用客户端和master服务器之间的连接从STDIN或STDOUT复制数据。

Parent topic: [装载和卸载数据](#)

从文件装载

COPY命令要求postgres后端打开指定的文件，读取文件并将其附加到表中。为了能够读取文件，后端需要具有对文件的可读权限，并且必须使用master主机上的绝对路径或master数据目录的相对路径来指定文件名。

```
COPY table_name FROM /path/to/filename;
```

从STDIN装载

为避免在加载数据之前将数据文件复制到master主机的问题，COPY FROM STDIN使用标准输入通道并将数据直接提供给postgres后端。COPY FROM STDIN命令启动后，后端将接受数据行，直到一行只包含反斜杠句点 (\.)。

```
COPY table_name FROM STDIN;
```

在psql中使用\copy装载数据

不要将psql\copy命令与COPY SQL命令混淆。 \copy调用常规COPY FROM STDIN并将数据从psql客户端发送到后端。因此，任何文件都

psql

必须驻留在运行客户端的主机上，并且必须可由运行客户端的用户访问。

为避免在加载数据之前将数据文件复制到master主机的问题，COPY FROM STDIN使用标准输入通道并将数据直接提供给postgres后端。COPY FROM STDIN命令启动后，后端将接受数据行，直到一行只包含反斜杠句点 (\.)。psql将所有这些包装到handy \copy 命令中

```
\copy table_name FROM filename;
```

输入格式

COPY FROM接受FORMAT参数，该参数指定输入数据的格式。可能的值为TEXT，CSV（逗号分隔值）和BINARY。

```
COPY table_name FROM /path/to/filename WITH (FORMAT csv);
```

FORMAT csv将读取逗号分隔值。默认情况下，FORMAT text使用制表符来分隔值，DELIMITER选项将不同的字符指定为值分隔符。

```
COPY table_name FROM /path/to/filename WITH (FORMAT text,  
DELIMITER '|');
```

默认情况下，使用默认客户端编码，可以使用ENCODING选项更改此编码。这对于来自其他操作系统的数据会非常有用。

```
COPY table_name FROM /path/to/filename WITH (ENCODING  
'latin1');
```



Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统
 - 管理Greenplum数据库访问
 - 定义数据库对象
 - 分布与倾斜
 - 插入, 更新, 和删除数据
 - 查询数据
 - 使用外部数据
 - 装载和卸载数据

使用外部表装载数据

- 装载和写入非HDFS自定义数据
- 处理装载错误

用gupload装载数据

使用PXF访问外部数据

使用gpfdist和gupload转换外部数据

用COPY装载数据

在单行错误隔离模式中运行COPY

优化数据装载和查询性能

使用下列技巧来帮助优化数据装载和后续的查询性能。

- 在装载数据到现有表中之前删掉索引。
在现有数据上创建一个数据比随着每个行被装载而增量更新索引更快。用户可以临时增加maintenance_work_mem服务器配置参数来帮助加速CREATE INDEX命令，不过装载性能会被影响。只有在系统中没有活动用户时才删除并重建索引。
- 在装载数据到新表中时，最后创建索引。创建表，装载数据，然后才创建任何需要的索引。
- 在装载数据后运行ANALYZE。如果一个表中的数据受到了显著的更改，运行ANALYZE或者VACUUM ANALYZE来为查询优化器更新表统计信息。当前的统计信息能确保优化器在查询规划时做出最好的决定并且避免由于不准确或者不存在的统计信息造成的糟糕性能。
- 在装载错误后运行VACUUM。如果装载错误没有运行在单行错误隔离模式中，操作会在第一个错误处停止。目标表会包含在错误发生前已经装载的行。用户无法访问这些行，但是它们仍占据磁盘空间。使用VACUUM命令来恢复被浪费的空间。

Parent topic: [装载和卸载数据](#)



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

使用外部表装载数据

□ 装载和写入非HDFS自
定义数据

□ 处理装载错误

用gupload装载数据

使用PXF访问外部数据

使用gpfdist和gupload转
换外部数据

用COPY装载数据

在单行错误隔离模式中
运行COPY优化数据装载和查询性
能

一个可写外部表允许用户从其他数据库表选择行并且把这些行输出到文件、命名管道、应用或者作为Greenplum并行MapReduce计算的输出目标。用户可以定义基于文件的和基于Web的可写外部表。

这一主题描述了如何从Greenplum数据库中使用并行卸载（可写外部表）和非并行卸载（COPY）来卸载数据。

- [定义基于文件的可写外部表](#)
- [定义基于命令的可写外部Web表](#)
- [使用可写外部表卸载数据](#)
- [使用COPY卸载数据](#)

Parent topic: [装载和卸载数据](#)







Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

使用外部表装载数据

□ 装载和写入非HDFS自
定义数据

□ 处理装载错误

用gupload装载数据

使用PXF访问外部数据

使用gpfdist和gupload转
换外部数据

用COPY装载数据

在单行错误隔离模式中
运行COPY优化数据装载和查询性
能

在使用装载和卸载数据的Greenplum工具时，用户必须指定数据是怎么格式化的。COPY、CREATE EXTERNAL TABLE和gupload都有子句允许用户指定其数据的格式。数据可以是被划定的文本（TEXT）或者逗号分隔的值（CSV）格式。外部数据必须被正确地格式化以被Greenplum数据库读取。这一主题解释了Greenplum数据库所期待的数据文件的格式。

• [格式化行](#)• [格式化列](#)• [表示NULL值](#)• [转义](#)• [字符编码](#)

Parent topic: [装载和卸载数据](#)



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

使用外部表装载数据

□ 装载和写入非HDFS自
定义数据

□ 处理装载错误

用gupload装载数据

使用PXF访问外部数据

使用gpfldist和gupload转
换外部数据

用COPY装载数据

在单行错误隔离模式中
运行COPY优化数据装载和查询性
能

Greenplum

下面是Greenplum数据库自定义数据访问协议的API。例子协议的实现[gpextprotocol.c](#) 用C语言编写, 它展示了如何使用这些API。有关访问一种自定义数据访问协议的信息请见[使用一种自定协议](#)。

```
/* ----- Read/Write function API -----*/
CALLED_AS_EXTPROTOCOL(fcinfo)
EXTPROTOCOL_GET_URL(fcinfo)(fcinfo)
EXTPROTOCOL_GET_DATABUF(fcinfo)
EXTPROTOCOL_GET_DATALEN(fcinfo)
EXTPROTOCOL_GET_SCANQUALS(fcinfo)
EXTPROTOCOL_GET_USER_CTX(fcinfo)
EXTPROTOCOL_IS_LAST_CALL(fcinfo)
EXTPROTOCOL_SET_LAST_CALL(fcinfo)
EXTPROTOCOL_SET_USER_CTX(fcinfo, p)

/* ----- Validator function API -----*/
CALLED_AS_EXTPROTOCOL_VALIDATOR(fcinfo)
EXTPROTOCOL_VALIDATOR_GET_URL_LIST(fcinfo)
EXTPROTOCOL_VALIDATOR_GET_NUM_URLS(fcinfo)
EXTPROTOCOL_VALIDATOR_GET_NTH_URL(fcinfo, n)
EXTPROTOCOL_VALIDATOR_GET_DIRECTION(fcinfo)
```

说明

这个协议对应于[使用一种自定协议](#)中描述的例子。源代码文件名称和共享对象名称分别是[gpextprotocol.c](#)和[gpextprotocol.so](#)。

该协议具有下列性质:

- 为该协议定义的名称是myprot。
- 该协议具有如下的简单形式: 协议名和一个路径, 用://分隔。
myprot:// path
- 实现了三个函数:
 - myprot_import() 是一个读函数
 - myprot_export() 是一个写函数
 - myprot_validate_urls() 是一个验证函数

当该协议被在数据库中创建和声明时, CREATE PROTOCOL语句中会引用这些函数。

这个例子的实现[gpextprotocol.c](#) 使用fopen()和fread()来模拟一个读取本地文件的简单协议。但实际上, 该协议可实现诸如通过网络远



从

数据库卸

程连接到某个进程这样的功能。

- [安装外部表协议](#)

Parent topic: [装载和卸载数据](#)

Greenplum数据库® 6.0文档

□ 管理员指南

Greenplum数据库® 6.0文档

□ 管理员指南

- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象
- 分布与倾斜
- 插入, 更新, 和删除数据
- 查询数据
- 使用外部数据
- 装载和卸载数据
- 性能管理

管理性能

性能问题的常见原因

Greenplum数据库内存总览

□ 管理资源

检修性能问题

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

这一节解释了常见性能问题的排查以及对这些问题可能的解决方案。

Parent topic: [性能管理](#)

识别硬件和Segment失效

Greenplum数据库的性能取决于它所运行的硬件和IT基础设施。Greenplum数据库由多台服务器（主机）构成，它们作为一个紧密的系统（阵列）一起工作。作为诊断性能的第一步，应确保所有的Greenplum数据库的Segment都在线。Greenplum数据库的性能将和阵列中最慢的那一台主机相同。CPU利用、内存管理、I/O处理或者网络负载方面的问题都会影响性能。常见的与硬件相关的问题有

- 磁盘失效 – 尽管在使用RAID时单一磁盘失效不会剧烈的影响数据库性能，但磁盘重新同步确实会在有失效磁盘的主机上消耗资源。`gpcheckperf` 工具可以帮助发现有磁盘I/O问题的Segment主机。
- 主机失效 – 当一台主机离线时，该主机上的Segment就不可操作。这意味着阵列中的其他主机必须执行两倍于它们通常的负载，因为它们运行着主要Segment和多个镜像。如果没有启用镜像，服务就会中断。为恢复失效的Segment也需要临时中断服务。`gpstate` 工具可以帮助发现失效的Segment。
- 网络失效 – 一块网卡、一台交换机或者DNS服务器的失效都可能让Segment宕掉。如果在Greenplum阵列中无法解析主机名或者IP地址，这就表明它们是Greenplum数据库中的Interconnect错误。`gpcheckperf` 可以帮助发现出现网络问题的Segment主机。
- 磁盘容量 – Segment主机上的磁盘容量应该永远不超过70%充满。Greenplum数据库需要一些空闲空间来做运行时处理。要回收已删除行占用的磁盘空间，可以在装载或者更新后运行VACUUM。`gp_toolkit` 管理方案中有很多视图可用来检查分布式数据库对象的尺寸。
有关检查数据库对象尺寸和磁盘空间的信息请见 *Greenplum* 数据库参考指南。

管理负载

一个数据库系统的CPU容量、内存和磁盘I/O资源是有限的。当多个负载竞争访问这些资源时，数据库性能就会受到影响。负载管理能在符



合多变的业务需求的同时最大化系统吞吐。Greenplum数据库提供了资源队列和资源组来协助管理这些系统资源。

资源队列和资源组限制了特定队列或组的资源使用量和并行查询总数，管理员可以控制并行用户查询，防止系通过载。更多有关资源队列和资源组的信息，包括如何选择合适的管理模式，请参见[管理资源](#)。

Greenplum数据库管理员应该在业务时段之后运行维护负载，例如数据装载和 VACUUM ANALYZE 操作，不要与数据库用户竞争系统资源，在低使用率时段执行管理任务。

避免竞争

当多个用户或者负载尝试以冲突的方式使用系统时，竞争就会发生。例如，当两个事务尝试同时更新一个表时会发生竞争。一个寻求表级或行级锁的事务将无限等待冲突的锁被释放。应用不应该保持事务打开很长时间，例如，在等待用户输入时。

维护数据库统计信息

Greenplum数据库使用一种依赖于数据库统计信息的基于代价的查询优化器。准确的统计信息让查询优化器更好地估计一个查询检索的行数，以便选择最有效的查询计划。如果没有数据库统计信息，查询优化器就不能估计将返回多少记录。优化器并不假设它有足够的内存来执行特定的操作，例如聚集，因此它会采取最保守的行动并且通过读写磁盘来做这些操作。这比在内存中做要慢很多。ANALYZE会收集查询优化器需要的数据库相关的统计信息。

Note: 在使用GPORCA执行SQL命令时，如果通过在该命令引用的一列或者多列上收集统计信息可以改进命令的性能，Greenplum数据库会发出一个警告。该警告在命令行上发出并且信息会被加入到Greenplum数据库的日志文件中。有关在表列上收集统计信息的内容请见*Greenplum Database* 数据库参考指南中的ANALYZE命令

识别查询计划中的统计信息问题

在使用EXPLAIN 或 EXPLAIN ANALYZE解释一个查询的计划之前，先熟悉一下有助于帮助发现统计信息问题的数据。在计划中检查下列不准确统计信息的指示器：

- 优化器的估计接近于现实吗？运行 EXPLAIN ANALYZE 并且看看优化器估计的行数是否和查询操作返回的行数接近。
- 计划中是否比较早地应用了选择性谓词？最具选择性的过滤条件应该在计划中早早地被应用，这样会有较少的行在计划树中向上移动。
- 优化器是否选择了最好的连接顺序？在一个查询连接多个表时，确保优化器选择最具选择性的连接顺序。消除行数最多的连接应该在计划中被最早执行，这样会有较少的行在计划树中向上移动。

更多有关阅读查询计划的信息请见[查询分析](#)

调整统计收集

下列配置参数控制统计信息收集采样的数据量：

- default_statistics_target

这些参数控制系统层面的统计信息采样。最好只对查询谓词中最频繁使用的列采样增加的统计信息。用户可以对一个特定的列采用下面的命令调整统计信息：

```
ALTER TABLE...SET STATISTICS
```

例如：

```
ALTER TABLE sales ALTER COLUMN region SET STATISTICS 50;
```

这等效于为特定列增加default_statistics_target。后续的ANALYZE操作接着将为该列收集更多统计数据并且结果就是会产生更好的查询计划。

优化数据分布

当用户在Greenplum数据库中创建一个表时，用户必须声明一个分布键，它允许在系统中所有的Segment上均匀地分布数据。因为Segment会以并行的方式工作在查询上，Greenplum数据库将总是和最慢的Segment速度相同。如果数据不平衡，拥有更多数据的Segment将更慢地返回它们的结果，因此会拖慢整个系统。

优化数据库设计

很多性能问题可以通过数据库设计改进。检查用户的数据库设计并且考虑以下几点：

- 模式是否反映了数据被访问的方式？
- 较大的表是否能被分解成分区？
- 是否在使用尽可能小的数据类型来存储列值？
- 用于连接表的列是否为相同的数据类型？
- 索引有没有被使用？

Greenplum数据库最大量限制

为了帮助优化数据库设计，回顾一下Greenplum数据库支持的最大量限制：

Table 1. Greenplum数据库的最大量限制

维度	限制
数据库尺寸	无限
表尺寸	无限，每个Segment的每个分区是128TB
行大小	1.6 TB (1600 列 * 1 GB)
域尺寸	1 GB
每个表的行	281474976710656 (2^48)
每个表/视图的列	1600
每个表的索引	无限
每个索引的列	32
每个表的表级约束	无限
表名长度	63 字节 (受name数据类型限制)

这里列出的“无限”维度本质上不受Greenplum数据库的限制。不过，它们实际受限于可用的磁盘空间和内存/交换空间。当这些值异乎寻常地大时，性能可能会受到损害。

Note:

可以同时存在的对象(表、索引以及视图，但不包括行)数有一个最大

限制。该限制为4294967296 (2^{32}).

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

□ 性能管理

管理性能

性能问题的常见原因

Greenplum数据库内存总览

□ 管理资源

检修性能问题

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

在Greenplum数据库系统中，内存是一项关键的资源，对其的高效使用可以保证优秀的性能和输出，这一节描述了Segment主机内存是在Segment之间的分配方式，以及管理员可用的配置方法。

Greenplum数据库的每个Segment主机都会运行多个PostgreSQL的实例，它们会共享主机的内存。Segment在同时进行查询时都会使用同一个配置，并同时消耗相近的量的内存，CPU和磁盘输入/输出。

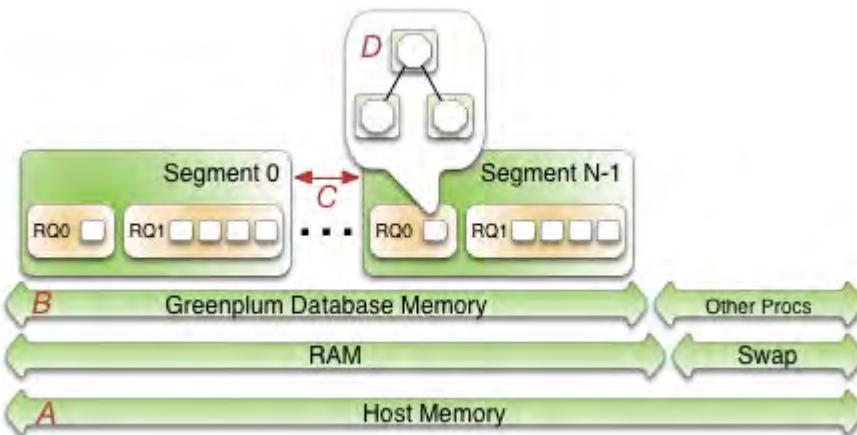
为了得到最佳的查询输出，内存配置需要精细的管理。在Greenplum数据库中，从操作系统参数，到利用资源队列和资源组管理资源，每个层级都有对应的内存配置，可以设置单个查询分配到的内存大小。

Segment主机内存

在Greenplum数据库的Segment主机上，所有正在运行的进程会共享可用的主机内存，其中包括了操作系统、Greenplum数据库Segment实例，以及其他应用进程。管理员必须要判断并有效地分配Greenplum数据库进程和非Greenplum数据库进程使用的内存大小。与此同时，监控Greenplum数据库或其他进程是否有异常的内存消耗行为也是很重要的。

以下图例显示了Greenplum数据库的Segment主机是启用基于资源队列的资源管理时是如何消耗内存的。

Figure 1. Greenplum数据库Segment主机内存



让我们从下往上看，标记了A的一行表示总主机内存，A行上面的一行表示总主机内存被分成了物理RAM和Swap空间。

标记了B表示所有内存会由Greenplum数据库和其他进程共享，非Greenplum数据的进程包括操作系统以及其他应用，例如系统监

控Agent。一些应用会使用大量内存，以至于用户可能不得不调整每个Greenplum数据主机所运行的Segment数量，或是每个Segment消耗的内存量。

每个Segment(C)将会使用等量 (B)。

每个Segment中的当前资源管理模式——资源队列或资源组——控制了运行一个SQL表达式所使用的的内存，这样的配置使得用户可以将业务需求转化为Greenplum数据库系统中的决策策略，以防止可能造成性能下降的查询行为，关于资源组和资源队列的详情，请参阅[管理资源](#)。

配置Segment主机内存的选项

主机内存将会由Segment主机上所有的应用共享。你可以通过以下方式来配置主机内存的分配方式：

- 增加节点的RAM以增加物理内存。
- 配置Swap空间来增加虚拟内存。
- 设置核心参数`vm.overcommit_memory`和`vm.overcommit_ratio`以配置操作系统处理对内存需求量较大的请求时的方法。

物理RAM和操作系统的配置通常由平台团队和系统管理员管理。有关推荐的核心参数请参考*Greenplum* 数据库安装指南

预留给操作系统和其他进程的内存量是取决于你的负载。推荐的操作台系统内存大小是32GB，但如果Greenplum数据库中存在太多的并发的话，可将内存增加到64GB。SLAB是使用最多操作系统内存的进程，其内存使用量也会随着Greenplum数据库并发和使用的套接字数量的增加而增加。

`vm.overcommit_memory`核心参数应该保持为2，这是对Greenplum来说最安全的数值。

`vm.overcommit_ratio`核心参数设置了应用进程能够使用的RAM的百分比，剩下的内存就会留给操作系统。Red Hat系统默认的值是50 (50%)，如果将该参数设置得过高，可能会导致Segment主机失效或数据库失效，一般来说比较保险的默认数值是50。该参数如果设置得过低会减少并发数，以及以减少Greenplum数据可用内存为代价来同时处理的查询行为的复杂程度。在增加`vm.overcommit_ratio`的数值的时候，要记住还要为操作系统的活动保留一定量的内存。

在启用了基于资源组的资源管理时，对`vm.overcommit_ratio`的配置方

法

当启用了基于资源组的资源管理时，操作系统的`vm.overcommit_ratio`默认数值其实就已经比较合适了，如非必要，可以不改动。如果你的内存使用率太低了，那么就增加该参数，反之亦然。

在启用了基于资源队列的资源管理时，对`vm.overcommit_ratio`的配置方法

计算在启用了基于资源组的资源管理下`vm.overcommit_ratio`的数值需要几个数据，首先要用`gp_vmem_rq`，确认Greenplum数据库进程可用的内存，计算规则如下：

```
gp_vmem_rq = ((SWAP + RAM) - (7.5GB + 0.05 * RAM)) / 1.7
```

此处的 SWAP 是主机上的Swap空间，以GB为单位，而RAM则是主机上安装的RAM，以GB为单位。当启用了基于资源队列的资源管理系统时，使用`gp_vmem_rq`来计算 `vm.overcommit_ratio`的数值，计算规则如下：

```
vm.overcommit_ratio = (RAM - 0.026 * gp_vmem_rq) / RAM
```

配置Greenplum数据库内存

Greenplum数据库内存指的是所有Greenplum数据库Segment实例可用的内存大小。

在你搭建Greenplum数据库集群时，你需要确认每个主机运行的主Segment数量，以及每个Segment消耗的内存大小。基于不同的CPU核心数、物理RAM大小以及负载特性，通常一台主机上会运行4到8个Segment。在启用了Segment镜像的情况下，内存必须按照该主机上可运行的最大主Segment数量来分配，以防出现系统故障。当你使用默认分组镜像的配置时，一台Segment主机故障将会加倍其镜像Segment所在的主机上的活跃主Segment数量。将每台主机上的镜像分散到其他主机上的镜像配置可以降低最大值，允许每台Segment分配到更多的可用内存。例如，如果你使用了区组 (block) 镜像配置，每个区组有4个主机，每个主机上有8个主Segment，单个主机的故障会导致区组中其他主机出现最多11个活跃主Segment，而使用默认组镜像配置的话则会产生最多16个。

在启用了基于资源组的资源管理系统下分配Segment内存

当启用了基于资源组的管理系统时，每个Segment主机上的每个Segment所分配到的内存等于Greenplum可用的内存乘以`gp_resource_group_memory_limit` 服务器配置参数，然后除以主机上活跃的主Segment数量，以下方程式就是在启用了基于资源组的资源管理系统时计算Segment内存的方式。

```
rg_perseg_mem = ((RAM * (vm.overcommit_ratio / 100) + SWAP)
                  * gp_resource_group_memory_limit) /
                  num_active_primary_segments
```

资源组带来的额外配置参数使用户可以进一步控制和精炼查询行为所分配到的内存大小。

在启用了基于资源队列的资源管理系统下分配Segment内存

当启用了基于资源队列的资源管理系统时，`gp_vmem_protect_limit` 服务器配置参数数值表示了每个Segment分配到的内存大小，该数值的预估值是通过计算所有Greenplum数据库进程可用的内存大小除以在发生故障时最大的主Segment数量得出，如果`gp_vmem_protect_limit` 的数值设置得过高，则查询可能会失败，使用以下方程式来计算`gp_vmem_protect_limit` 的安全值，其中`gp_vmem_rq`是你刚刚计算得出的数值。

```
gp_vmem_protect_limit = gp_vmem_rq /
max_acting_primary_segments
```

其中 `max_acting_primary_segments` 是指在有主机或Segment发生故障时触发镜像Segment时的最大主Segment数量。

在使用基于资源组的Greenplum数据库资源管理系统时，`gp_vmem_protect_limit` 是不会影响Segment内存大小的。

资源队列带来的额外配置参数使用户可以进一步控制和精炼查询行为所分配到的内存大小。

内存配置计算示例 Example Memory Configuration Calculations

该小节将会展示以下配置的Greenplum数据库系统的资源队列和资源组的内存计算示例：

- 总RAM = 256GB

Swap = 64GB

- 一个区组 (block) 4个主机，每个主机8个主Segment和8个镜像Segment
- 发生故障时每个主机的最大活跃主Segment数量为11

资源组示例

当Greenplum数据库启用了基于资源组的资源管理系统时，一个主机上可用的内存由系统配置的RAM和Swap空间，以及`vm.overcommit_ratio`的系统参数设置决定：

```
total_node_usable_memory = RAM * (vm.overcommit_ratio / 100) + Swap
                            = 256GB * (50/100) + 64GB
                            = 192GB
```

假设默认的`gp_resource_group_memory_limit`数值(.7)，示例中的Greenplum数据库分配到的内存大小就是：

```
total_gp_memory = total_node_usable_memory * gp_resource_group_memory_limit
                    = 192GB * .7
                    = 134.4GB
```

在Greenplum数据库的Segment主机上的一个Segment可用的内存由主机上为Greenplum预留的内存和主机上活跃的主Segment数量组成，在集群启动时：

```
gp_seg_memory = total_gp_memory / number_of_active_primary_segments
                    = 134.4GB / 8
                    = 16.8GB
```

注意，当三个镜像Segment切换为主Segment时，每个Segment的内存仍然是16.8GB，Segment主机使用的总内存大小预计为：

```
total_gp_memory_with_primaries = 16.8GB * 11 = 184.8GB
```

资源队列示例

在启用了基于资源队列的资源管理系统下的Greenplum示例数据库中，`vm.overcommit_ratio`的计算遵循以下方程式：

```
gp_vmem_rq = ((SWAP + RAM) - (7.5GB + 0.05 * RAM)) / 1.7
                = ((64 + 256) - (7.5 + 0.05 * 256)) / 1.7
```

```
= 176
```

```
vm.overcommit_ratio = (RAM - (0.026 * gp_vmem_rq)) / RAM  
= (256 - (0.026 * 176)) / 256  
= .982
```

你可以将示例系统中的`vm.overcommit_ratio`数值设置为98。

在启用了基于资源队列的资源管理系统下的Greenplum数据库中，`gp_vmem_protect_limit`的计算方式如下：

```
gp_vmem_protect_limit = gp_vmem_rq /  
maximum_acting_primary_segments  
= 176 / 11  
= 16GB  
= 16384MB
```

你可以将示例系统中的`gp_vmem_protect_limit`服务器配置参数数值设置为16384.

Parent topic: [性能管理](#)

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

□ 性能管理

管理性能

性能问题的常见原因

Greenplum数据库内存总览

□ 管理资源

检修性能问题

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

Greenplum数据为用户提供了根据业务需求对查询进行排序以及提供资源的功能，该功能的主要目的是为了防止在资源不可用时仍然发起查询的行为。

用户可以使用资源管理功能来限制并行的查询数、单个查询所使用的内存量和CPU使用量。Greenplum数据库提供了两种管理资源的模式——资源队列和资源组。

Important: 在启用着基于资源组的负载管理的RedHat 6.x 以及 CentOS 6.x系统中存在明显的Greenplum数据库性能下降现象，这是Linux cgroup的内核Bug造成的。该内核Bug已经在 RedHat 7.x 以及 CentOS 7.x系统中被修复了。

若用户对在RedHat 6系统下使用资源组的性能满意，且符合使用场景的话，请将内核版本升级到 2.6.32-696或更高以获得cgroups Bug修复带来的其他优势。

Greenplum数据库中同时只能启用资源组或资源队列中的一个资源管理模式。

在安装Greenplum数据库集群时，默认启用的资源管理模式是资源队列管理模式。尽管在该模式下，用户仍然可以创建并分配资源组，但启用资源组管理模式仍需要先启用资源组。

下表总结了一些资源队列和资源组的区别。

参数	资源队列	资源组
并行	在查询级别管理	在事务级别管理
CPU	指定队列顺序	指定CPU的使用百分比；使用Linux控制组
内存	在队列和操作级别管理；用户可以过量使用	在事务级别管理，可以进一步分配和追踪；用户不可以过量使用。
内存隔离	无	同资源组下的事务使用的内存是隔离的，不同资源组使用的内存也是隔离的。
用户	仅非管理员用户有 限制。	非管理员用户和超级用户都有限制。
排序	当没有可用槽位时，才开始排序	当槽位或内存不足时，开始排序

查询失效	当内存不足时，查询可能会立即失效	在没有更多的共享资源组内存的情况下，若事务到达了内存使用量限制后仍然提出增加内存的申请，查询可能会失效
避开限制	超级用户角色以及特定的操作者和功能不受限制。	SET、RESET和SHOW指令不受限制
外部组件	无	管理PL/Container CPU和内存资源

- [用资源组进行工作负载管理](#)

- [使用资源队列](#)

Parent topic: [性能管理](#)

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

 PXF介绍 PXF管理手册 配置PXF

PXF安装和配置目录

为PXF安装JAVA环境

初始化PXF

配置PXF服务器

 配置PXF HADOOP连接器 (可选)

配置用户模拟和代理

为安全HDFS配置PXF

配置Minio和S3对象存储的连接器 (可选)

配置Azure和Google云端存储的连接器 (可选)

配置JDBC连接器 (可选)

配置PXF客户端主机和端口(可选)

升级PXF

PXF的启动、停止和重启

In this topic:

- [准备](#)
- [过程](#)

为您的HDFS文件系统启用Kerberos时，作为HDFS客户端的PXF需要principal文件和keytab文件来验证对HDFS的访问。要在安全的HDFS上读取或写入文件，必须创建和部署PXF的Kerberos主体和keytab文件，并确保Kerberos身份验证已启用并起作用。

准备

在你配置PXF去访问需要安全访问的HDFS文件系统之前，确保你已经做到：

- 使用默认的PXF服务器配置配置了Hadoop连接器。
- 按照[Configuring PXF](#) 章节秒速初始化、配置和启动PXF，包括PXF功能和Hadoop用户模拟功能启用
- 根据特定分发的说明为Hadoop集群启用Kerberos，并验证配置。
- 确保HDFS的配置参数 `dfs.block.access.token.enable` 已经被设置成了 `true`。你可以在你hadoop集群的 `hdfs-site.xml` 配置文件中找到该配置。
- 注意每个Greenplum数据库segment主机(<seghost>)和Kerberos密钥分发中心(KDC) <kdc-server>主机的主机名或IP地址
- 注意群集所在的Kerberos <realm> 的名称。

过程

按照如下流程为有安全的HDFS配置PXF。您将在Kerberos KDC服务器和Greenplum数据库segment主机上执行操作。

登录到Kerberos KDC服务主机执行如下操作：

1. 使用 `root` 账户登录到 Kerberos KDC 服务主机。

```
$ ssh root@<kdc-server>
```

```
root@kdc-server$
```

- 如果你的集群主机不存在配置文件的话，需要在KDC服务主机上分发 `/etc/krb5.conf` 配置文件到GPDB的每一个的segment节点，例如：

```
root@kdc-server$ scp /etc/krb5.conf seghost:/etc/krb5.conf
```

- 使用 `kadmin.local` 命令为GPDB的每个segment节点主机创建一个Kerberos PXF服务主体。服务主体的格式应为 `gpadmin/<segghost>@<realm>`，其中`<segghost>`是segment主机系统的DNS可解析的全限定主机名(`hostname -f` 命令的输出)。例如，以下命令在名为EXAMPLE.COM的Kerberos领域中为名为 `host1.example.com`, `host2.example.com` 和 `host3.example.com` 的主机创建PXF服务主体：

```
root@kdc-server$ kadmin.local -q "addprinc -randkey -pw
changeme gpadmin/host1.example.com@EXAMPLE.COM"
root@kdc-server$ kadmin.local -q "addprinc -randkey -pw
changeme gpadmin/host2.example.com@EXAMPLE.COM"
root@kdc-server$ kadmin.local -q "addprinc -randkey -pw
changeme gpadmin/host3.example.com@EXAMPLE.COM"
```

- 为每个Kerberos PXF服务主体生成一个密钥表文件。将密钥表文件保存在任何方便的位置(本示例使用目录 `/etc/security/keytabs`)。您将在以后的步骤中将keytab文件部署到它们各自的Greenplum数据库segment主机。例如：

```
root@kdc-server$ kadmin.local -q "xst -norandkey -k
/etc/security/keytabs/pxf-host1.service.keytab
gpadmin/host1.example.com@EXAMPLE.COM"
root@kdc-server$ kadmin.local -q "xst -norandkey -k
/etc/security/keytabs/pxf-host2.service.keytab
gpadmin/host2.example.com@EXAMPLE.COM"
root@kdc-server$ kadmin.local -q "xst -norandkey -k
/etc/security/keytabs/pxf-host3.service.keytab
gpadmin/host3.example.com@EXAMPLE.COM"
```

根据需要重复 `xst` 命令用以在每个PXF服务主体上生成一个秘钥表。

- 列出principals:

```
root@kdc-server$ kadmin.local -q "listprincs"
```

6. 将每个PXF服务主体的密钥表文件复制到其各自的segment主机。例如，以下命令在 `PXF_CONF=/usr/local/greenplum-pxf` 时，将在步骤4中生成的每个主体复制到segment主机上的PXF默认keytab目录中

```
root@kdc-server$ scp /etc/security/keytabs/pxf-
host1.service.keytab host1.example.com:/usr/local/greenplum-
pxf/keytabs/pxf.service.keytab
root@kdc-server$ scp /etc/security/keytabs/pxf-
host2.service.keytab host2.example.com:/usr/local/greenplum-
pxf/keytabs/pxf.service.keytab
root@kdc-server$ scp /etc/security/keytabs/pxf-
host3.service.keytab host3.example.com:/usr/local/greenplum-
pxf/keytabs/pxf.service.keytab
```

Note the file system location of the keytab file on each PXF host; you will need this information for a later configuration step.

7. 更改 `pxf.service.keytab` 文件的属主和权限。这些文件必须仅由 `gpadmin` 用户拥有并可读。例如:

```
root@kdc-server$ ssh host1.example.com chown gpadmin:gpadmin
/usr/local/greenplum-pxf/keytabs/pxf.service.keytab
root@kdc-server$ ssh host1.example.com chmod 400
/usr/local/greenplum-pxf/keytabs/pxf.service.keytab
root@kdc-server$ ssh host2.example.com chown gpadmin:gpadmin
/usr/local/greenplum-pxf/keytabs/pxf.service.keytab
root@kdc-server$ ssh host2.example.com chmod 400
/usr/local/greenplum-pxf/keytabs/pxf.service.keytab
root@kdc-server$ ssh host3.example.com chown gpadmin:gpadmin
/usr/local/greenplum-pxf/keytabs/pxf.service.keytab
root@kdc-server$ ssh host3.example.com chmod 400
/usr/local/greenplum-pxf/keytabs/pxf.service.keytab
```

在每个GPDB的segment节点执行以下步骤:

1. 登录segment节点，例如:

```
$ ssh gpadmin@<segghost>
```

2. 如果在Greenplum的各个segment节点上Kerberos客户端包还没有被安装，那么首先安装他们。您必须具有超级用户权限才能安装操作系统软件包。例如:

```
root@segghost$ rpm -qa | grep krb
root@segghost$ yum install krb5-libs krb5-workstation
```

3. 打开 `pxf-env.sh` 用户配置文件。例如使用vi命令打开

```
PXF_CONF=/usr/local/greenplum-pxf:
```

```
gpadmin@segghost$ vi /usr/local/greenplum-pxf/conf/pxf-env.sh
```

4. 更新 `PXF_KEYTAB` 和 `PXF_PRINCIPAL` 配置. 指定keytab 文件和 Kerberos principal的位置。这些设置的默认值如下所示:

```
export PXF_KEYTAB="${PXF_CONF}/keytabs/pxf.service.keytab"
export PXF_PRINCIPAL="gpadmin/_HOST@EXAMPLE.COM"
```

PXF automatically replaces `_HOST` with the FQDN of the segment host.

5. 在segment上重启PXF服务:

```
gpadmin@segghost$ $GPHOME/pxf/bin/pxf restart
```

Greenplum数据库® 6.0文档

[Greenplum数据库最佳实践](#)

最佳实践概要

系统配置

模式设计

采用资源组管理内存和资源

采用资源队列管理内存和资源

[系统监控和维护](#)

用ANALYZE更新统计信息

管理数据库膨胀

监控Greenplum数据库日志文件

装载数据

安全性

加密数据和数据库连接

SQL查询调优

了解系统日志文件的位置和内容并且定期监控它们而不是在问题发生时才监控。

下面的表格展示了各种Greenplum数据库日志文件的位置。在文件路径中：

- `$GPADMIN_HOME` 是操作系统用户`gpadmin`的家目录 路径。
- `$MASTER_DATA_DIRECTORY`是Greenplum数据库master主机的数据目录。
- `$GPDATA_DIR`是Greenplum数据库segment主机的数据目录。
- `host`表示segment主机的主机名。
- `segprefix`是segment前缀。
- `N`是segment实例数量。
- `date`是YYYYMMDD格式的日期。

路径	描述
<code>\$GPADMIN_HOME/gpAdminLogs/*</code>	很多不同种类的日志文件，每台服务器都有目的目录。 <code>\$GPADMIN_HOME</code> 是 <code>gpAdminLogs/</code> 目录的默认存放位置。用户也可以在运行管理工具命令时指定一个另外的位置。
<code>\$GPADMIN_HOME/gpAdminLogs/gpinitsystem_date.log</code>	系统初始化日志
<code>\$GPADMIN_HOME/gpAdminLogs/gpstart_date.log</code>	启动日志
<code>\$GPADMIN_HOME/gpAdminLogs/gpstop_date.log</code>	停止日志
<code>\$GPADMIN_HOME/gpAdminLogs/gpsegstart.py_host:gpadmin_date.log</code>	segment主机启动
<code>\$GPADMIN_HOME/gpAdminLogs/gpsegstop.py_host:gpadmin_date.log</code>	segment主机停止日志
<code>\$MASTER_DATA_DIRECTORY/pg_log/startup.log,</code> <code>\$GPDATA_DIR/segprefixN/pg_log/startup.log</code>	segment实例启动日志
<code>\$MASTER_DATA_DIRECTORY/gpperfmon/logs/gpmon.*.log</code>	gpperfmon日志
<code>\$MASTER_DATA_DIRECTORY/pg_log/*.csv,</code> <code>\$GPDATA_DIR/segprefixN/pg_log/*.csv</code>	master和segment数据库日志
<code>\$GPDATA_DIR/mirror/segprefixN/pg_log/*.csv</code>	镜像segment数据库日志
<code>\$GPDATA_DIR/primary/segprefixN/pg_log/*.csv</code>	主segment数据库日志

首先使用`gplogfilter -t (--trouble)` 在master日志中搜索以 `ERROR:`、`FATAL:`或者`PANIC:`开始的消息。以`WARNING`开始的消息也可能提供有用的信息。

要在segment主机上搜索日志文件，可以用`gpssh`从master主机连接到segment主机使用 Greenplum的`gplogfilter`工具。用户可以通过`statement_id`在 segment日志中定位对应的日志项。

Greenplum数据库可以基于文件大小或当前日志文件年龄配置数据库日志的循环复写周期。`log_rotation_size` 配置参数设置单独一个日志文件的循环触发大小。当前日志文件大小大于等于该值时，该文件被关闭并新建另一个 日志文件。`log_rotation_age`配置参数指定当前日志文件的循环触发年龄。当从该文件 创建到目前的时间达到该设置时，创建一个新的日志文件。默认的`log_rotation_age`为`1d` (1天) ， 当前日志文件写完后，会创建一个新的以天为单位的文件。

Parent topic: [系统监控和维护](#)

Greenplum数据库® 6.0文档

□ Greenplum数据库最佳实践

最佳实践概要

系统配置

模式设计

采用资源组管理内存和资源

采用资源队列管理内存和资源

□ 系统监控和维护

用ANALYZE更新统计信息

管理数据库膨胀

监控Greenplum数据库日志文件

装载数据

安全性

加密数据和数据库连接

SQL查询调优

高可用性

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

良好查询性能的最重要的先决条件是从表的正确统计信息开始。用ANALYZE语句更新统计信息 让查询规划器能生成最优的查询计划。当表被分析时，有关数据的信息被存储在系统目录表中。如果存储的信息过时， 规划器可能会生成低效的执行计划。

有选择地生成统计信息

不带参数运行ANALYZE 会为数据库中所有的表更新统计信息。这样操作运行时间可能会非常长，因此不推荐这样做。当数据被改变时，使用者应该有选择地ANALYZE表或者使用`analyzedb`工具。

在大型表上运行ANALYZE可能需要很长时间。如果在非常大的表的所有列上运行ANALYZE 行不通，使用者可以只使用ANALYZE `table(column, ...)`为选择的列生成统计信息。确保包括用在连接、WHERE子句、SORT子句、GROUP BY子句或者 HAVING子句中的列都被收集了统计信息。

对于一个分区表，使用者可以只在更改过的分区（例如，使用者增加一个分区）上运行ANALYZE。注意对于分区表，使用者可以在父（主）表上或者叶子节点（实际存储数据和统计信息的分区文件）上运行ANALYZE。子分区表的中间文件没有存储数据或统计信息，因此在其上运行ANALYZE没有效果。使用者可以在`pg_partitions`系统目录中寻找分区表的名字：

```
SELECT partitiontablename from pg_partitions WHERE
tablename='parent_table';
```

提升统计信息质量

在生成统计信息所花的时间和统计信息的质量或者准确性之间存在着权衡。

为了允许大型表能在合理的时间内被分析完，ANALYZE会对表内容做随机采样而不是检查每一行。要对所有表列增加采样，可调整`default_statistics_target`配置参数。其目标值范围从1到1000，默认的目标值是100。`default_statistics_target`变量默认会被应用到所有的列。更大的目标值会增加执行ANALYZE所需的时间，但是可以提升查询规划器的评估质量。对于带有不规则数

据模式的列尤其如此。`default_statistics_target`可以在master或者会话级别设置，并且要求重新载入配置。

何时运行ANALYZE

在下列时机运行ANALYZE：

- 装载数据后；
- `CREATE INDEX`操作后；
- 在显著更改底层数据的`INSERT`、`UPDATE`以及`DELETE`操作之后。

`ANALYZE`仅在表上要求一个读锁，因此它可以与其他数据库活动并行运行。但不要在执行 装载、`INSERT`、`UPDATE`、`DELETE`以及`CREATE INDEX`操作期间运行`ANALYZE`。

配置统计信息自动收集

`gp_autostats_mode`配置参数

与`gp_autostats_on_change_threshold`参数一起决定何时触发自动分析操作。当自动统计信息收集被触发时，规划器会为查询增加一个`ANALYZE`步骤。

`gp_autostats_mode`默认为`on_no_stats`，这会为任何没有统计信息的表上的`CREATE TABLE AS SELECT`、`INSERT`或者`COPY`操作触发统计信息收集。

把`gp_autostats_mode`设置为`on_change`时，只有当受影响的行数超过由`gp_autostats_on_change_threshold`定义的阈值时才会触发统计信息收集，该阈值参数的默认值为2147483647。`on_change`设置下能触发自动统计信息收集的操作有：`CREATE TABLE AS SELECT`、`UPDATE`、`DELETE`、`INSERT`以及`COPY`。`CREATE TABLE AS SELECT`、`UPDATE`、`DELETE`、`INSERT`以及`COPY`。

将`gp_autostats_mode`设置为`none`会禁用自动统计信息收集。

对于分区表，如果数据从分区表的顶层父表插入，则自动统计信息收集不会被触发。但是如果数据直接被插入在分区表的叶子表（存储数据的地方）中，则自动统计信息收集会被触发。

Parent topic: [系统监控和维护](#)

Greenplum数据库® 6.0文档

□ Greenplum数据库最佳实践

最佳实践概要

系统配置

模式设计

采用资源组管理内存和资源

采用资源队列管理内存和资源

□ 系统监控和维护

用ANALYZE更新统计信息

管理数据库膨胀

监控Greenplum数据库日志文件

装载数据

安全性

加密数据和数据库连接

SQL查询调优

高可用性

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

Greenplum数据库的堆表使用PostgreSQL的多版本并发控制（MVCC）存储实现。被删除或更新的行被从数据库逻辑删除，但是该行的一个不可见映像保留在表中。这些被删除的行（也被称为过期行）被存储在一个空闲空间映射文件中。运行VACUUM会把过期行标记为可以被后续插入重用的空闲空间。

如果空闲空间映射不足以容纳所有的过期行，VACUUM命令就不能从导致空闲空间映射溢出的过期行回收空间。磁盘空间只能通过运行VACUUM FULL恢复，这个操作会锁住表，逐行拷贝到文件的开头，然后截断文件。这是一种昂贵的操作，对于大型的表，它可能需要超乎想象的时间来完成。应该只在较小的表上使用这种操作。如果使用者尝试杀死VACUUM FULL操作，系统可能会损坏。

Important:

在大量的的UPDATE以及DELETE操作之后非常有必要运行VACUUM，这样可以避免运行VACUUM FULL。

如果空闲空间映射溢出并且需要恢复空间，推荐使用CREATE TABLE...AS SELECT命令把该表拷贝为一个新表，这将会创建一个新的紧凑的表。然后删除原始表并且重命名拷贝的表为原始表名。

对于频繁更新的表来说，有少量或者中等数量的过期行以及空闲空间很正常，空闲空间将随着新数据的加入而被重用。但是当表被允许增长得非常大以至于活动数据只占空间的一小部分时，该表就明显地“膨胀”了。膨胀的表要求更多磁盘存储以及可能拖慢查询执行的额外I/O。

膨胀影响堆表、系统目录和索引。

在表上定期运行VACUUM语句可以防止它们长得过大。如果表确实出现了明显的膨胀，必须使用VACUUM FULL语句（或者可替代的过程）来紧缩文件。如果一个大型表变得明显膨胀，更好的方法是使用[从数据库表移除膨胀](#)中描述的方法之一来移除膨胀。

CAUTION:

绝不运行VACUUM FULL <database_name>并且不要在Greenplum数据库中的大型表上运行VACUUM FULL。



检测膨胀

ANALYZE语句所收集的统计信息可以被用来计算存储一个表所要求的磁盘页面的预计数量。页面的预计数量和实际数量之间的差别就是膨胀的度量。gp_toolkit模式提供了一个gp_bloat_diag视图，它通过预计页数和实际页数的比率来确定表膨胀。要使用这个视图，确定为数据库中所有的表都收集了最新的统计信息。然后运行下面的SQL：

```
gpadmin=# SELECT * FROM gp_toolkit.gp_bloat_diag;
 bdirelid | bdinspname | bdirelname | bdirelpages | bdiexppages | bdidiag
-----+-----+-----+-----+
-----+
 21488 | public      | t1          |           | 97          |
 1 | significant amount of bloat suspected
(1 row)
```

其结果只包括发生了中度或者明显膨胀的表。当实际页面数和预期页面的比率超过4但小于10时，就会报告为中度膨胀。当该比率超过10时就会报告明显膨胀。

gp_toolkit.gp_bloat_expected_pages视图会为每个数据库对象列出其已用页面的实际数量和预期数量。

```
gpadmin=# SELECT * FROM gp_toolkit.gp_bloat_expected_pages
LIMIT 5;
 btdrelid | btdrelopages | btdexppages
-----+-----+-----+
-----+
 10789 |           1 |           1
 10794 |           1 |           1
 10799 |           1 |           1
 5004  |           1 |           1
 7175  |           1 |           1
(5 rows)
```

btdrelid是该表的对象ID。btdrelopages列报告该表使用的页面数，btdexppages列是预期的页面数。另外，报出的数字是基于表统计信息的，因此要确保在已经更改的表上运行ANALYZE。

从数据库表移除膨胀

VACUUM命令会把过期行加入到共享的空间映射中，这样这些空间能被重用。当在被频繁更新的表上定期运行VACUUM时，过期行所占用的空间可以被迅速地重用，从而防止表文件长得更大。在空间映射被填满之前运行VACUUM也很重要。对于更新密集的表，用户可能需要每天运行VACUUM至少一次来防止表膨胀。

Warning: 当表出现显著膨胀时，在运行ANALYZE之前先运行VACUUM会更好。如果采样包含空的数据页，分析膨胀表会生成不合适的统计信息，所以在分析表之前先做VACUUM是最好的选择。

当表积累了显著的膨胀时，运行VACUUM命令并不能起到明显作用。对于小型表，运行 `VACUUM FULL <table_name>`能够回收导致空闲空间映射溢出的行所使用的空间并且减小表文件的尺寸。不过，`VACUUM FULL`语句是一种昂贵的操作，它要求一个`ACCESS EXCLUSIVE`锁并且可能需要异常长的时间完成。比起在一个大型表上运行`VACUUM FULL`，采用另一种方法从大型文件中移除膨胀会更好。注意每一种从大型表中移除膨胀的方法都是资源密集型的，并且只应该在极端情况下完成。

第一种从大型表中移除膨胀的方法是创建一个将过期行排除在外的表拷贝，删掉原始的表并且把这个拷贝重命名为原来的表名。这种方法使用`CREATE TABLE <table_name> AS SELECT`语句创建新表，例如：

```
gpadmin=# CREATE TABLE mytable_tmp AS SELECT * FROM
mytable;
gpadmin=# DROP TABLE mytable;
gpadmin=# ALTER TABLE mytable_tmp RENAME TO mytable;
```

第二种从表移除膨胀的方法是重新分布该表，这会把该表重建为不含过期行的表。参考步骤如下：

1. 把表的分布列记下来。
2. 把该表的分布策略改为随机分布：

```
ALTER TABLE mytable SET WITH (REORGANIZE=false)
DISTRIBUTED randomly;
```

这会为该表更改分布策略，但不会移除任何数据。该命令应该会立即完成。

3. 将分布策略改回其初始设置：

```
ALTER TABLE mytable SET WITH (REORGANIZE=true)
DISTRIBUTED BY (<original distribution columns>);
```

这一步会重新分布数据。因为表之前是用同样的分布键分布的，表中的行只需要简单地在同一Segment上重写即可，同时排除过期行。

从索引移除膨胀

VACUUM命令只会从表中恢复空间。要从索引中恢复空间，需要使用REINDEX命令重建它们。 The VACUUM command only recovers space from tables. To recover the space from indexes, recreate them using the REINDEX command.

要在表上重建所有的索引，可运行`REINDEX table_name;`。要重建一个特定的索引，可运行`REINDEX index_name;`。 REINDEX会将该索引相关`reltuples`和`relopages`的值设置为0（零），如果要更新统计信息，则有必要在重建索引后运行ANALYZE来更新它们。

从系统目录移除膨胀

Greenplum数据库系统目录也是堆表并且也可能随着时间推移变得膨胀。随着数据库对象被创建、修改或者删除，过期行会留在系统目录中。使用gpload装载数据会加剧膨胀，因为gpload会创建并且删除外部表。（为了避免使用gpload，推荐使用gpfldist装载数据。）

系统目录中的膨胀会导致扫描表所需的时间增加，例如在创建执行计划时需要扫描系统目录。系统目录会被频繁扫描，那么如果它们变得膨胀，整体的系统性能都会退化。

推荐每晚在系统目录上运行VACUUM，或者至少每周运行一次。同时，运行`REINDEX SYSTEM`从索引中移除膨胀。此外，还可以使用带`-s`（`--system`）选项的`reindexdb`工具对系统目录重建索引。在移除系统目录膨胀后，还有必要运行ANALYZE以更新系统目录表的统计信息。

以下是Greenplum数据库系统目录维护步骤。

1. 在系统目录表上执行REINDEX操作用于重建系统目录索引。该操作可以移除索引膨胀并提高 VACUUM性能。

Note: 当在系统目录表上执行REINDEX操作时，会锁住相应的表，进而影响到当前正在执行的查询性能。用户可以在系统的非活动窗口时间来调用REINDEX命令重建索引，以避免打扰正常业务操作的进行。

2. 在系统目录表上执行VACUUM命令。
3. 在系统目录表上执行ANALYZE操作来更新表的统计信息。

如果在维护窗口期内，由于时间限制需要停止目前正在运行的系统目

录维护，可以运行Greenplum数据库函数 pg_cancel_backend(<PID>) 来安全的停止该任务。

下面的脚本在系统目录上运行REINDEX、VACUUM和ANALYZE。

```
#!/bin/bash
DBNAME="<database_name>"
SYSTABLES=' pg_catalog.' || relname || ';' from pg_class
a, pg_namespace b \
where a.relnamespace=b.oid and b.nspname='pg_catalog' and
a.relkind='r'

reindexdb -s -d $DBNAME
psql -tc "SELECT 'VACUUM' || $SYSTABLES" $DBNAME | psql -a
$DBNAME
analyzedb -s pg_catalog -d $DBNAME
```

如果系统目录膨胀得很厉害，使用者就必须执行一次大强度的系统目录维护过程。采用CREATE TABLE AS SELECT 移除膨胀的方法以及重新分布数据的方法均不能被用于系统目录。使用者必须转而在计划的停机时段运行VACUUM FULL。在此期间，停止系统上所有的目录活动，VACUUM FULL会对系统目录加排他锁。定期运行 VACUUM能够预防最终不得不采用上面的高代价方法。

以下是较为彻底解决系统目录膨胀的步骤。

1. 停止Greenplum数据库上所有系统目录操作。
2. 在系统目录表上执行REINDEX操作来重建系统目录索引。该操作可以移除索引膨胀 并提高VACUUM性能。
3. 在系统目录表上执行VACUUM FULL操作。注意关注下面提到的注意事项。
4. 在系统目录表上执行ANALYZE操作来更新系统目录表的统计信息。

Note: 系统目录表pg_attribute通常是这里面最大的表。如果pg_attribute 表明显膨胀，在该表上的VACUUM FULL操作会占用很长时间，此时可能需要将操作分解。以下两种情形表明pg_attribute表存在明显膨胀并可能需要运行长时间的VACUUM FULL操作：

- pg_attribute表包含大量记录。
- gp_toolkit.gp_bloat_diag视图中有关pg_attribute表的诊断信息上显示该表存在明显膨胀。

从追加优化表移除膨胀

对追加优化表的处理与堆表有很大不同。尽管追加优化表允许更新、插入和删除，但它们并非为这些操作而优化，因此不推荐对追加优化表使用这些操作。如果使用者采纳这一建议并且为一次装载/多次读取负载使用追加优化，追加优化表上的VACUUM几乎会即刻运行。

如果使用者确实在追加优化表上运行了UPDATE或者DELETE命令，过期行会由一个辅助位图而不是空闲空间映射来跟踪。VACUUM是唯一能恢复空间的方式。在有过期行的追加优化表上运行VACUUM会通过把整个表重写成没有过期行的表以紧缩该表。不过，如果表中过期行的百分数超过了gp_appendonly_compaction_threshold配置参数的值，则不会执行任何操作，该参数的默认值是10（10%）。每个segment上都会检查该阈值，因此VACUUM语句可能会在某些segment上对追加优化表进行紧缩而在其他segment上不做紧缩。通过将gp_appendonly_compaction参数设置为no可以禁用对追加表的紧缩。

Parent topic: [系统监控和维护](#)

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

□ 性能管理

管理性能

性能问题的常见原因

Greenplum数据库内存

总览

□ 管理资源

用资源组进行工作负载管理

使用资源队列

检修性能问题

Greenplum database 5管理员
指南

用户可以使用资源组在Greenplum数据库中设置和实施CPU, 内存和并发事务限制。定义资源组后, 可以将该组分配给一个或多个Greenplum数据库角色, 或分配给外部组件 (如PL /Container), 以便控制这些角色或组件使用的资源。

当用户将资源组分配到角色 (基于角色的资源组), 用户定义该组的资源限制应用于所有的该组的所有角色。例如, 对于一个资源组的内存限制标识了该组所有Greenplum数据库用户提交的运行事务的最大内存使用量。

同样, 将资源组分配给外部组件时, 组限制将应用于组件的所有正在运行的实例。例如, 如果为PL/Container外部组件创建资源组, 则为组定义的内存限制将指定为其分配组的每个PL/Container运行时的所有正在运行的实例的最大内存使用量。

该章节包含以下小节

- [理解角色和组件资源组](#)

- [资源组的参数和限制](#)

- [内存审计器](#)

- [事务并行限制](#)

- [CPU限制](#)

- [内存限制](#)

- [使用资源组](#)

- [启用资源组](#)

- [创建资源组](#)

- [将资源组分配给角色](#)

- [监视资源组状态](#)

- [资源组常见问题解答](#)

Parent topic: [管理资源](#)

理解角色和组件资源组

Greenplum数据库支持两类型的资源组：管理角色资源的 [□](#) 以及管理外部组件 (如PL/Container) 资源的组。

资源组的最常见应用程序是管理不同角色可以在Greenplum数据库群集中并发执行的活动查询的数量。用户还可以管理Greenplum为每个查询分

配的CPU和内存资源量。

角色的资源组使用Linux控制组 (cgroup) 进行CPU资源管理。Greenplum数据库使用称为vmtracker的内存审计器在内部为这些资源组跟踪虚拟内存。

当用户执行查询时，Greenplum数据库会根据为资源组定义的一组限制来评估查询。如果尚未达到组的资源限制并且查询不会导致组超过并发事务限制，Greenplum数据库会立即执行查询。如果不满足这些条件，Greenplum数据库会对查询进行排队。例如，如果已达到资源组的最大并发事务数，则后续查询将排队，并且必须等待其他查询在运行之前完成。当资源组的并发和内存限制被更改为足够大的值时，Greenplum数据库也可以执行挂起的查询。

在角色的资源组中，事务以先进先出的方式进行评估。Greenplum数据库定期评估系统的活动工作负载，根据需要重新分配资源和启动/排队作业。

用户还可以使用资源组来管理外部组件（如PL/Container）的CPU和内存资源。外部组件的资源组使用Linux cgroup来管理组件的总CPU和总内存资源。

Note: Greenplum数据库的容器化部署（例如Greenplum for Kubernetes）可能会创建一组嵌套的cgroup，以管理主机系统资源。嵌套cgroup会影响Greenplum数据库资源组对CPU百分比，CPU核心和内存的限制（Greenplum数据库外部组件除外）。Greenplum数据库资源组系统资源限制基于父组的配额。

例如，Greenplum数据库在cgroup演示中运行，Greenplum Database cgroup嵌套在cgroup演示中。如果cgroup演示配置CPU限制为系统CPU资源的60%且Greenplum数据库资源组CPU限制设置为90%，则主机系统CPU资源的Greenplum数据库限制为54% (0.6×0.9)。

嵌套的cgroup不会影响Greenplum数据库外部组件（如PL/Container）的内存限制。仅当用于管理Greenplum数据库资源的cgroup未嵌套时，才能管理外部组件的内存限制，cgroup配置为顶级cgroup。

有关配置cgroup以供资源组使用的信息，请参阅[使用资源组](#)。

资源组的参数和限制

当用户创建一个资源组时，请注意以下几点：

- 通过识别该组存储器是如何审核指定资源组的类型。
- 提供一组限制，用于确定组可用的CPU和内存资源量。

资源组的参数和限制：

限制类型	描述
MEMORY_AUDITOR	用于资源组的内存审计器。如果要将资源组分配给角色，则需要vmtracker缺省值）。指定cgroup以将资源组分配给外部组件。
CONCURRENCY	资源组中允许的最大并发事务数，包括活动和空闲事务。
CPU_RATE_LIMIT	此资源组可用的CPU资源百分比。
CPUSSET	为该资源组保留的CPU核心数
MEMORY_LIMIT	该资源组可用的内存资源百分比。
MEMORY_SHARED_QUOTA	提交到该资源组的事务之间共享的内存资源百分比
MEMORY_SPILL_RATIO	内存密集型事务的内存使用阈值。当事务达到此阈值时，它将溢出到磁盘。

Note: 不对SET, RESET和SHOW命令强制执行资源限制。

内存审计器

MEMORY_AUDITOR属性通过标识组的内存审计器来指定资源组的类型。指定vmtrackerMEMORY_AUDITOR的资源组标识角色的资源组。指定cgroupMEMORY_AUDITOR的资源组标识外部组件的资源组。

默认的MEMORY_AUDITOR是vmtracker。

用户为资源组指定的MEMORY_AUDITOR确定Greenplum数据库是否以及如何使用限制属性来管理CPU和内存资源：

限制类型	角色资源组	外部组件资源组
CONCURRENCY	是	否；必须为零(0)。
CPU_RATE_LIMIT	是	是
CPUSSET	是	是
MEMORY_LIMIT	是	Yes

MEMORY_SHARED_QUOTA	是	视组件决定
MEMORY_SPILL_RATIO	是	视组件决定

事务并行限制

CONCURRENCY限制控制角色资源组允许的最大并发事务数。

Note: CONCURRENCY限制不适用于外部组件的资源组，对于此类组必须设置为零 (0) 。

角色的每个资源组在逻辑上被划分为等于CONCURRENCY限制的固定数量的槽。 Greenplum数据库为这些插槽分配相等，固定百分比的内存资源。

角色资源组的默认CONCURRENCY限制值为20。

Greenplum数据库将资源组达到CONCURRENCY限制后提交的任何事务排队。当正在运行的事务完成时，如果存在足够的内存资源，Greenplum Database将排队并执行最早的排队事务。

用户可以设置服务器配置参数[gp_resource_group_bypass](#)来绕过资源组的并行限制。

CPU限制

通过将特定CPU核心分配给组，或通过标识要分配给组的分段CPU资源的百分比，可以配置要为段主机上的资源组保留的CPU资源份额。

Greenplum数据库使用CPUSERT和CPU_RATE_LIMIT资源组限制来标识CPU资源分配模式。配置资源组时，必须仅指定其中一个限制。

用户可以在Greenplum数据库群集中同时使用两种CPU资源分配模式。用户还可以在运行时更改资源组的CPU资源分配模式。

[gp_resource_group_cpu_limit](#)服务器配置参数标识要分配给每个Greenplum数据库段主机上的资源组的系统CPU资源的最大百分比。无论为组配置的CPU分配模式如何，此限制都将控制段主机上所有资源组的最大CPU使用率。剩余的未预留CPU资源用于OS内核和Greenplum数据库辅助守护进程。默认的gp_resource_group_cpu_limit值为.9 (90%)。

Note: 如果用户在Greenplum数据库群集节点上运行其他工作负载，则默认的gp_resource_group_cpu_limit值可能不会留下足够的CPU资源，因此请务必相应地调整此服务器配置参数。

Warning: 避免将gp_resource_group_cpu_limit设置为高于.9的值。这样做可能会导致高工作负载查询接近所有CPU资源，可能使Greenplum数据库辅助进程匮乏。

按核心数分配CPU资源

用户可以使用CPUSED属性标识要为资源组保留的CPU核心。用户指定的CPU核心必须在系统中可用，并且不能与为其他资源组保留的任何CPU核心重叠。（尽管Greenplum数据库使用用户专门为该组分配给资源组的核心，但请注意，系统中的非Greenplum进程也可以使用这些CPU核心。）

配置CPUSED时，请指定以逗号分隔的单核数字或数字间隔列表。必须将核心数字/间隔用单引号括起来，例如“1,3-4”。

将CPU核心分配给CPUSED组时，请考虑以下事项：

- 使用CPUSED创建的资源组仅使用指定的核心。如果组中没有正在运行的查询，则保留的核心处于空闲状态，并且其他资源组中的查询无法使用这些核心。请考虑最小化CPUSED组的数量以避免浪费系统CPU资源。
- CPU核心0未分配：在以下情况下，CPU核心0用作回退机制：
 - admin_group和default_group至少需要一个CPU核心。保留所有CPU内核后，Greenplum Database会将CPU内核0分配给这些默认组。在这种情况下，用户为其分配CPU核心0的资源组与admin_group和default_group共享核心。
 - 如果通过一个节点替换重新启动Greenplum数据库集群，并且节点没有足够的内核来为所有CPUSED资源组提供服务，则会自动为这些组分配CPU核心0以避免系统启动失败。
- 将核心分配给资源组时，请使用尽可能少的核心编号。如果替换Greenplum数据库节点并且新节点的CPU核心数少于原始节点，或者备份数据库并希望在具有较少CPU核心的节点的群集上还原它，则操作可能会失败。例如，如果用户的Greenplum数据库群集有16个核心，则分配核心1-7是最佳选择。如果创建资源组并将CPU核心9分配给该组，则数据库还原到8核心节点将失败。

使用CPUSED配置的资源组在CPU资源上具有更高的优先级。在段主机上配置了CPUSED的所有资源组的最大CPU资源使用百分比是保留的CPU核心数除以所有CPU核心数乘以100。

为资源组配置CPUSED时，Greenplum数据库会禁用组的CPU_RATE_LIMIT并将值设置为-1。

Note: 在为Greenplum数据库群集启用基于资源组的资源管理后，必须为

资源组配置CPUSSET。

按百分比分配CPU资源

Greenplum数据库节点CPU百分比在Greenplum节点上的每个段之间平均分配。 使用CPU_RATE_LIMIT配置的每个资源组都保留用于资源管理的段CPU的指定百分比。

用户可以为资源组指定的最小CPU_RATE_LIMIT百分比为1，最大值为100。

在Greenplum数据库群集中定义的所有资源组指定的CPU_RATE_LIMIT的总和不得超过100。

使用CPU_RATE_LIMIT在段主机上配置的所有资源组的最大CPU资源使用量是以下值中的最小值：

- 非保留CPU核心数除以所有CPU核心数乘以100，和
- gp_resource_group_cpu_limit值。

配置有CPU_RATE_LIMIT的资源组的CPU资源分配是弹性的，因为Greenplum数据库可以将空闲资源组的CPU资源分配给更繁忙的资源组。在这种情况下，当该资源组接下来变为活动时，CPU资源被重新分配给先前空闲的资源组。如果多个资源组繁忙，则根据其CPU_RATE_LIMIT的比率为它们分配任何空闲资源组的CPU资源。例如，使用CPU_RATE_LIMIT为40创建的资源组将分配两倍于使用CPU_RATE_LIMIT为20创建的资源组的额外CPU资源。

为资源组配置CPU_RATE_LIMIT时，Greenplum数据库会禁用组的CPUSSET并将值设置为-1。

内存限制

启用资源组后，将在Greenplum数据库节点，段和资源组级别管理内存使用情况。用户还可以使用角色资源组在事务级别管理内存。

`gp_resource_group_memory_limit`服务器配置参数标识要分配给每个Greenplum数据库段主机上的资源组的系统内存资源的最大百分比。默认的gp_resource_group_memory_limit值为.7 (70%)。

Greenplum数据库节点上可用的内存资源在节点上的每个段之间进一步平均分配。当基于资源组的资源管理处于活动状态时，分配给段主机上每个段的内存量是Greenplum数据库可用的内存乘

以gp_resource_group_memory_limit服务器配置参数，并除以主机上活动主段的数量：

```
rg_perseg_mem = ((RAM * (vm.overcommit_ratio / 100) + SWAP) *  
gp_resource_group_memory_limit) / num_active_primary_segments
```

每个资源组保留用于资源管理的段内存的百分比。用户可以通过在创建资源组时指定的MEMORY_LIMIT值来标识此百分比。用户可以为资源组指定的最小MEMORY_LIMIT百分比为1，最大值为100。

在Greenplum数据库群集中定义的所有资源组指定的MEMORY_LIMIT总和不得超过100。

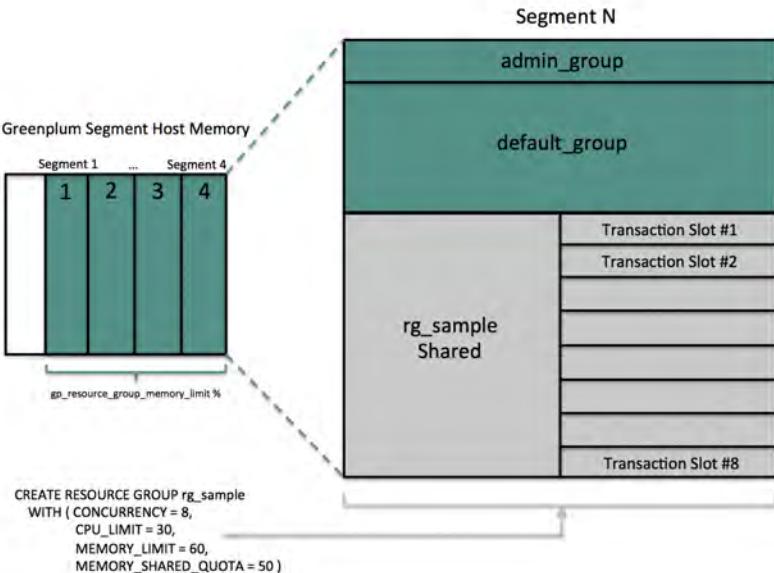
于角色的资源组的附加内存限制

资源组为角色保留的内存进一步分为固定和共享组件。创建资源组时指定的MEMORY_SHARED_QUOTA值标识可在当前运行的事务之间共享的预留资源组内存的百分比。此记忆按先到先得的原则分配。正在运行的事务可以使用MEMORY_SHARED_QUOTA中的任何一个，一些或全部。

用户可以指定的最小MEMORY_SHARED_QUOTA为0，最大值为100。默认MEMORY_SHARED_QUOTA为20。

如前所述，CONCURRENCY标识角色资源组中允许的最大并发运行事务数。由资源组保留的固定存储器被分成CONCURRENCY个事务槽数。每个插槽分配一个固定的，相等数量的资源组内存。Greenplum数据库会保证每个事务的固定内存。

Figure 1. 资源组内存分配



当查询的内存使用量超过固定的每事务内存使用量时，Greenplum数据库会将可用的资源组共享内存分配给查询。特定事务槽可用的最大资源组内存量是事务的固定内存和完整资源组共享内存分配的总和。

全局共享内存

为所有资源组（包括默认`admin_group`和`default_group`组）配置的`MEMORY_LIMIT`的总和标识预留资源组内存的百分比。如果此总和小于100，则Greenplum数据库会将任何未预留的内存分配给资源组全局共享内存池。

资源组全局共享内存仅适用于使用`vmtracker`内存审计器配置的资源组。

在可用时，Greenplum数据库在首次分配插槽和资源组共享内存后为事务分配全局共享内存。Greenplum数据库以先到先得的方式为事务分配资源组全局共享内存。

Note: Greenplum数据库跟踪但不主动监视资源组中的事务内存使用情况。如果资源组的内存使用量超过其固定内存分配，则在满足以下所有条件时，资源组中的事务将失败：

- 没有可用的资源组共享内存。
- 没有可用的全局共享内存。
- 该事务请求额外的内存。

当用户为全局共享内存池保留一些未分配的内存（例如，10-20%）时，Greenplum数据库会更有效地使用资源组内存。全局共享内存的可用性还有助于缓解内存消耗或不可预测的查询失败。

查询运算符内存

大多数查询运算符都是非内存密集型的；也就是说，在处理过程中，Greenplum数据库可以将其数据保存在已分配的内存中。当内存密集型查询运算符（如连接和排序）处理的数据多于可以保存在内存中的数据时，数据会溢出到磁盘。

`gp_resgroup_memory_policy`服务器配置参数控制所有查询运算符的内存分配和分配算法。Greenplum数据库支持`eager-free`（默认）和资源组的自动内存策略。当用户指定自动策略时，Greenplum数据库使用资源组内存限制在查询运算符之间分配内存，为非内存密集型运算符分配固定大小的内存，其余为内存密集型运算符。当`eager_free`策略到位时，Greenplum数据库通过在稍后的查询阶段将已完成处理的运算符释放的内存重新分配给运算符，从而更好地在运算符之间分配内存。

`MEMORY_SPILL_RATIO`标识事务中内存密集型运算符的内存使用阈值。达到此阈值时，事务将溢出到磁盘。Greenplum数据库使用`MEMORY_SPILL_RATIO`来确定要分配给事务的初始内存。

可以为资源组指定的最小`MEMORY_SPILL_RATIO`百分比为0.最大值为100.默认`MEMORY_SPILL_RATIO`为20。

在为角色创建资源组时定义`MEMORY_SPILL_RATIO`，用户可以使用`memory_spill_ratio`服务器配置参数在会话级别基于每个查询选择性地设置此限制。

memory_spill_ratio和低内存查询

低`memory_spill_ratio`设置（例如，在0-2%范围内）已被证明可以提高具有低内存要求的查询的性能。使用`memory_spill_ratio`服务器配置参数可以基于每个查询覆盖该设置。例如：

```
SET memory_spill_ratio=0;
```

其他内存注意事项

角色的资源组跟踪通过`palloc()`函数分配的所有Greenplum数据库内

存。使用Linux `malloc()` 函数分配的内存不受这些资源组的管理。要确保角色的资源组准确跟踪内存使用情况，请避免使用 `malloc()` 在自定义 Greenplum 数据库用户定义函数中分配大量内存。

使用资源组

Important: 在 RedHat 6.x 和 CentOS 6.x 系统上启用基于资源组的工作负载管理时，已观察到显着的 Greenplum 数据库性能下降。此问题是由于 Linux cgroup 内核错误引起的。这个内核错误已在 CentOS 7.x 和 Red Hat 7.x 系统中修复。

如果用户使用 RedHat 6 并且资源组的性能对于用户的用例是可接受的，请将用户的内核升级到版本 2.6.32-696 或更高版本，以便从 cgroups 实现的其他修复程序中受益。

条件

Greenplum 数据库资源组使用 Linux 控制组 (cgroup) 来管理 CPU 资源。Greenplum 数据库还使用 cgroup 来管理外部组件的资源组的内存。使用 cgroups，Greenplum 将 Greenplum 进程的 CPU 和 外部组件内存使用与节点上的其他进程隔离开来。这允许 Greenplum 在每个资源组的基础上支持 CPU 和 外部组件内存使用限制。

有关 cgroup 的详细信息，请参阅 Linux 发行版的控制组文档。

在 Greenplum 数据库群集中的每个节点上完成以下任务，以设置用于资源组的 cgroup：

1. 如果用户在 Greenplum 数据库群集节点上运行 SuSE 11+ 操作系统，则必须在每个节点上启用交换记帐并重新启动 Greenplum 数据库群集。`swapaccount` 内核引导参数控制 SuSE 11+ 系统上的交换记帐设置。设置此引导参数后，必须重新引导系统。有关详细信息，请参阅 SuSE 11 发行说明中的 [Cgroup 交换控制](#) 讨论。用户必须是超级用户或具有 `sudo` 访问权限才能配置内核引导参数和重新引导系统。
2. 创建 Greenplum 数据库 cgroups 配置文件 `/etc/cgconfig.d/gpdb.conf`。用户必须是超级用户或具有 `sudo` 访问权限才能创建此文件：

```
sudo vi /etc/cgconfig.d/gpdb.conf
```

3. 将以下配置信息添加到 `/etc/cgconfig.d/gpdb.conf`：

```

group gpdb {
    perm {
        task {
            uid = gpadmin;
            gid = gpadmin;
        }
        admin {
            uid = gpadmin;
            gid = gpadmin;
        }
    }
    cpu {
    }
    cpuacct {
    }
    cpuset {
    }
    memory {
    }
}

```

此内容配置由gpadmin用户管理的CPU， CPU计算， CPU核心集和内存控制组。Greenplum数据库仅将内存控制组用于使用cgroupMEMORY_AUDITOR创建的资源组。

- 如果尚未安装并正在运行，请安装Control Groups操作系统软件包，并在每个Greenplum Database节点上启动cgroups服务。用户运行以执行这些任务的命令将根据节点上安装的操作系统而有所不同。用户必须是超级用户或具有sudo访问权限才能运行这些命令：

- Redhat/CentOS 7.x 系统：

```

sudo yum install libcgrouptools
sudo cgconfigparser -l /etc/cgconfig.d/gpdb.conf

```

- Redhat/CentOS 6.x systems:

```

sudo yum install libcgrouptools
sudo service cgconfig start

```

- SuSE 11+ systems:

```

sudo zypper install libcgrouptools
sudo cgconfigparser -l /etc/cgconfig.d/gpdb.conf

```

- 标识节点的cgroup目录安装点：

```

grep cgroup /proc/mounts

```

第一行输出标识cgroup挂载点。

6. 通过运行以下命令验证是否正确设置了Greenplum Database cgroups配置。将<cgroup_mount_point> 替换为用户在上一步中标识的安装点：

```
ls -l <cgroup_mount_point>/cpu/gpdb  
ls -l <cgroup_mount_point>/cpuacct/gpdb  
ls -l <cgroup_mount_point>/cpuset/gpdb  
ls -l <cgroup_mount_point>/memory/gpdb
```

如果这些目录存在且由gpadmin:gpadmin拥有，则用户已成功为Greenplum数据库CPU资源管理配置了cgroup。

7. 要在系统重新启动时自动重新创建Greenplum数据库所需的cgroup层次结构和参数，请将系统配置为启用Linux cgroup服务守护程序cgconfig.service (Redhat / CentOS 7.x和SuSE 11+) 或cgconfig (Redhat / CentOS 6.x) 节点启动时。例如，在首选服务自动启动工具中配置以下cgroup服务命令之一：
- Redhat/CentOS 7.x 和 SuSE11+ 系统：

```
sudo systemctl enable cgconfig.service
```

- Redhat/CentOS 6.x systems:

```
sudo chkconfig cgconfig on
```

用户可以选择其他方法来重新创建Greenplum数据库资源组cgroup层次结构。

程序

要在Greenplum数据库群集中使用资源组，用户需要：

1. [为Greenplum数据库群集启用资源组。](#)
2. [创建资源组。](#)
3. [将资源组分配给一个或多个角色。](#)
4. [使用资源管理系统视图来监视和管理资源组。](#)

启用资源组

安装Greenplum Database时，默认情况下会启用资源队列。要使用资源组而不是资源队列，必须设置[gp_resource_manager](#)服务器配置参数。

- 将gp_resource_manager服务器配置参数设置为值"group":

```
gpconfig -s gp_resource_manager
gpconfig -c gp_resource_manager -v "group"
```

- 重启Greenplum 数据库:

```
gpstop
gpstart
```

启用后，角色提交的任何事务都将定向到分配给该角色的资源组，并受该资源组的并发、内存和CPU限制的约束。同样，外部组件的CPU和内存使用量由为分配给组件的资源组配置的CPU和内存限制控制。

Greenplum数据库为名为admin_group和default_group的角色创建两个默认资源组。启用资源组时，将为未明确分配资源组的任何角色分配角色功能的默认组。SUPERUSER角色分配了admin_group，非管理员角色分配了名为default_group的组。

使用以下资源限制创建默认资源组admin_group和default_group：

限制类型	admin_group	default_group
CONCURRENCY	10	20
CPU_RATE_LIMIT	10	30
CPUSET	-1	-1
MEMORY_LIMIT	10	30
MEMORY_SHARED_QUOTA	50	50
MEMORY_SPILL_RATIO	20	20
MEMORY_AUDITOR	vmtracker	vmtracker

请记住，默认资源

组admin_group和default_group的CPU_RATE_LIMIT和MEMORY_LIMIT值对分段主机上的总百分比有贡献。在创建Greenplum数据库部署并将新资源组添加到Greenplum数据库部署时，用户可能会发现需要为admin_group和/或default_group调整这些限制。

创建资源组

为角色创建资源组时可以提供名称、CPU资源分配模式和内存限制。用户可以选择提供并发事务限制和内存共享配额和溢出比率。使用CREATE RESOURCE GROUP命令创建新资源组。

为角色创建资源组时，必须提供CPU_RATE_LIMIT或CPUSED和MEMORY_LIMIT限制值。这些限制标识要分配给此资源组的Greenplum数据库资源的百分比。例如，要创建名为*rgroup1*的资源组，其CPU限制为20，内存限制为25：

```
=# CREATE RESOURCE GROUP rgroup1 WITH (CPU_RATE_LIMIT=20,
MEMORY_LIMIT=25);
```

CPU限制为20由*rgroup1*分配到的每个角色共享。同样，内存限制为25由*rgroup1*分配到的每个角色共享。*rgroup1*使用默认的MEMORY_AUDITORvmtracker和默认的CONCURRENCY设置为20。

为外部组件创建资源组时，必须提供CPU_RATE_LIMIT或CPUSED和MEMORY_LIMIT限制值。用户还必须提供MEMORY_AUDITOR并将CONCURRENCY显式设置为零（0）。例如，要创建名为*rgroup_extcomp*的资源组，请为其保留CPU核心1并指定内存限制为15：

```
=# CREATE RESOURCE GROUP rgroup_extcomp WITH
(MEMORY_AUDITOR=cgroup, CONCURRENCY=0,
CPUSED='1', MEMORY_LIMIT=15);
```

The [ALTER RESOURCE GROUP](#)命令更新资源组的限制。要更改资源组的限制，请指定组所需的新值。例如：

```
=# ALTER RESOURCE GROUP rg_role_light SET CONCURRENCY 7;
=# ALTER RESOURCE GROUP exec SET MEMORY_LIMIT 25;
=# ALTER RESOURCE GROUP rgroup1 SET CPUSED '2,4';
```

Note: 用户无法将admin_group的CONCURRENCY值设置或更改为零（0）。

[DROP RESOURCE GROUP](#)命令会删除资源组。要删除角色的资源组，不能将该组分配给任何角色，也不能在资源组中激活或等待任何事务。删除正在运行的实例的外部组件的资源组会导致正在运行的实例。

删除资源组：

```
=# DROP RESOURCE GROUP exec;
```

将资源组分配给角色

使用默认MEMORY_AUDITORvmtracker创建资源组时，该组可用于分配给一个或多个角色（用户）。使用[CREATE ROLE](#)或[ALTER ROLE](#)命令的RESOURCE GROUP子句将资源组分配给数据库角色。如果未为角色指定资源组，则会为角色分配角色功能的默认组。SUPERUSER角色分配了admin_group，非管理员角色分配了名为default_group的组。

使用ALTER ROLE或CREATE ROLE命令将资源组分配给角色。例如：

```
=# ALTER ROLE bill RESOURCE GROUP rg_light;
=# CREATE ROLE mary RESOURCE GROUP exec;
```

用户可以将资源组分配给一个或多个角色。如果已定义角色层次结构，则将资源组分配给父角色不会向下传播到该角色组的成员。

Note: 用户无法将为外部组件创建的资源组分配给角色。

如果要从角色中删除资源组分配并将角色分配为默认组，请将角色的组名称分配更改为NONE。例如：

```
=# ALTER ROLE mary RESOURCE GROUP NONE;
```

监视资源组状态

监视资源组和查询的状态可能涉及以下任务：

- [查看资源组限制](#)
- [查看资源组查询状态和CPU/内存使用情况](#)
- [查看分配给角色的资源组](#)
- [查看资源组的运行和待定查询](#)
- [取消资源组中的正在运行或已排队的事务](#)

查看资源组限制

The [gp_resgroup_config](#) gp_toolkit系统视图显示资源组的当前和建议限制。当用户更改限制时，建议的限制与当前限制不同，但不能立即应用新值。如果要查看所有资源组的限制：

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

查看资源组查询状态和CPU/内存使用情况

`gp_resgroup_status` gp_toolkit 系统视图可以查看资源组的状态和活动。该视图显示正在运行和排队的事务数。它还显示资源组的实时CPU和内存使用情况。要查看此信息：

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

查看每个主机的资源组CPU/内存使用情况

通过`gp_resgroup_status_per_host` gp_toolkit 系统视图，用户可以基于每个主机查看资源组的实时CPU和内存使用情况。要查看此信息：

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

查看每个段的资源组CPU/内存使用情况

通过`gp_resgroup_status_per_segment` gp_toolkit 系统视图，用户可以按每个段，每个主机查看资源组的实时CPU和内存使用情况。要查看此信息：

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

查看分配给角色的资源组

要查看资源组到角色分配，请对 `pg_roles` 和 `pg_resgroup` 系统目录表执行以下查询：

```
=# SELECT rolname, rsgname FROM pg_roles, pg_resgroup  
WHERE pg_roles.rolresgroup=pg_resgroup.oid;
```

查看资源组的运行和待定查询

要查看资源组的运行查询，挂起的查询以及挂起的查询排队的时间，请检查 `pg_stat_activity` 系统目录表：

```
=# SELECT current_query, waiting, rsgname, rsgqueueduration
   FROM pg_stat_activity;
```

`pg_stat_activity`显示有关启动查询的用户/角色的信息。使用外部组件（如PL/Container）的查询由两部分组成：在Greenplum数据库中运行的查询运算符和在PL/Container实例中运行的UDF。Greenplum数据库处理分配给发起查询的角色的资源组下的查询运算符。在PL/Container实例中运行的UDF在分配给PL/Container运行时的资源组下运行。后者没有在`pg_stat_activity`中表示；Greenplum数据库无法深入了解PL/Container等外部组件如何在运行实例中管理内存。

取消资源组中的正在运行或已排队的事务

在某些情况下，用户可能希望取消资源组中的正在运行或排队的事务。例如，用户可能希望删除在资源组队列中等待但尚未执行的查询。或者，用户可能希望停止执行时间太长的正在运行的查询，或者在事务中处于空闲状态并占用其他用户所需的资源组事务插槽的查询。

要取消正在运行或排队的事务，必须首先确定与事务关联的进程ID（pid）。获得进程ID后，可以调用`pg_cancel_backend()`来结束该进程，如下所示。

例如，要查看与当前活动或在所有资源组中等待的所有语句关联的进程信息，请运行以下查询。如果查询未返回任何结果，则任何资源组中都没有正在运行或排队的事务。

```
=# SELECT rolname, g.rsgname, procpid, waiting,
       current_query, datname
      FROM pg_roles, gp_toolkit.gp_resgroup_status g,
           pg_stat_activity
     WHERE pg_roles.rolresgroup=g.groupid
       AND pg_stat_activity.username=pg_roles.rolname;
```

示例部分查询输出：

rolname	rsgname	procpid	waiting	current_query
datname				
-----	-----	-----	-----	-----
-----	-----	-----	-----	-----
sammy	rg_light	31861	f	<IDLE> in transaction
	testdb			
billy	rg_light	31905	t	SELECT * FROM topten
	testdb			

使用此输出来标识要取消的事务的进程ID（`procpid`），然后取消该进

程。例如，要取消上面示例输出中标识的待处理查询：

```
=# SELECT pg_cancel_backend(31905);
```

用户可以在`pg_cancel_backend()`的第二个参数中提供可选消息，以向用户指示进程被取消的原因。

Note:

不要使用操作系统KILL命令取消任何Greenplum数据库进程。

资源组常见问题解答

CPU

- 为什么CPU使用率低于为资源组配置的**CPU_RATE_LIMIT**?
当资源组中运行的查询和片数较少时，您可能会遇到这种情况，并且这些进程未使用系统上的所有核心。
- 为什么资源组的CPU使用率高于配置的**CPU_RATE_LIMIT**?
在下列情况下可能会发生这种情况：
 - 当其他资源组空闲时，资源组可以使用比其**CPU_RATE_LIMIT**更多的CPU。在这种情况下，Greenplum数据库将空闲资源组的CPU资源分配给更繁忙的资源组。此资源组功能称为CPU突发。
 - 操作系统CPU调度程序可能导致CPU使用率出现峰值，然后下降。如果您认为可能发生这种情况，请计算给定时间段内的平均CPU使用率（例如，5秒），并使用该平均值来确定CPU使用率是否高于配置的限制。

内存

- 为什么我的查询返回“内存不足”错误?
在资源组中提交的事务失败并在内存使用超过其固定内存分配时退出，不存在可用资源组共享内存，并且事务请求更多内存。
- 为什么我的查询返回“达到内存限制”错误?
当您使用`ALTER RESOURCE GROUP`更改资源组的内存和/或并发限制时，Greenplum数据库会自动将事务和组内存调整为新设置。如果您最近更改了资源组属性并且当前正在运行的查询不再有足够的可用内存，则可能会出现“内存不足”错误。
- 为什么我的资源组的实际内存使用量超过为该组配置的数量?

当从组中运行的一个或多个查询从全局共享内存池分配内存时，资源组的实际内存使用量可能超过配置的量。（如果没有可用的全局共享内存，查询将失败，并且不会影响其他资源组的内存资源。）

当全局共享内存可用时，当事务溢出到磁盘时，内存使用量也可能超过配置的量。Greenplum数据库语句在开始溢出到磁盘时继续请求内存，因为：

- 溢出到磁盘需要额外的内存才能工作。
- 其他运营商可能会继续请求内存。

泄漏情况下内存使用量增加；当全局共享内存可用时，资源组最终可能最多使用其配置的组内存限制的200-300%。

并发

- 为什么运行事务的数量低于为资源组配置的**CONCURRENCY**限制？
Greenplum数据库在运行事务之前考虑内存可用性，如果没有足够的内存可供服务，它将对事务进行排队。如果使用ALTER RESOURCE GROUP增加资源组的CONCURRENCY限制但不调整内存限制，则当前正在运行的事务可能会占用该组的所有分配的内存资源。处于此状态时，Greenplum数据库会对资源组中的后续事务进行排队。
- 为什么资源组中正在运行的事务数高于配置的**CONCURRENCY**限制？
资源组可能正在运行SET和SHOW命令，这些命令会绕过资源组事务检查。

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfldist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

一个提供对表的递增和并发ANALYZE操作的工具。对追加优化表来说， analyzedb 只在统计数据不是最新的时候才更新统计信息。

概要

```
analyzedb -d dbname
    { -s schema | 
    { -t schema.table
    [ -i col1[, col2, ...] |
    -x col1[, col2, ...] ] } |
    { -f | --file} config-file }
    [ -l | --list ]
    [ --gen_profile_only ]
    [ -p parallel-level ]
    [ --full ]
    [ -v | --verbose ]
    [ --debug ]
    [ -a ]

analyzedb { --clean_last | --clean_all }
analyzedb --version
analyzedb { -? | -h | --help }
```

描述

analyzedb 工具递增并发地更新Greenplum数据库中指定表的表数据的统计信息。

在执行ANALYZE操作时， analyzedb 创建了表元数据的快照并将它存储在Master主机的磁盘上。只有在表格被修改的情况下才执行ANALYZE操作。如果 表或者分区自从上次分区依赖没有被修改过，则analyzedb 会自动跳过表或分区， 因为它已经包含了最新的统计信息。

- 对追加优化表来说， analyzedb 如果统计数据数据不是最新的，则会逐渐分析统计数据。例如，如果表数据在表的统计数据收集之后改变了。如果没有该表的统计数据，则收集统计数据
- 对于堆表，统计信息总是被更新。

即使表格统计信息是新的，也可以指定--full选项来更新追加优化

gprecoverseg

表的统计信息。

默认情况下，analyzedb 最多创建5个并发会话来并行分析表。对于每个会话，analyzedb 发出一个ANALYZE命令到 数据库并指定不同的表名。-p选项控制最大并发会话数。

分区追加优化表

对于一个分区的追加优化表analyzedb 检查分区表的根分区和叶子分区。如果需要，该工具将更新非当前分区和根分区的统计信息。

GPORCA需要根分区统计信息。默认情况下，如果统计信息不存在，那么analyzedb 工具将收集分区表的根分区的统计信息。如果任何叶子节点有陈旧的统计信息，那么analyzedb 也会刷新根分区的统计信息。刷新根节点的统计信息的成本和分析一个叶子分区相当。用户可以指定 --skip_root_stats来禁用分区表的根分区的统计信息的收集。

注意

如果表已经由DML或DDL命令（包括INSERT, DELETE, UPDATE, CREATE TABLE, ALTER TABLE 和 TRUNCATE）修改，则analyzedb 工具会更新追加优化表的统计信息。该工具通过比较表的元数据和之前analyzedb 操作中所保存的快照目录元数据来确定表是否已经被修改。表元数据的快照作为状态文件存储在Greenplum数据库主数据目录中的db_analyze/<db_name>/<timestamp> 目录中。

该工具不会自动删除旧的快照信息。时间久了，这些快照会占用大量的磁盘空间。为了回收磁盘空间，您可以保留最近的快照信息，将旧的快照删掉。用户可以指定--clean_last或者--clean_all选项来删除由 analyzedb 生成的状态文件。

如果用户未指定表，一组表或模式，那么analyzedb 工具会根据需要收集所有系统目录表和用户定义的表中的统计信息。

外部表不受analyzedb 的影响。

包含空格的表名是不支持的。

在表上运行ANALYZE命令，（不使用analyzedb工具），不会更新表的元数据，analyzedb工具通常使用这些元数据通常来确定表的统计信息是否是最新的表的元数据。

选项

--clean_last

删除由上一次analyzedb 操作生成的状态文件。所有的其他选项，除了-d都会被忽略。

--clean_all

删除由analyzedb 生成的所有状态文件。所有其他操作除了-d都会被忽略。

-d *dbname*

指定包含要分析的表的数据库的名称。如果未指定此选项，则从环境变量PGDATABASE 中读取数据库名称。如果未设置PGDATABASE，则使用连接指定的用户名。

--debug

如果指定，则将日志级别设置为调试。在执行命令期间，调试级别信息被写入日志文件和命令行。这些信息包括公用程序执行的命令和每个ANALYZE操作的持续时间。

-f *config-file* | --file *config-file*

包含要分析的表的列表的文本文件。可以指定当前目录的相对文件路径。

该文件每行列出一个表。表名必须使用模式名称进行限定。或者，可以使用-i或 -x来指定列名。文件中不允许其他选项。其他诸如--full 的选项必须在命令行中指定。

只有一个选项可用于指定要分析的文件：-f或--file, -t ,或 -s。

当在多个表上执行ANALYZE操作时候，analyzedb 会创建并行会话来并行分析表。该-p选项控制并发会话的最大数目。

在下面的示例中，第一行施加了一个ANALYZE操作对表public.nation，第二行仅对public.lineitem表的l_shipdate和l_receiptdate列施加了一个ANALYZE操作。

```
public.nation
            public.lineitem -i l_shipdate,
l_receiptdate
```

--full

对所有指定的表执行ANALYZE操作。即使统计数据是最新的，也会执行该操作。

--gen_profile_only

不执行ANALYZE操作，但是更新analyzedb收集的表统计信息快照。如果有其他选项制定了表或者模式名，该工具仅更新指定表的快照信息。

如果ANALYZE命令在数据库表上运行并且您想更

新analyzedb该表 的快照信息， 请指定该选项。

-i col1, col2, ...

可选的， 必须用-t选项指定。对于使用-t选项指定的表， 仅收集指定列的统计信息。

只能指定-i, 或-x。这两个选项不能同时指定。

-l | --list

列出将用指定选项进行分析的表格。ANALYZE操作不执行。

-p parallel-level

并行分析的表的数量。并行级别可以是1到10之间的整数， 包含首尾值。默认值是5.

-s schema

指定要分析的模式。模式中的所有表将被分析。只能在命令行中指定一个模式名称。

只有一个选项可用于指定要分析的文件：-f或--file, -t, 或-s。

-t schema.table

只收集schema.table的统计信息。表名称必须使用模式名称进行限定。在命令行上只能指定一个表名。用户可以指定-f 选项来指定文件中的多个表，或者指定-s选项来指定模式中的所有表。

只有其中一个选项可用于指定要分析的文件:-f或--file, -t, 或-s。

-x col1, col2, ...

可选的， 必须用-t选项来指定。对于使用-t选项指定的表，请排除指定列上的收集的统计信息。统计信息只收集在未列出的列上。

只能指定-i, 或-x。这两个选项不能同时指定。

-a

静默模式， 不提示用户确认。

-h | -? | --help

展示在线帮助。

-v | --verbose

如果指定，则将日志记录级别设置为verbose， 以在命令执行过程中向日志文件和命令行写入附加信息。这些信息包括所有要分析的表的列表（包括分区表的子叶子分区）。输出还包括每个ANALYZE 操作操作的持续时间。

--version

显示此工具的版本。

示例

仅收集一组表格列的统计信息的示例。在数据库mytest中， 收集表public.orders上的shipdate和receiptdate 列的统计信息：

```
analyzedb -d mytest -t public.orders -i shipdate,
receiptdate
```

一个在表上收集统计信息并排除一组列的示例。在数据库mytest中，收集表public.foo 的统计数据，并且不收集bar和test2列的统计信息。

```
analyzedb -d mytest -t public.foo -x bar, test2
```

指定包含表列表的文件的示例。此命令收集名为mytest的数据库中文件 analyze-tables列出的表的统计信息。

```
analyzedb -d mytest -f analyze-tables
```

如果用户未指定表，一组表或模式，则analyzedb 实用程序会根据需要收集所有目录表 和用户定义的表上的统计信息。此命令刷新数据库mytest中的系统目录表和用户定义的表上的表统计信息。

```
analyzedb -d mytest
```

用户可以创建一个PL/Python函数来运行analyzedb程序作为Greenplum的数据库函数。这个例子CREATE FUNCTION命令创建一个用户定义的PL/ Python函数，运行 analyzedb工具并在命令行上显示输出。指定analyzedb选项作为函数参数。

```
CREATE OR REPLACE FUNCTION analyzedb(params TEXT)
RETURNS VOID AS
$BODY$
import subprocess
cmd = ['analyzedb', '-a'] + params.split()
p = subprocess.Popen(cmd, stdout=subprocess.PIPE,
stderr=subprocess.STDOUT)

# verbose output of process
for line in iter(p.stdout.readline, ''):
    plpy.info(line);

p.wait()
$BODY$
LANGUAGE plpythonu VOLATILE;
```

当这个SELECT命令由gpadmin用户运行时，该analyzedb工具对数据库mytest中的public.mytable表执行一个分析操作。

```
SELECT analyzedb(''-d mytest -t public.mytable') ;
```

Note: 注意：要创建一个PL/Python函数，该PL/Python程序语言必须在数据库中注册。例如，以管理员运行的 CREATE LANGUAGE命令将吧PL/Python注册成不受信任的语言：

```
CREATE LANGUAGE plpythonu;
```

另见

[ANALYZE](#)

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

激活备机并使其成为Greenplum数据库系统的活动主机。

概要

```
gpactivatestandby [-d standby_master_datadir] [-f] [-a] [-q]
```

```
[ -l logfile_directory]
```

```
gpactivatestandby -v
```

```
gpactivatestandby -? | -h | --help
```

描述

gpactivatestandby工具激活了一台备份主机上的后备Master实例，并将其作为Greenplum数据库系统的活动Master实例运行。被激活的后备Master节点有效地成为Greenplum数据库的Master来接受端口上的客户端连接。

在初始化后备Master时，默认情况下使用与活动Master相同的端口。有关后备Master端口的信息，请参考[gpinitstandby](#)。

用户必须从正在激活的Master主机上运行此工具，而不是在被停用的故障Master主机上运行。运行此工具时候假设用户为系统配置了一台后备Master主机。（参见[gpinitstandby](#)）。

该工具执行以下步骤：

- 停止后备Master上的同步进程 (*walreceiver*)
- 使用日志更新后备Master的系统目录表
- 激活后备Master成为系统中新的活动Master
- 重启带有新Master主机的Greenplum数据库系统

在Greenplum的主Master主机变得不可操作时，备份的后备Master主机充当“温备份”的角色。后备Master通过事务日志复制进程（*walsender*和*walreceiver*）保持最新状态，这两个进程一个运行在主Master，一个运行在后备Master，并使得主Master和后备Master之间保持数据同步。

gprecoverseg

如果主Master出现故障，日志复制进程将关闭，后备Master可以通过使用gpactivatestandby 工具在其位置激活。一旦后备Master激活，复制的日志将用于重建最后一次成功提交事务之后Greenplum中Master主机的状态。

为了使用gpactivatestandby来激活一个新的主Master主机，之前充当主Master的Master主机则不能再运行。该工具会检查已被禁用的Master主机的数据目录中的postmaster.pid文件，如果发现该文件，则会假定旧的Master机仍处于活动状态。在某些情况下，在运行gpactivatestandby（例如，如果被禁用的Master主机进程被意外终止）之前，用户可能需要从被禁用的Master主机的数据目录中删除 postmaster.pid。

激活后备Master之后，运行ANALYZE来更新数据库查询统计信息。例如：

```
psql dbname -c 'ANALYZE;'
```

将后备的Master激活成为主Master之后，该Greenplum数据库系统将不再配置有后备Master。用户可能想要用 [gpinitstandby](#) 工具指定另外一台主机作为新的后备。

选项

-a (不要提示)

不要提示用户确认。

-d *standby_master_datadir*

用户正在激活的Master主机的数据目录的绝对路径。

如果未指定此选项，gpactivatestandby将使用用户正在激活的Master主机上的 MASTER_DATA_DIRECTORY环境变量设置。如果指定了这一选项，它会覆盖MASTER_DATA_DIRECTORY的设置。

如果无法确定目录，则该工具将返回错误。

-f (强制激活)

使用此选项强制激活备份的Master主机。只有在确定后备Master主机和主Master主机一致的情况下，才能使用此选项。

-l *logfile_directory*

写入日志文件的目录。默认为~/gpAdminLogs。

-q (没有屏幕输出)

以静默模式运行。命令输出不会显示在屏幕上，但是仍然会写入日志文件。

-v (显示工具的版本)

显示这一工具的版本、状态、上次更新的日期和校验和。

-? | -h | --help (帮助)

显示在线帮助。

示例

激活后备Master主机并使其成为Greenplum数据库系统的活动Master实例（从正在激活的备份Master主机运行）：

```
gpactivatestandby -d /gpdata
```

另见

[gpinitsystem](#), [gpinitstandby](#)

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

将镜像Segment添加到最初没有配置镜像的Greenplum数据库系统。

概要

```
gpaddmirrors [-p port_offset] [-m datadir_config_file [-a]]  
[-s]  
[-d master_data_directory] [-B  
parallel_processes] [-l logfile_directory]  
[-v]  
  
gpaddmirrors -i mirror_config_file [-a] [-d  
master_data_directory]  
[-B parallel_processes] [-l  
logfile_directory] [-v]  
  
gpaddmirrors -o output_sample_mirror_config  
[-s] [-m datadir_config_file]  
  
gpaddmirrors -?  
  
gpaddmirrors --version
```

描述

gpaddmirrors工具为初始仅配置了主Segment实例的现有Greenplum数据库系统配置镜像 Segment实例。该工具创建镜像实例并开始主Segment实例和镜像Segment实例之间的在线复制进程。一旦所有的镜像与其主Segment同步好，用户的Greenplum数据库系统就建立了完全的数据冗余。

Important: 在在线复制进程中，Greenplum数据库应处于静止状态，负载和其他查询不应该运行。

默认情况下，该工具将提示用户输入将创建镜像Segment数据目录的文件系统位置。如果用户不想被提示，可以使用-m选项传递包含文件系统位置的文件。

镜像位置和端口必须与用户的主Segment数据位置和端口。如果用户创建了额外的文件空间，则还将提示用户为每个文件空间提供镜像位置。

gprecoverseg

该工具使用预定义的命名习惯在指定位置中为每个镜像Segment实例创建唯一的数据目录。必须为镜像Segment 实例声明与主Segment实例相同数量的文件系统位置。如果用户希望在同一位置创建镜像数据目录，可以多次指定同样的目录名称，或者可以为每个镜像输入不同的数据位置。对于文件系统位置，请输入绝对路径。例如：

```
Enter mirror segment data directory location 1 of 2 >
/gpdb/mirror
Enter mirror segment data directory
location 2 of 2 > /gpdb/mirror
```

或

```
Enter mirror segment data directory location 1 of 2 >
/gpdb/m1
Enter mirror segment data directory
location 2 of 2 > /gpdb/m2
```

还有，用户可以运行gpaddmirrors工具，并使用-i选项提供详细的配置文件。如果用户希望镜像Segment位于完全不同于主Segment的主机集合上，这非常有用。镜像配置文件的格式是：

```
mirror<row_id>=<contentID>:<address>:<port>:<data_dir>
```

此处row_id是文件中的行号，contentID是segment实例的content_id，address是segment所在的主机名或IP地址，port是交互端口，data_dir是segment实例数据目录。

例如：

```
mirror0=0:sdwl-1:60000:/gpdata/mir1/gp0
mirror1=1:sdwl-1:60001:/gpdata/mir2/gp1
```

gp_segment_configuration系统目录表可以帮助用户确定当前的主Segment配置，以便用户可以规划镜像Segment配置。例如，运行以下的查询：

```
=# SELECT dbid, content, address as host_address, port,
datadir
FROM gp_segment_configuration
ORDER BY dbid;
```

如果在后补的镜像主机上创建镜像，则新的镜像Segment主机必须预先安装Greenplum数据库软件，并且被配置为与现有的主Segment主机完全相同。

用户必须确保运行gpaddmirrors的用户（gpadmin用户）有权在指定的数据目录位置写入。用户可能想要在Segment主机上创建这些目录并且在运行gpaddmirrors之前将这些目录的拥有者改为（chown）适当的用户。

Note: 该工具在系统内部采用SSH连接执行各项操作任务。在大型Greenplum集群、云部署或每台主机部署了大量的segment实例时，可能会遇到超过主机最大授权连接数限制的情况。此时需要考虑更新SSH配置参数MaxStartups以提高该限制。更多关于SSH配置的选项，请参考您的Linux分发版的SSH文档。

选项

-a (不提示)

以静默模式运行 - 不提示输入信息。如果使用这一选项，必须使用-m或-i提供一个配置文件。

-B *parallel_processes*

并行启动的镜像设置进程的数量。如果未指定，则该工具将启动最多10个并行进程，具体取决于需要设置多少个镜像Segment实例。

-d *master_data_directory*

主数据目录。如果没有指定，将使用为\$MASTER_DATA_DIRECTORY设置的值。

-i *mirror_config_file*

一个配置文件，每个要创建的镜像Segment对应一行。用户必须为系统中的每个主Segment都在该文件中列出一个镜像Segment。该文件格式如下
(如gp_segment_configuration 目录表中的每个属性)：

```
mirror<row_id>=<contentID>:<address>:<port>:<data_dir>
```

此处row_id是文件中的行号，contentID是segment实例的content_id，address是segment所在的主机名或IP地址，port是交互端口，data_dir是segment实例数据目录。

-l *logfile_directory*

用来写日志的目录，默认为~/gpAdminLogs。

-m *datadir_config_file*

包含将创建镜像数据目录的文件系统位置列表的配置文件。如果未提供，该工具将提示用户输入位置。文件中的每一行都指定一个镜像数据目录位置。例如：

```
/gpdata/m1
/gpdata/m2
/gpdata/m3
/gpdata/m4
```

-o *output_sample_mirror_config*

如果用户不确定如何布置-*i*选项使用的镜像配置文件，用户使用这一选项运行 `gpaddmirrors` 来生成一个基于主Segment配置的镜像配置文件示例。该工具将提示用户输入用户的镜像Segment数据目录位置（除非用户使用-*m*在一个文件中提供了这些位置）。然后，用户可以编辑此文件根据需要将主机名更改为主机的镜像主机。

-p *port_offset*

可选。这个数字被用于计算用于镜像Segment的数据库端口和复制端口。默认偏移量为1000。镜像端口分配计算如下：

主机端口 + 偏移量 = 镜像数据库端口

例如，如果主Segment的端口为50001，则默认情况下，其镜像将使用数据库端口51001。

-s (散布镜像)

在可用主机上散布镜像Segment。默认情况下，将把一组镜像Segment集合在一起放在不同于其主Segment集合的后补主机上。镜像散布将把每个镜像放置在Greenplum数据库阵列中的不同主机上。只有在阵列中有足够数量的主机（主机数大于每个主机的Segment实例数）时，才允许使用散布镜像方式。

-v (详细模式)

将日志记录输出设置为详细。

--version (显示工具版本)

显示该工具的版本。

-? (help)

显示在线帮助。

示例

使用和主要数据相同的一组主机向现有的Greenplum数据库系统添加镜像。通过在当前主Segment端口号上加上100来计算镜像数据库端口和复制端口：

```
$ gpaddmirrors -p 100
```

使用和主要数据不同的一组主机向现有的Greenplum数据库系统添加镜像：

```
$ gpaddmirrors -i mirror_config_file
```

其中mirror_config_file看起来像这样：

```
mirror0=0:sdw1-1:52001:/gpdata/mir1/gp0
    mirror1=1:sdw1-2:52002:/gpdata/mir2/gp1
    mirror2=2:sdw2-1:52001:/gpdata/mir1/gp2
    mirror3=3:sdw2-2:52002:/gpdata/mir2/gp3
```

使用gpaddmirrors -o输出一个要使用的镜像配置文件样例：

```
$ gpaddmirrors -o /home/gpadmin/sample_mirror_config
```

另见

[gpinitstandby](#), [gpactivatestandby](#)

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gpkg

gprecoversea

创建一个可以给gprestore工具使用的Greenplum数据库备份。

概要

```
gpbackup --dbname database_name
  [--backup-dir directory]
  [--compression-level level]
  [--data-only]
  [--debug]
  [--exclude-schema schema_name]
  [--exclude-table schema.table]
  [--exclude-table-file file_name]
  [--include-schema schema_name]
  [--include-table schema.table]
  [--include-table-file file_name]
  [--incremental [--from-timestamp backup-timestamp]]
  [--jobs int]
  [--leaf-partition-data]
  [--metadata-only]
  [--no-compression]
  [--plugin-config config_file_location]
  [--quiet]
  [--single-data-file]
  [--verbose]
  [--version]
  [--with-stats]
```

gpbackup --help

描述

gpbackup工具用来备份数据库的内容到元数据文件集合和数据文件集合，这些文件可以被 gprestore工具用来恢复数据库。当备份数据库时，可以通过指定表级别和模式级别选项来 备份某些特定的表。例如，可以通过组合模式级别和表级别选项来备份模式下的所有表，并排除某一张单独的表。

默认情况下， gpbackup备份指定数据库的对象和Greenplum数据库系统全局对象。可以通过 gprestore工具指定可选参数--with-globals来 全局对象。更多 信息请见[备份或还原中包含的对象](#)。

gpbackup默认将Greenplum数据库备份对象元数据文件和DDL文件存储在Master数据目录下。 Greenplum Segment使用COPY ... ON SEGMENT命令将数据备份为压缩CSV数据文件，并存储在每个Segment的数据目录下。更多信息请见[理解备份文件](#)。

可以通过指定--backup-dir选项将Master和Segment主机上的数据备份到一个绝

对路径下。可以通过指定其他选项来过滤备份集合来排除或包含指定的表。

可以通过指定 `--incremental` 选项来启动增量备份。增量备份在追加优化表或表分区上的变化数据小于未发生变化的数据时有效。有关增量备份的详细信息，请见[使用gpbackup和gprestore创建增量备份](#)。

指定`--jobs`选项（1 job），Greenplum数据库Master主机上的每个 `gpbackup` 操作都会启动一个单独的事务。`COPY ... ON SEGMENT`命令在每个Segment主机上并行执行备份任务。备份进程会在备份的每张表上唤起ACCESS SHARE锁。在表锁处理过程中，数据库处于静止状态。

当备份操作完成后，`gpbackup`会返回状态码。详情请见[返回码](#)。

`gpexpand`正在初始化新segments时，`gpbackup`工具不能运行。集群扩展完成后，在扩展之前创建的备份不能被`gprestore`使用。

`gpbackup`可以在备份操作完成后发送Email状态通知。客户可以在配置文件中创建工具发送和接收服务器信息。详情请见[配置邮件通知](#)。

Note: 该工具在系统内部采用SSH连接执行各项操作任务。在大型Greenplum集群、云部署或每台主机部署了大量的 segment实例时，可能会遇到超过主机最大授权连接数限制的情况。此时需要考虑更新SSH配置参数`MaxStartups`以提高该限制。更多关于SSH配置的选项，请参考您的Linux分发版的SSH文档。

选项

--dbname *database_name*

必须。指定要备份的数据库。

--backup-dir *directory*

可选。复制所有备份文件（元数据文件和数据文件）到指定路径。必须指定*directory*为绝对路径（不能是相对路径）。如果不提供该选项，元数据文件会保存在Greenplum Master主机的

`$MASTER_DATA_DIRECTORY/backups/YYYYMMDD/YYYYMMDDhhmmss/`

路径下。Segment主机的CSV数据文件保存

在`<seg_dir>/backups/YYYYMMDD/YYYYMMDDhhmmss/`路径下。当指

定该选项时，文件会被复制到备份目录的子路径下。

该选项不能和`--plugin-config`一起使用。

--compression-level *level*

可选。指定用来压缩数据文件的gzip压缩级别（从1到9）。默认为1。注意，`gpbackup`默认使用压缩。

--data-only

可选。仅将表数据备份到CSV文件，不备份用来重建表和其他数据库对象的元数据文件。

--debug

可选。显示操作期间的调试信息。

--exclude-schema *schema_name*

可选。指定要被排除备份的模式名。该参数可以指定多次以排除多个模式。该选项不能与`--include-schema`一起使用，也不能与表过滤选项例如：

--include-table一起使用。更多信息请见[过滤备份或恢复的内容](#)。

--exclude-table schema.table

可选。指定要排除在备份中的表。被排除的表必须为格式<schema-name>.<table-name>。如果表或模式名使用了任何非小写字母、数字或下划线，必须用双引号包裹名字。该选项可以指定多次。该选项不能与--exclude-schema或--include-table同时使用。

该选项不能和--leaf-partition-data组合使用。虽然您可以指定子分区名字，但是gpbackup会忽略分区名称。

更多信息详见[过滤备份或恢复的内容](#)。

--exclude-table-file file_name

可选。指定一个包含需要排除在备份之外的列表文件。每行必须指定一个单独的表，格式为 <schema-name>.<table-name>。文件不能包含多余的行。如果表或模式名使用了任何非小写字母、数字或下划线，必须用双引号包裹名字。该选项可以指定多次。该选项不能与--exclude-schema或--include-table同时使用。

该选项不能和--leaf-partition-data组合使用。虽然您可以在--exclude-table-file 中指定子分区名字，但是gpbackup会忽略分区名称。

更多信息请见[过滤备份或恢复的内容](#)。

--include-schema schema_name

可选。指定要包含在备份中的数据库模式名。可以多次指定该参数来包含多个备份模式。如果指定该选项，任何不包含在--include-schema中的模式都不会放在备份集中。该选项不能与选项 --exclude-schema、--include-table或 --include-table-file一起使用。更多详情请见[过滤备份或恢复的内容](#)。

--include-table schema.table

可选。指定一个要包含在备份中的表。该表的格式必须为<schema-name>.<table-name>。如果表或模式名使用了任何非小写字母、数字或下划线，必须用双引号包裹名字。有关模式和表名中支持的字符信息详情，请见[模式和表名](#)中的模式和表名部分。

该选项可以指定多次。该选项不能和模式筛选选项--include-schema或--exclude-table-file等一起使用。

客户也可以指定序列或视图的名字。

如果指定该选项，该工具不会自动备份依赖对象。客户必须指定相关依赖对象。例如，如果备份了视图，客户必须备份试图所用的表。如果备份表用到了一个序列，客户必须也备份相关的序列。

可以在表名处通过指定--leaf-partition-data选项来指定表子分区名字，仅备份指定的分区。当指定子分区备份时，子分区数据和分区表的元数据信息都会被备份。

更多信息请见[过滤备份或恢复的内容](#)。

--include-table-file file_name

可选。指定一个备份时需要的表名列表文件。文件中的每行必须定义为单独的表名，格式为：<schema-name>.<table-name>。该文件不能包含多余的列。有关模式和表名中支持的字符信息详情，请见[模式和表名](#)中的模式和表名部分。

任何没有列在文件中的表都会被排除在备份集以外。该选项不能与模式过滤选项--include-schema 或--exclude-table-file等一起使用。

该选项也可以指定序列和视图名称。

如果指定该选项，该工具不会自动备份依赖对象。客户必须指定相关依赖对

象。例如，如果备份了视图，客户必须备份试图所用的表。如果备份表用到了一个序列，客户必须也备份相关的序列。

可以在表名处通过指定--leaf-partition-data选项来指定表子分区名字，仅备份指定的分区。当指定子分区备份时，子分区数据和分区表的元数据信息都会被备份。

[更多信息请见过滤备份或恢复的内容。](#)

--incremental

指定该选项可以增加一个增量备份到增量备份集中。一个备份集包括一个全量备份和一个或多个增量备份。该备份集合必须是连续性的，以保证在恢复时是可用的。

默认情况下，gpbackup会找出最近的备份。如果该备份是全备份，工具会创建一个备份集合。如果备份为增量备份，工具会将备份增加到已经存在的备份集。增量备份被增加到备份集的最后。可以通过指定--from-timestamp来覆盖默认的行为。

--from-timestamp *backup-timestamp*

可选。指定备份时间戳。指定的备份必须已经有增量备份存在。如果指定的备份只有全量备份，该工具会创建一个备份集。如果指定的备份是增量备份，工具会将增量备份直接备份到该备份集合。

该选项必须与--leaf-partition-data选项一起使用。不能与--data-only 或--metadata-only一起使用。

如果备份操作不能增加备份到已经存在的增量备份集或者不能使用备份来创建一个备份集，工具将不能创建备份 并且工具会返回错误。

有关创建和使用增量备份的详细信息，请见[使用gpbackup和gprestore创建增量备份](#)。

--jobs *int*

可选。指定进行表备份的并行任务数量。默认gpbackup采用1个任务（数据库连接）。增加该参数的值可以提高数据备份的速度。多运行多个任务时，每个任务会在一个单独的事务中备份表。例如，如果指定--jobs 2，工具会创建2个进程，每个进程启动一个单独的事务，工具会使用这两个进程并行的备份表。

Important: 如果指定超过1的值，数据库在工具从表上获取备份用的锁时处于静默状态。如果数据库操作正在那些被备份的表上执行，此时进行备份的话这些表在不同事务中的锁处理和一致性问题不能被保证。

该选项不能和--metadata-only、--single-data-file或--plugin-config一起使用。

--leaf-partition-data

可选。对于分区表，该选项定义后会为每一个子分区创建一个单独的数据文件，而不是为整个表创建一个完成的文件（默认）。使用该选项要求客户通过--include-table-file选项指定包含在备份中的单独子节点分区。该选项不能与--exclude-table-file或--exclude-table选项同时使用。

--metadata-only

可选。仅创建可用来重建数据库的元数据文件（DDL），不备份表的实际数据。

--no-compression

可选。不对表数据CSV文件进行压缩。

--plugin-config *config-file_location*

指定gpbackup插件配置文件的位置，该文件是一个YAML格式的文本文

件。该文件包含gpbackup在备份操作期间使用的配置信息。

如果在备份数据库时指定了--plugin-config选项，那么在从备份恢复数据库时也需要同样指定对应的配置信息。

该选项不能与--backup-dir一起使用。

有关使用存储插件应用的详细信息，请见[使用gpbackup存储插件](#)部分。

--quiet

可选。不显示所有非告警、非错误日志信息。

--single-data-file

可选。在每个Segment主机上为所有备份的表创建一个单独的数据文件。默认情况下，每个gpbackup会为每张表创建一个压缩CSV文件。

Note: 如果选用--single-data-file选项来将每个Segment上的数据备份为1个文件，就不能在使用gprestore时指定--jobs选项为超过1的值来执行并行恢复。

--verbose

可选。打印详细日志信息。

--version

可选。打印工具版本号并退出。

--with-stats

可选。在备份集中包含查询计划统计信息。

--help

显示在线帮助信息。

返回码

gpbackup完成后会返回如下返回码之一。

- **0** – 成功完成备份。
- **1** – 备份完成，没有严重错误。具体信息见日志文件。
- **2** – 备份失败，有严重错误。具体信息见日志文件。

模式和表名

当指定表过滤项--include-table或--include-table-file为列表文件时，gpbackup支持备份模式或表名包含以下特殊字符时。

~ # \$ % ^ & * () _ - + [] { } > < \ | ; : / ? ! ,

如果在命令行指定--include-table选项时，名字包含大写字母或特殊字符时，名字必须被单引号包裹。

```
gpbackup --dbname test --include-table 'my#1schema'.my_$42_Table'
```

当使用--include-table-file选项，表名被放在文件中时，不需要使用单引号。例如，使用--include-table-file选项备份两个表。

```
my#1schema.my_$42_Table
```

```
my#1schema.my_$590_Table
```

Note: `--include-table`和`--include-table-file`选项不支持 模式名或表名包含双引号 ("")、点号 (.)、换行符 (\n) 或空格符 ()。

示例

备份"demo"数据库中的所有模式和表，包括Greenplum数据库全局系统对象：

```
$ gpbackup --dbname demo
```

备份"demo"数据库中除了"twitter"模式以外的所有模式和表：

```
$ gpbackup --dbname demo --exclude-schema twitter
```

仅备份"demo"数据库中的"twitter"模式：

```
$ gpbackup --dbname demo --include-schema twitter
```

备份"demo"数据库中的所有模式和表，包括Greenplum数据库全局系统对象，将所有备份文件复制到 /home/gpadmin/backup路径下。

```
$ gpbackup --dbname demo --with-stats --backup-dir  
/home/gpadmin/backup
```

该示例使用`--include-schema`和`--exclude-table`备份指定模式，除了单独的一张表。

```
$ gpbackup --dbname demo --include-schema mydata --exclude-table  
mydata.addresses
```

`--exclude-schema`选项不能与表过滤选项一起使用，例如`--include-table`。

另见

[gprestore](#)、[使用gpbackup和gprestore并行备份](#)和[使用带有gpbackup和gprestore的S3存储插件](#)

[工具指南](#)

[管理工具参考](#)

[analyzedb](#)

[gpactivatestandby](#)

[gpaddmirrors](#)

[gpbackup](#)

gpcheck

[gpcheckcat](#)

[gpcheckperf](#)

[gpconfig](#)

[gpdeletesystem](#)

[gpexpand](#)

[gpfdist](#)

[gpinitstandby](#)

[gpinitsystem](#)

[gupload](#)

[gplogfilter](#)

[gpmapreduce](#)

[gpmovemirrors](#)

[gpperfmon_install](#)

[gppkg](#)

检查和验证Greenplum数据库平台的设置。

概要

```
gpcheck {{-f | --file} hostfile_gpcheck | {-h | --host}
host_ID| --local }
[-m master_host] [-s standby_master_host]
[--stdout | --zipout]
[--config config_file]

gpcheck --zipin gpcheck_zipfile

gpcheck -?

gpcheck --version
```

描述

gpcheck 工具确定用户正在运行Greenplum数据库的平台，并且验证各种平台相关的配置设置。**gpcheck** 可以使用一个主机文件或者之前由--zipout选项创建的文件来验证平台设置。在成功的验证过程结束时，将显示GPCHECK_NORMAL消息。如果显示 GPCHECK_ERROR，则一个或多个验证检查失败。用户也可以使用**gpcheck** 来收集和查看主机上的平台设置，而不运行验证检查。

用户应该以root用户运行**gpcheck**。如果用户没有以root 用户运行**gpcheck**，该工具将显示一条警告消息，并且将无法验证所有的配置设置；只有其中的一部分设置将被验证。

选项

--config config_file

替代默认配置文件\$GPHOME/etc/gpcheck.cnf (或者Dell EMC Greenplum Data Computing Appliance 上的~/gpconfigs/gpcheck_dca_config) 配置文件的名字。该文件指定要运行的OS相关的检查。

{-f | --file} hostfile_gpcheck

gprecoverseg

包含被gpcheck用来验证平台相关设置的主机列表的文件名称。该文件应包含 Greenplum数据库系统中所有主机（Master、后备Master和Segment）的单一主机名。gpcheck使用SSH连接到这些主机。

{--h | --host} host_ID

在由host_ID指定的Greenplum数据库系统中的主机上检查平台相关的设置。gpcheck使用SSH连接到该主机。

--local

检查运行gpcheck的Segment主机上的平台相关设置。这一选项不需要SSH认证。

-m master_host

此选项已弃用，将在以后的版本中删除。

-s standby_master_host

此选项已弃用，将在以后的版本中删除。

--stdout

显示从gpcheck收集的主机信息。不执行检查或验证。

--zipout

将所有收集的数据保存到当前工作目录中的一个.zip文件中。gpcheck会自动创建.zip文件并将其命名为gpcheck_timestamp.tar.gz。不执行检查或验证。

--zipin gpcheck_zipfile

使用此选项解压缩并检查--zipout选项创建的.zip文件。

gpcheck根据用户在此选项中指定的文件执行验证任务。

-? (help)

展示在线帮助。

--version

显示工具的版本。

示例

通过输入主机文件验证Greenplum数据库平台设置：

```
# gpcheck -f hostfile_gpcheck
```

将Greenplum数据库平台设置保存到一个zip文件：

```
# gpcheck -f hostfile_gpcheck --zipout
```

使用--zipin选项创建的zip文件验证Greenplum数据库平台设置：

```
# gpcheck --zipin gpcheck_timestamp.tar.gz
```

查看收集的Greenplum数据库平台设置：

```
# gpcheck -f hostfile_gpcheck --stdout
```

另见

[gpssh](#)、[gpscp](#)、[gpcheckperf](#)

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

gpcheckcat工具测试Greenplum数据库目录表的不一致性。

该工具在\$GPHOME/bin/lib中。

概要

```
gpcheckcat [ options ] [ dbname ]
```

Options:

- g dir
- p port
- P password
- U user_name
- s {none | only}
- o
- R test_name
- C catalog_name
- B parallel_processes
- v
- A

```
gpcheckcat -l
```

```
gpcheckcat -?
```

描述

gpcheckcat工具运行多种测试检查数据库目录的不一致。某些测试不能与其他负载语句同时运行，否则结果将不可用。运行gpcheckcat时，要以受限模式重新启动数据库，否则，gpcheckcat可能会由于正在进行的数据库操作而报告不一致，这会与实际的不一致数量不同。如果用户在不停止数据库活动的情况下运行gpcheckcat，请使用-o选项运行它。

Note: 每当用户运行该工具时，它都会检查并删除指定数据库中的孤立的临时数据库方案（没有会话ID的临时方案）。该工具命令行上显示孤立临时模式检查的结果，并在日志中记录该结果。目录不一致是Greenplum数据库系统表之间发生的不一致。一般来说，

gprecoverseg

有三种不一致：

- Segment级别的系统表不一致。例如，包含表数据的系统表与包含列数据的系统表之间的不一致。另一个例子是一个系统表在本该唯一的列中包含重复。
- Segment之间相同系统表之间的不一致。例如，一个系统表在一个Segment上缺少一行，但其他Segment具有这一行。作为另一个例子，特定的行列数据的值在不同的Segment之间不同，例如表所有者或表访问特权。
- 持久化系统表对象状态和文件系统对象间的不一致。例如，一个文件在数据库目录中存在，但数据库系统表中不存在相应的对象。

选项

-A

在Greenplum数据库安装的所有数据库上运行gpcheckcat。

-B *parallel_processes*

并行运行的进程数量。

gpcheckcat工具尝试确定要同时使用的进程数（批尺寸）。

该工具假定它可以为每个进程使用最小为20MB的缓冲区。并行进程的最大数量是Greenplum数据库Segment实例的数量。当该工具开始检查目录时，它会显示所使用的并行进程的数量。

Note: 如果返回的错误数量超过缓冲区大小，则该工具可能会耗尽内存。如果发生内存不足错误，可以使用 -B选项降低批尺寸。例如，如果该工具显示批尺寸为936并且内存不足，则可以指定 -B 468并行运行468个进程。

-C *catalog_table*

对指定的目录表运行交叉一致性、外键和ACL测试。

-g *data_directory*

生成SQL脚本来修复目录不一致。脚本被放置在*data_directory*中。

-l

列出gpcheckcat测试。

-O

只运行可以在线（不受限）模式运行的gpcheckcat测试。

-p *port*

这一选项指定Greenplum数据库使用的端口。

-P *password*

连接到Greenplum数据库的用户的密码。

-R *test_name*

指定要运行的测试。某些测试只有当Greenplum数据库处于受限模式时才能运行。

这些是可以执行的测试：

`acl` - 对访问控制特权的交叉一致性检查

`duplicate` - 检查重复项

`foreign_key` - 检查外键

`inconsistent` - 对Master和Segment不一致性的交叉一致性检查

`missing_extraneous` - 对缺少的或无关的项的交叉一致性检查

`owner` - 检查表的拥有关系是否与Master数据库不一致

`orphaned_toast_tables` - 检查孤立的TOAST表

Note: 有很多种情况都会导致TOAST表变成孤立表，此时不会生成修复脚本，要求进行手工元数据维护。其中一种情况是`pg_class`中的`reltoastrelid`指向一个不正确的TOAST表（不能匹配到TOAST表）。另外一种情况可能是`pg_class`表的`reltoastrelid`和`pg_depend`丢失（双孤立TOAST表）。如果需要手工修复元数据，`gpcheckcat`会显示详细的元数据更新步骤。

`part_integrity` - 检查`pg_partition`分支的完整性、带OID的分区、分区分布策略

`part_constraint` - 检查分区表上的约束

`unique_indexViolation` - 检查有唯一索引约束的列的表中是否有重复项

`dependency` - 检查不存在对象的依赖关系（仅限于受限模式）

`distribution_policy` - 检查随机分布表上的约束（仅限于受限模式）

`namespace` - 检查缺少方案定义的方案（仅限于受限模式）

`pgclass` - 检查没有任何对应的`pg_attribute`项的（仅限于受限模式）`pg_class`项。

`-S {none | only}`

指定这一选项以控制对Greenplum数据库安装中所有数据库（如`pg_database`）上共享的目录表的测试。

值`none`禁用对共享目录表的测试。值`only`仅测试共享目录表。

`-U`

user_name
连接到Greenplum数据库的用户。
 -? (帮助)
显示在线帮助。
 -v (详细模式)
显示所执行测试的详细信息。

注解

该工具可识别缺少属性的表，并将其以非标准格式显示在输出中的各个位置。在显示输出信息之后，该工具还会以格式
database.schema.table.segment_id 显示缺少属性的表的摘要列表。

如果gpcheckcat检测到不一致的OID（对象ID）信息，它将生成一个或者多个包含SQL查询的验证文件。 用户可以运行SQL查询来查看有关OID不一致的详细信息，并调查不一致之处。这些文件在gpcheckcat被调用的目录中生成。

这是该文件的格式：

```
gpcheckcat.verify.dbname.catalog_table_name.test_name.TIMESTAM
```

这是当gpcheckcat检测到数据库mydb中的目录表*pg_type*中的不一致OID（对象ID）信息时，它创建的验证文件名示例：

```
gpcheckcat.verify.mydb.pg_type.missing_extraneous.20150420102
```

这是验证文件中查询的一个示例：

```
SELECT *
  FROM (
    SELECT relname, oid FROM pg_class WHERE reltype
      IN (1305822,1301043,1301069,1301095)
    UNION ALL
    SELECT relname, oid FROM gp_dist_random('pg_class')
  WHERE reltype
    IN (1305822,1301043,1301069,1301095)
  ) altyprelids
 GROUP BY relname, oid ORDER BY count(*) desc ;
```


Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

验证指定主机的基准硬件性能。

概要

```
gpcheckperf -d test_directory [-d test_directory ...]
{-f hostfile_gpcheckperf | -h hostname
[-h hostname ...]}
[-r ds] [-B block_size] [-s file_size]
[-D] [-v| -V]

gpcheckperf -d temp_directory
{-f hostfile_gpchecknet | -h hostname
[-h hostname ...]}
[ -r n|M [--duration time] [--
netperf] ] [-D] [-v | -V]

gpcheckperf -?

gpcheckperf --version
```

描述

gpcheckperf 工具在指定的主机上启动会话并运行以下性能测试：

- **磁盘I/O测试 (dd测试)** — 要测试逻辑磁盘或文件系统的顺序吞吐性能，该工具使用**dd**命令，该命令是一个标准的UNIX工具。它记录花费多长时间在磁盘上读写一个大文件，并以兆字节 (MB) 每秒为单位计算磁盘I/O性能。默认情况下，用于测试的文件尺寸按照主机上的总随机访问内存 (RAM) 的两倍计算。这确保了测试是真正地测试磁盘I/O而不是使用内存缓存。
- **内存带宽测试 (流)** — 为了测试内存带宽，该工具使用STREAM基准程序来测量可持续的内存带宽 (以MB/s为单位)。这测试用户的系统在不涉及CPU计算性能情况下是否受系统内存带宽的限制。在数据集较大的应用程序中 (如在Greenplum数据库中)，低内存带宽是一个主要的性能问题。如果内存带宽明显低于CPU的理论带宽，则会导致CPU花费大量的时间等待数据从系统内存到达。
- **网络性能测试 (gpnetbench*)** — 为了测试网络性能 (以及Greenplum数据库Interconnect的性能)，该工具运行一种网络基



gprecoverseg

准测试程序，该程序当前主机发送5秒钟的数据流 到测试中包含的每台远程主机。数据被并行传输到每台远程主机，并以兆字节 (MB) 每秒报告最小、最大、平均和中位网络传输速率。如果汇总的传输速率比预期慢（小于100MB/s），则可以使用-r n 选项串行地运行该网络测试以获取每台主机的结果。要运行全矩阵带宽测试，用户可以指定-r M，这将导致每台主机都发送和接收来自指定的每台其他主机的数据。该测试最适用于验证交换结构是否可以承受全矩阵负载。

为了指定要测试的主机，请使用-f 选项指定包含主机名列表的文件，或使用 -h 选项在命令行上指名单个主机名。如果运行网络性能测试，主机文件中的所有项必须是同一子网内的网络接口。如果用户的Segment 主机具有在不同子网上配置的多个网络接口，请为每个子网运行一次网络测试。

用户还必须指定至少一个测试目录（使用-d）。运行gpcheckperf 的用户必须具有对所有远程主机上指定测试目录的写入权限。对于磁盘I/O 测试，测试目录应与用户的 Segment 数据目录（主Segment 和/或镜像Segment）相对应。对于内存带宽和网络测试，测试程序文件需要临时目录。

在使用gpcheckperf之前，用户必须在涉及性能测试的主机之间建立可信的主机设置。用户可以使用gpssh-exkeys 工具更新已知主机文件并在主机之间交换公钥（如果尚未这样做的话）。请注意，gpcheckperf 调用gpssh 和gpscp，这些Greenplum工具也必须在\$PATH 中。

选项

-B *block_size*

指定用于磁盘I/O 测试的块大小（以KB或MB为单位）。缺省值是32KB，与Greenplum数据库页面大小相同。最大块大小是1 MB。

-d *test_directory*

对于磁盘I/O 测试，指定要测试的文件系统目录位置。用户必须具有对性能测试中涉及的所有主机上 测试目录的写入权限。用户可以多次使用-d 选项指定多个测试目录（例如，测试主数据目录和镜像数据目录的磁盘I/O）。

-d *temp_directory*

对于网络和流测试，指定单个目录，测试程序文件在测试期间将被复制到该目录。用户必须具有 对测试中涉及的所有主机上该目录的写入权限。

-D (显示每台主机的结果)

报告每个主机的磁盘I/O测试的性能结果。缺省情况下，仅报告具有最低和最高性能的主机的结果，以及所有主机的总体和平均性能。

--duration *time*

以秒 (s)、分钟 (m)、小时 (h) 或天数 (d) 指定网络测试的持续时间。默认值是15秒。

-f *hostfile_gpcheckperf*

对于磁盘I/O和流测试，请指定一个包含将参与性能测试的主机名的文件名称。主机名是必需的，用户可以选择指定每个主机的后补用户名和/或SSH端口号。主机文件的语法是每行一台主机，如下所示：

```
[username@]hostname[:ssh_port]
```

-f *hostfile_gpchecknet*

对于网络性能测试，主机文件中的所有项都必须是同一子网内的主机地址。如果用户的Segment 主机在不同子网上配置有多个网络接口，请为每个子网运行一次网络测试。例如（包含互连子网1 的Segment主机地址名的主机文件）：

```
sdw1-1
sdw2-1
sdw3-1
```

-h *hostname*

指定将参与性能测试的单个主机名（或主机地址）。用户可以多次使用-h 选项来指定多个主机名。

--netperf

指定应该用netperf二进制文件来执行网络测试，而不是Greenplum网络测试。要使用此选项，用户必须从<http://www.netperf.org> 下载 netperf 并且安装到所有Greenplum主机（Master和Segment）的 \$GPHOME/bin/lib 目录中。

-r ds{n|N|M}

指定要运行的性能测试，默认是dsn：

- 磁盘I/O测试 (d)
- 流测试 (s)
- 网络性能测试，串行 (n)、并行 (N) 或全矩阵 (M) 模式。可选的--duration选项指定了运行网络测试的时间（以秒为单位）。要使用并行 (N) 模式，用户必须在偶数台主机上运行测试。

如果用户宁愿使用netperfnetperf (<http://www.netperf.org>) 而不是Greenplum网络测试，用户必须下载它并安装到所有Greenplum主机（Master和Segment）

的\$GPHOME/bin/lib目录中。然后，用户可以指定可选的--netperf选项来使用 netperf二进制文件而不是默认的gpnetbench*工具。

-S file_size

指定用于-d所指定的所有目录的磁盘I/O测试的总文件尺寸。

*file_size*应该等于主机上总RAM的两倍。如果未指定，则默认值是在执行gpcheckperf的主机上的总RAM的两倍，这确保了测试是真正地 测试磁盘I/O而不是使用内存缓存。用户可以以KB、MB或GB为单位指定尺寸。

-v (详细模式) | -V (非常详细模式)

详细 (Verbose) 模式显示性能测试运行时的进度和状态信息。

非常详细 (Very Verbose) 模式显示该工具生成的所有输出消息。

--version

显示该工具的版本。

-? (帮助)

显示在线帮助。

示例

使用/*data1*和/*data2*作为测试目录在文件 *host_file*中的所有主机上运行磁盘I/O和内存带宽测试：

```
$ gpcheckperf -f hostfile_gpcheckperf -d /data1 -d /data2 -r ds
```

在名为*sdw1*和*sdw2* 的主机上只使用测试目录 /*data1*运行磁盘I/O测试。显示单个主机结果并以详细模式运行：

```
$ gpcheckperf -h sdw1 -h sdw2 -d /data1 -r d -D -v
```

使用测试目录/tmp,运行并行网络测试，其中*hostfile_gpcheck_ic**指定同一Interconnect子网内的所有网络接口的主机地址名称：

```
$ gpcheckperf -f hostfile_gpchecknet_ic1 -r N -d /tmp
$ gpcheckperf -f hostfile_gpchecknet_ic2 -r N -d /tmp
```

运行与上面相同的测试，但使用netperf而不是Greenplum网络测试（注意，netperf必须安装在所有Greenplum主机上的\$GPHOME/bin/lib中）：

```
$ gpcheckperf -f hostfile_gpchecknet_ic1 -r N --netperf -d /tmp
$ gpcheckperf -f hostfile_gpchecknet_ic2 -r N --netperf -d /tmp
```

另见

[gpssh](#)、[gpscp](#)、[gpcheck](#)

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

在Greenplum数据库系统中所有的Segment上设置服务器配置参数。

概要

```
gpconfig -c param_name -v value [-m master_value | --masteronly]
| -r param_name [--masteronly | -l
[--skipvalidation] [--verbose] [--debug]

gpconfig -s param_name [--file | --file-compare] [-v
--verbose] [--debug]

gpconfig --help
```

描述

gpconfig工具允许用户在Greenplum数据库系统中所有实例（Master、Segment和镜像）的postgresql.conf文件中设置、复原或查看配置参数。设置参数时，如果需要，还可以为Master指定一个不同的值。例如，诸如max_connections之类的参数要求Master的设置不同于Segment的设置。如果要设置或复原全局参数或仅可对Master设置的参数，请使用--masteronly选项。

gpconfig只能用来管理某些参数。例如，用户不能使用它来设置port等参数，这些参数对每个Segment实例都不同。使用-l (list) 选项查看gpconfig支持的配置参数的完整列表。

当gpconfig在Segment的postgresql.conf 文件中设置配置参数时，新的参数设置将总是显示在该文件的底部。当用户使用gpconfig移除配置参数时，gpconfig会在所有Segment的postgresql.conf文件中把该参数注释掉，从而恢复系统默认设置。例如，如果使用gpconfig 删除（注释掉）一个参数，并且稍后把它添加回来（设置新值），则该参数会有两个实例，一个被注释掉，另一个被启用并添加到postgresql.conf文件的底部。

设置参数之后，用户必须重新启动其Greenplum数据库系统，或者重新加载postgresql.conf 文件以使得更改生效。是否需要重新启动或



gprecoverseg

者加载取决于被设置的参数。

有关服务器配置参数的更多信息，请参阅Greenplum数据库参考指南。

要显示系统中当前参数的设置值，请使用-s选项。

`gpconfig`使用以下环境变量连接到Greenplum数据库的 Master实例并获取系统配置信息：

- PGHOST
- PGPORT
- PGUSER
- PGPASSWORD
- PGDATABASE

选项

-c | --change *param_name*

通过在`postgresql.conf`文件的底部添加新的设置来改变配置参数的设置。

-v | --value *value*

用于由-c选项指定的配置参数的值。默认情况下，此值将应用于所有 Segment及其镜像、Master和后备Master。

非单个字符或数字的参数值必须用单引号包裹（'）。例如，包括空格或特殊字符的字符串 必须用单引号包裹。如果要在字符串参数重嵌入单引号，需要用2个单引号或反斜杠进行转移（\\'）。

工具会在将值写入`postgresql.conf`时带着单引号。

-m | --mastervalue *master_value*

用于由-c选项指定的配置参数的Master值。如果指定，则该值仅适用于Master 和后备Master。该选项只能与-v一起使用。

--masteronly

当被指定时，`gpconfig`将仅编辑Master的`postgresql.conf`文件。

-r | --remove *param_name*

通过注释掉`postgresql.conf`文件中的项删除配置参数。

-l | --list

列出所有被`gpconfig`工具支持的配置参数。

-s | --show *param_name*

显示在Greenplum数据库系统中所有实例（Master和Segment）上使用的配置参数的值。如果实例中参数值存在差异，则工具将

显示错误消息。使用-s选项运行 `gpconfig` 将直接从数据库中读取参数值，而不是从 `postgresql.conf` 文件中读取。如果用户使用 `gpconfig` 在所有 Segment 中设置配置参数，然后运行 `gpconfig -s` 来验证更改，用户仍可能会看到以前的（旧）值。用户必须重新加载配置文件 (`gpstop -u`) 或重新启动系统 (`gpstop -r`) 以使更改生效。

--file

对于配置参数，显示在 Greenplum 数据库系统中的所有 Segment (Master 和 Segment) 上的 `postgresql.conf` 文件中的值。如果实例中的参数值存在差异，则工具会显示一个消息。必须与 -s 选项一起指定。

例如，使用 `ALTER ROLE` 为用户设置配置参数 `statement_mem` 为 64MB，而 `postgresql.conf` 文件中的值为 128MB。运行命令 `gpconfig -s statement_mem --file` 显示 128MB。用户运行的命令 `gpconfig -s statement_mem` 显示 64MB。

与 `--file-compare` 选项一起指定时无效。

--file-compare

对于配置参数，将当前 Greenplum 数据库值与主机 (Master 和 Segment) 上 `postgresql.conf` 文件中的值进行比较。`postgresql.conf files` 文件中的值表示重新启动 Greenplum 数据库时的值。

如果值不一样，该工具显示来自所有主机的值，如果所有主机的值一样，则该程序显示摘要报告。

与 `--file` 选项一起指定时无效。

--skipvalidation

覆盖 `gpconfig` 的系统验证检查，并允许用户对任何服务器配置参数进行操作，包括隐藏参数和 `gpconfig` 无法更改的受限参数。当与 -l 选项（列表）一起使用时，它显示受限参数的列表。

Warning: 使用此选项设置配置参数时要格外小心。

--verbose

在 `gpconfig` 命令执行期间显示额外的日志信息。

--debug

设置日志输出级别为调试级别。

-? | -h | --help

显示在线帮助。

示例

设置所有 Segment 上的 `max_connections` 为 100，而 Master 上为 10。

```
gpconfig -c max_connections -v 100 -m 10
```

该示例展示bash shell字符串处理的语法要求。

```
gpconfig -c search_path -v '"\$user",public'  
gpconfig -c dynamic_library_path -v '\$libdir'
```

配置参数增加到postgresql.conf文件的格式如下。

```
search_path='\"$user\",public'  
dynamic_library_path='\$libdir'
```

注释掉default_statistics_target配置参数的所有实例，并恢复系统默认值：

```
gpconfig -r default_statistics_target
```

列出所有gpconfig支持的配置参数：

```
gpconfig -l
```

显示系统中一个特定配置参数的值：

```
gpconfig -s max_connections
```

另见

[gpstop](#)

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

删除一个使用gpinitsystem初始化的Greenplum数据库系统gpinitsystem。

概要

```
gpdeletesystem [-d master_data_directory] [-B parallel_processes]
                  [-f] [-l logfile_directory] [-D]

gpdeletesystem -?

gpdeletesystem -v
```

描述

gpdeletesystem工具执行以下两种操作：

- 停止所有postgres进程（Segment实例和Master实例）。
- 删除所有的数据目录。

在运行gpdeletesystem之前：

- 将所有备份文件移出Master数据目录和Segment数据目录。
- 确保Greenplum数据库在运行。
- 如果用户当前位于Segment数据目录中，请将目录更改为另一个位置。从Segment数据目录中运行时，该工具会失败，并显示错误。

该工具不会卸载Greenplum数据库软件。

选项

-d master_data_directory

指定Master主机数据目录。如果未指定此选项，则环境变量MASTER_DATA_DIRECTORY的设置。如果指定此选项，则会覆盖MASTER_DATA_DIRECTORY的任何设置。如果无法确定master_data_directory，则该工具返回错误。

`gprecoverseg``-B parallel_processes`

并行删除的Segment数。如果未指定，则该工具将根据需要删除多少个Segment实例启动最多60个并行进程。

`-f (force)`

即使在数据目录中找到备份文件，也强制删除。如果备份文件存在，默认不删除Greenplum数据库实例。

`-l logfile_directory`

写入日志文件的目录。默认为~/gpAdminLogs。

`-D (调试)`

设置日志级别为debug。

`-? (帮助)`

显示在线帮助。

`-v (显示工具版本)`

显示该工具的版本、状态、上次更新的日期和校验和。

示例

删除一个Greenplum数据库系统：

```
gpdeletesystem -d /gpdata/gp-1
```

删除一个Greenplum数据库系统，即使备份文件存在：

```
gpdeletesystem -d /gpdata/gp-1 -f
```

另见

[gpinitsegment](#)

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

在阵列中的新主机上扩展现有的Greenplum数据库。

概要

```
gpexpand [ {-f|--hosts-file} hosts_file ]
          | { -i|--input} input_file [-B batch_size]
          | [{-d | --duration} hh:mm:ss | {-e|--end}
'YYYY-MM-DD hh:mm:ss']
          [-a|--analyze]
          | { -n parallel_processes}
          | { -r|--rollback}
          | { -c|--clean}
          [-v|--verbose] [-s|--silent]
          [{-t|--tardir} directory ]
          [-s|--simple-progress ]
```

gpexpand --?

gpexpand --version

先决条件

- 用户作为Greenplum数据库超级用户 (gpadmin) 登录。
- 新的Segment主机已被根据现有的Segment主机安装和配置。这包括：
 - 配置硬件和操作系统
 - 安装Greenplum软件
 - 创建gpadmin用户帐户
 - 交换SSH密钥
- 用户的Segment主机上有足够的磁盘空间来临时保存最大表的副本。
- 重新分布数据时，Greenplum数据库必须以生产模式运行。Greenplum数据库不能是受限模式或 Master模式。不能指定gpstart的选项-R或者-m 启动Greenplum数据库。

Note: 当gpexpand正在执行segment初始化时，以下工具不能被运行。

- gpbackup

gprecoverseg

- `gpcheck`
- `gpcheckcat`
- `gpconfig`
- `gppkg`
- `gprestore`

有关扩展Greenplum系统的更多信息，请见*Greenplum数据库管理员手册的扩展Greenplum数据库系统*。

描述

`gpexpand`工具分两个阶段执行系统扩展：Segment初始化和表重新分布。

在初始化阶段，`gpexpand`用一个输入文件运行，该文件指定新Segment的数据目录、`dbid`值和其他特征。用户可以手动创建输入文件，也可以在交互式对话中按照提示进行操作。

如果用户选择使用交互式对话创建输入文件，则可以选择指定包含扩展主机列表的文件。在提示输入信息时，如果用户的平台或命令shell限制可键入的主机名列表的长度，则可能不得不使用`-f`指定主机。

除了初始化Segment，初始化阶段还执行这些操作：

- 创建扩展模式`gpexpand`以存储扩展操作的状态，包括表的详细状态。

在表数据重分布阶段，`gpexpand`会重分布表的数据，使数据在新旧segment实例之间平衡。

Note: 数据重新分布应该在低峰（相对高峰期）时段进行。重新分布可以在很长时间内分批次进行。

要开始重分布阶段，可以通过运行`gpexpand`并指定`-d`（运行时间周期）或`-e`（结束时间）选项，或者不指定任何选项。如果客户指定了结束时间或运行周期，工具会在扩展模式下重分布表，直到达到设定的结束时间或执行周期。如果没指定任何选项，工具会继续处理直到扩展模式的表全部完成重分布。每张表都会通过`ALTER TABLE`命令来在所有的节点包括新增加的segment实例上进行重分布，并设置表的分布策略为其原始策略。如果`gpexpand`完成所有表的重分布，它会显示成功信息并退出。

Note: 该工具在系统内部采用SSH连接执行各项操作任务。在大型Greenplum集群、云部署或每台主机部署了大量的segment实例时，可能会遇到超过主机最大授权连接数限制的情况。此时需要考虑更新SSH配置参数MaxStartups以提高该限制。更多关于SSH配置的选项，请参考您的Linux分发版的SSH文档。

选项

-a | --analyze

在扩展后运行ANALYZE更新表的统计信息，默认是不运行ANALYZE。

-B batch_size

在暂停一秒钟之前发送给给定主机的远程命令的批量大小。默认值是16，有效值是1-128。

gpexpand工具会发出许多设置命令，这些命令可能会超出主机的已验证连接的最大阈值（由SSH守护进程配置中的MaxStartups定义）。该一秒钟的暂停允许在gpexpand发出更多命令之前完成认证。

默认值通常不需要改变。但是，如果gpexpand由于连接错误（例如'ssh_exchange_identification: Connection closed by remote host.'）而失败，则可能需要减少命令的最大数量。

-c | --clean

删除扩展模式。

-d | --duration hh:mm:ss

扩展会话的持续时间。

-e | --end ' YYYY-MM-DD hh:mm:ss'

扩展会话的结束日期及时间。

-f | --hosts-file filename

指定包含用于系统扩展的新主机列表的文件的名称。文件的每一行都必须包含一个主机名。

该文件可以包含指定或不指定网络接口的主机名。gpexpand工具处理这两种情况，如果原始节点配置了多个网络接口，则将接口号添加到主机名的末尾。

Note: Greenplum数据库Segment主机的命名习惯是sdwN，其中sdw是前缀并且N是数字。例如，sdw1、sdw2等等。对于具有多个接口的主机，约定是在主机名后面添加破折号（-）和数字。例如sdw1-1和sdw1-2是主机sdw1的两个接口名称。

-i | --input input_file

指定扩展配置文件的名称，其中为每个要添加的Segment包含一行，格式为：

hostname:address:port:datadir:dbid:content:preferred_role

-n parallel_processes

要同时重新分布的表的数量。有效值是1 - 96。

每个表重新分布过程都需要两个数据库连接：一个用于更改表，另一个用于在扩展方案中更新表的状态。在增加-n之前，检查服务器配置参数max_connections的当前值，并确保不超过最大连接限制。

-r | --rollback

回滚失败的扩展设置操作。

-s | --silent

以静默模式运行。在警告时，不提示确认就可继续。

-S | --simple-progress

如果指定，gpexpand工具仅在Greenplum数据库表*gpexpand.expansion_progress*中记录最少的进度信息。该工具不在表*gpexpand.status_detail*中记录关系大小信息和状态信息。

指定此选项可通过减少写入*gpexpand*表的进度信息量来提高性能。

[-t | --tardir] directory

Segment主机上一个目录的完全限定*directory*，gpexpand工具会在其中拷贝一个临时的tar文件。该文件包含用于创建Segment实例的Greenplum数据库文件。默认目录是用户主目录。

-v | --verbose

详细调试输出。使用此选项，该工具将输出用于扩展数据库的所有DDL和DML。

--version

显示工具的版本号并退出。

-? | -h | --help

显示在线帮助。

示例

使用输入文件运行gpexpand以初始化新Segment，并在postgres数据库中创建扩展模式：

```
$ gpexpand -i input_file
```

运行gpexpand最长持续60小时，以将表重新分布给新的Segment：

```
$ gpexpand -d 60:00:00
```

另见

*Greenplum*数据库管理员手册中的[gpssh-exkeys扩展Greenplum系统](#)

◦

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

将数据文件载入Greenplum数据库Segment或从其中写出数据文件到文件系统。

概要

```
gpfdist [-d directory] [-p http_port] [-P last_http_port]
[-l log_file]
      [-t timeout] [-s] [-w time] [-v | -V] [-s] [-m
max_length]
      [--ssl certificate_path [--sslclean wait_time] ]
      [-c config.yml]

gpfdist -? | --help

gpfdist --version
```

描述

gpfdist是Greenplum数据库并行文件分发程序。它可以被外部表和gupload用来并行地将外部表文件提供给所有的Greenplum数据库Segment。它由可写外部表使用，并行接受来自Greenplum数据库Segment的输出流，并将它们写出到文件中。

Note: **gpfdist**和gupload是在Greenplum的主版本级别有效的。例如，Greenplum 4.x版本的**gpfdist**不能用于Greenplum 5.x或6.x版本。

为了使外部表使用**gpfdist**，外部表定义的LOCATION子句必须使用**gpfdist://**协议（参见Greenplum数据库命令CREATE EXTERNAL TABLE）。

Note: 如果--ssl选项被指定来启用SSL安全性，请使用**gpfdists://**协议创建外部表。

使用**gpfdist**的好处是在读取或写入外部表时可以保证最大的并行性，从而提供最佳的性能，并且更容易管理外部表。

对于只读外部表，当用户在外部表中SELECT时，**gpfdist**将数据文件均匀地分析并提供给Greenplum数据库系统的所有Segment实例。对于可写的外部表，**gpfdist**在用户INSERT外部表时接受来

Segment

`gprecoverseg`

自的并行输出流，并写入输出文件。

对于可读外部表，如果被加载的文件使用gzip或bzip2（具有.gz或.bz2的文件扩展名），gpfdist会在装载之前自动解压文件，前提是gunzip或bunzip2在用户的可执行文件路径中。

Note: 目前，可读外部表不支持在Windows平台上的压缩，可写外部表不支持任何平台上的压缩。

当使用gpfdist或gpfdists协议读写数据时，Greenplum数据库在HTTP请求头部中包含X-GP-PROTO，以指示该请求来自Greenplum数据库。该工具拒绝请求头部中不包含X-GP-PROTO的HTTP请求。

大多数情况下，用户很可能希望在ETL机器而不是安装Greenplum数据库的主机上运行gpfdist。要在其他主机上安装gpfdist，只需简单的将该程序复制到该主机上，然后将gpfdist添加到用户的\$PATH路径中。

Note: 使用IPv6时，请始终将数字IP地址包裹在括号内。

选项

`-d directory`

指定一个目录，gpfdist将从该目录中为可读外部表提供文件，或为可写外部表创建输出文件。如果没有指定，默认为当前目录。

`-l log_file`

要记录标准输出消息的完全限定路径和日志文件名称。

`-p http_port`

gpfdist提供文件要使用的HTTP端口。默认为8080。

`-P last_http_port`

gpfdist将会提供文件服务的端口号范围

(`http_port`到`last_http_port`包含最后一个HTTP端口号)。gpfdist会以端口号设定范围内第一个成功绑定的端口号作为服务端口。

`-t timeout`

设置Greenplum数据库建立与gpfdist进程的连接所允许的时间。默认值是5秒。允许的值是2到7200秒(2小时)。在网络流量大的系统上可能要增加。

`-m max_length`

设置以字节为单位的最大数据行长度。默认值是32768。当用户数据包含非常宽的行时(或者当line too long错误消息发

生时) 应该使用, 否则不应该使用, 因为它会增加资源分配。有效范围是32K到256MB (Windows系统上限为1MB)。
Note: 如果用户指定较大的最大行长度并运行大量的gpfdist并发连接, 则可能会发生内存问题。例如, 使用96个并行gpfdist进程需要大约 24GB的内存 ((96 + 1) × 246MB)。

-S

启用简化的日志记录。指定此选项时, 只有具有WARN级别或者更高级别的消息才会写入gpfdist日志文件。INFO级别的消息不写入日志文件。如果未指定这一选项, 则所有gpfdist消息都写入日志文件。

用户可以指定此选项以减少写入日志文件的信息。

-S (use O_SYNC)

使用O_SYNC标志打开同步I/O的文件。任何对结果文件描述的写都会阻塞gpfdist, 直到数据被物理地写到底层硬件。

-w time

设置关闭目标文件 (如命名管道) 之前Greenplum数据库延迟的秒数。默认值是0, 没有延迟。最大值是7200秒 (2小时)。

对于具有多个Segment的Greenplum数据库, 在将不同Segment中的数据写入文件时, Segment之间可能会有延迟。用户可以指定Greenplum数据库关闭文件之前要等待的时间, 以确保所有数据都被写入文件。

--ssl certificate_path

将SSL加密添加到使用gpfdist传输的数据。使用--ssl certificate_path选项执行gpfdist之后, 从此文件服务器加载数据的唯一方法是使用gpfdist://协议。有关gpfdist://协议的信息, 请参阅Greenplum数据库管理员指南中的“装载和卸载数据”部分。

在certificate_path中指定的位置必须包含以下文件:

- 服务器证书文件, server.crt
- 服务器私钥文件, server.key
- 可信证书机构, root.crt

根目录 (/) 不能指定为certificate_path。

--sslclean wait_time

使用--ssl选项运行该工具时, 设置该工具完成向Greenplum数据库Segment读写数据后关闭SSL会话和清除SSL资源之前要延迟的秒数。默认值是0, 没有延迟。最大值是500秒。如果延迟增加, 传输速度降低。

在某些情况下, 复制大量数据时可能会发生此错误: gpfdist server closed connection。为了避免这类错误, 用户可

以增加一个延迟，例如`--sslclean 5`。

`-c config.yaml`

指定gpfdist在装载或抽取数据时，用来选择要应用的转换的规则。gpfdist配置文件是一个YAML1.1文档。

有关文件格式的信息，请参阅Greenplum数据库管理员指南中的[配置文件格式](#)配置件格式。有关用gpfdist配置数据转换的信息，请参阅Greenplum数据库管理员指南中的[使用gpfdist和gupload转换外部数据](#)转换XML数据。

该选项在windows平台中不可用。

`-v` (详细模式)

显示进度和状态信息的详细模式。

`-V` (非常详细模式)

显示由该工具生成的所有输出信息的详细模式。

`-?` (帮助)

显示在线帮助。

`--version`

显示该工具的版本。

注解

服务器配置文件[verify_gpfdists_cert](#)用来控制Greenplum数据库与gpfdist 工具沟通进行读写数据时是否启用SSL授权。当在进行Greenplum数据库外部表与gpfdist 工具测试时，客户可以设置该参数值为`false`来禁用授权。如果参数设置为`false`，以下SSL异常会被忽略：

- gpfdist使用的自有签名SSL证书授权不被Greenplum信任。
- SSL证书中包含的主机名与gpfdist主机名不匹配。

Warning: 禁用SSL证书授权会暴露安全风险，因为gpfdist SSL不会再进行验证。

如果gpfdist工具在读写操作中hang住，客户可以在下次hang住时生成一个 core dump文件来帮助调试问题原因。在gpfdist强制退出之前，设置环境变量GPFDIST_WATCHDOG_TIMER为异常时间段的秒数。当客户设置了该环境变量，并且gpfdist hang住时，工具会在特定秒数后停止并创建 core dump文件，发送相关信息到日志文件。

例子中设置的Linux系统环境变量为gpfdist在无活动操作后300秒(5分钟)退出。

```
export GPFDIST_WATCHDOG_TIMER=300
```

示例

使用端口8081从指定目录提供文件（并在后台启动gpfdist）：

```
gpfdist -d /var/load_files -p 8081 &
```

在后台启动gpfdist并将输出和错误重定向到日志文件：

```
gpfdist -d /var/load_files -p 8081 -l /home/gpadmin/log &
```

停止在后台运行的gpfdist：

--首先找到它的进程ID：

```
ps ax | grep gpfdist
```

--然后杀死该进程，例如：

```
kill 3456
```

另见

[gpload](#)、*Greenplum*数据库管理员指南中的CREATE EXTERNAL TABLE

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

为Greenplum数据库系统添加或初始化后备Master主机。

概要

```
gpinitstandby { -s standby_hostname [-P port] | -r | -n }
[-a] [-q]
[-D] [-S standby_data_directory] [-l
logfile_directory]

gpinitstandby -v

gpinitstandby -?
```

描述

`gpinitstandby`工具为用户的Greenplum数据库系统添加一个备份的后备Master主机。如果用户的系统已经配置了后备Master主机，请在添加新的后备Master主机之前使用`-r`选项将其删除。

在运行此工具之前，请确保Greenplum数据库软件已安装在后备Master主机上，并且用户已在主机之间交换了SSH密钥。推荐在Master主机和后备Master主机上将主端口设置为相同的端口号。

此工具应该在当前活动的主Master主机上运行。方法请参见*Greenplum*数据库安装指南。

该工具执行以下的步骤：

- 更新该Greenplum数据库系统目录来删除现有的后备Master主机信息（如果指定了`-r`）
- 更新该Greenplum数据库系统 目录来添加新的后备Master主机信息
- 编辑Greenplum数据库Master的`pg_hba.conf` 来允许从新添加的后备Master主机进行访问。
- 在后补Master主机上设置后备Master实例。
- 开始同步进程。

如果主Master主机变为不可操作，后备Master主机将作为“热备用”。后备Master通过事务日志 复制进程 (`walsender`和`walreceiver`) 保



gprecoverseg

持最新状态，这些进程在主Master主机和后备Master主机上运行，并使两者的数据保持同步。如果主Master失效，日志复制过程将关闭，后备Master可以使用gpactivatestandby工具在其位置激活。一旦后备Master激活，复制日志将用于重建最后一次成功提交事务时Master主机的状态。

激活的后备Master实际上成为Greenplum数据库的Master，在主端口上接受客户端连接并执行正常的Master操作，例如SQL命令处理和负载管理。

Important: 如果gpinitstandby工具之前初始化后备Master失败，则必须显示删除后备Master数据目录中的文件，然后再次运行gpinitstandby。后备Master数据目录在初始化失败后不会被清理，因为它包含可帮助确定失败原因的日志文件。

如果发生初始化失败，则会在后备主机目录/tmp中生成摘要报告文件。该报告列出了后备主机上需要清理的目录。

选项

-a (不提示)

不要提示用户确认。

-D (调试)

设置日志级别为debug。

-l *logfile_directory*

写入日志的目录，默认为~/gpAdminLogs。

-n (重启被主机)

指定此选项以启动已配置但由于某种原因已停止的Greenplum数据库的后备Master。

-P *port*

此选项指定Greenplum数据库的后备Master使用的端口。默认值是Greenplum数据库的活动Master使用的端口。

如果Greenplum数据库的后备Master和活动Master在同一台主机上，则端口必须不同。如果活动Master和后备Master的端口相同，并且所在的主机也相同，则该工具将返回错误。

-q (无屏幕输出)

以静默模式运行。命令输出不显示在屏幕上，但是仍然写入日志文件。

-r (移除后备Master)

从Greenplum数据库系统中删除当前已配置的后备Master。

-s *standby_hostname*

后备Master的主机名。

-S *standby_data_directory*

新备用Master主机使用的数据目录。默认为和活动Master相同的路径。

如果备用Master和活动Master在同一台主机上，必须使用该选项定义一个不同的路径。

-v (显示工具版本)

显示该工具的版本、状态、上次更新的日期和校验和。

-? (帮助)

显示在线帮助。

示例

添加一个后备Master到用户的Greenplum数据库系统，并重启同步进程：

```
gpinitstandby -s host09
```

开启一个存在的后备Master主机，并且和当前的主Master主机同步：

```
gpinitstandby -n
```

Note: 不要在同一命令指定-n和-s选项。

指定一个不同的端口向Greenplum数据库系统添加一台后备Master主机：

```
gpinitstandby -s myhost -P 2222
```

如果指定的主机名与Greenplum数据库的活动Master相同，则用作后备Master的Greenplum数据库软件必须用-P参数指定与Greenplum数据库的活动Master的软件位于不同位置。另外，后备Master使用的文件空间位置必须与Greenplum数据库的活动Master不同。

从Greenplum系统配置中删除现有的后备Master：

```
gpinitstandby -r
```

另见

[gpinitsystem](#)、[gpaddmirrors](#)、[gpactivatestandby](#)

□ 工具指南

□ 管理工具参考

analyzeedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfldist

gpinitstandby

gpinitsystem

gload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

abrecoversea

使用gpinitsystem_config文件中指定的配置参数初始化一个Greenplum数据库系统。

概要

```
gpinitsystem -c cluster_configuration_file
    [-h hostfile_gpinitsystem]
    [-B parallel_processes]
    [-p postgresql_conf_param_file]
    [-s standby_master_host
        [-P standby_master_port]
        [-S standby_master_datadir | --
         standby_datadir=standby_master_datadir]]
    [-m number | --max_connections=number]
    [-b size | --shared_buffers=size]
    [-n locale | --locale=locale] [--lc-
     collate=locale]
    [--lc-ctype=locale] [--lc-messages=locale]
    [--lc-monetary=locale] [--lc-numeric=locale]
    [--lc-time=locale] [-e password | --
     su_password=password]
    [--mirror-mode={group|spread}] [-a] [-q] [-l
     logfile_directory] [-D]
    [-I input_configuration_file]
    [-O output_configuration_file]

gpinitsystem -v | --version

gpinitsystem -? | --help
```

gpinitsystem

描述

gpinitsystem工具将使用配置文件中定义的值和客户提供的命令行选项创建一个Greenplum数据库实例。有关配置文件的更多信息，请参见[初始配置文件格式](#)。在运行此工具之前，请确保已经在阵列中所有主机上安装了Greenplum数据库软件。

带有-O *output_configuration_file*选项时，gpinitsystem不会创建一个新数据库，而是把所有的配置都写入一个配置文件。该文件使用QD_PRIMARY_ARRAY和 PRIMARY_ARRAY参数来定义每个成员相关的主机名、端口号、数据目录、Segment前缀、Segment ID和Content ID。集群配置的详细信息可以根据需要修改，以匹配Greenplum数据库备

份中的可用值，活着可以简单的被用来重建相同集群配置。使用`QD_PRIMARY_ARRAY`和`PRIMARY_ARRAY`的配置文件必须被传递给`gpinit system -I input_configuration_file`。详细信息请见[初始配置文件格式](#)。

在Greenplum数据库DBMS中，必须在系统中的所有主机上初始化每个数据库实例（Master和所有的Segment），以便它们可以作为统一的DBMS一起使用。`gpinit system`工具负责初始化Greenplum的Master和每个Segment实例，并作为一个整体配置系统。

在运行`gpinit system`之前，用户必须设置`$GPHOME`环境变量以指向Master上的Greenplum数据库安装位置，并且使用`gpssh-exkeys`在阵列中的所有主机地址之间交换SSH密钥。

这个工具执行以下任务：

- 验证配置文件中参数的正确。
- 确保可以建立到每个主机地址的连接。如果主机地址无法到达，该工具将退出。
- 验证区域设置。
- 显示将要使用的配置并提示用户进行确认。
- 初始化Master实例。
- 初始化后备Master实例（如果指定）。
- 初始化主Segment实例。
- 初始化镜像Segment实例（如果配置）。
- 配置Greenplum数据库系统并检查错误。
- 启动Greenplum数据库系统。

Note: 该工具在系统内部采用SSH连接执行各项操作任务。在大型Greenplum集群、云部署或每台主机部署了大量的segment实例时，可能会遇到超过主机最大授权连接数限制的情况。此时需要考虑更新SSH配置参数`MaxStartups`以提高该限制。更多关于SSH配置的选项，请参考您的Linux分发版的SSH文档。

选项

`-a`

不提示用户进行确认。

`-B parallel_processes`

要并行创建的Segment数。如果未指定，该工具一次最多启动4个并行进程。

`-c cluster_configuration_file`

必需。配置文件的完整路径和文件名称，其中包括所有已经定义的参数，用于配置和初始化新的 Greenplum 数据库系统。有关此文件的说明，请参见[初始配置文件格式](#)[初始化配置文件格式](#)。`gpinit system`必须指定 `-c cluster_configuration_file` 选项或 `-I input_configuration_file` 选项。

-D

设置日志输出等级为 debug。

-h *hostfile_gpinit system*

可选。包含 Segment 主机地址的文件的完整路径和文件名。如果未在命令行中指定，则可以使用 `gpinit system_config` 文件中的 `MACHINE_LIST_FILE` 参数指定主机文件。

-I *input_configuration_file*

配置文件的全路径及文件名，该文件使用 `QD_PRIMARY_ARRAY` 和 `PRIMARY_ARRAY` 参数定义 Greenplum 数据库成员和 Segment 实例。该配置文件通常用 `gpinit system -O output_configuration_file` 生成。`gpinit system` 必须指定 `-c cluster_configuration_file` 选项或 `-I input_configuration_file` 选项。

-n *locale* | --locale= *locale*

设置 Greenplum 数据库使用的默认区域。如果未指定，则 Master 主机的 `LC_ALL`、`LC_COLLATE` 或 `LANG` 环境变量决定区域。如果这些没有设置，则默认的区域是 C (POSIX)。区域标识符由语言标识符、地区标识符和可选的字符集编码组成。例如，`sv_SE` 是瑞典语，`en_US` 是美国英语，`fr_CA` 是加拿大法语。如果不止一个字符集可以用于一个区域，则规范如下所示：`en_US.UTF-8`（区域规范和字符集编码）。在大多数系统中，命令 `locale` 将显示区域环境设置，`locale -a` 将会显示所有可用的区域的列表。

--lc-collate= *locale*

类似于 `--locale`，但是设置用于排序规则（排序数据）的区域。Greenplum 数据库初始化后无法更改排序顺序，因此有必要选择与用户计划用于数据的字符集编码兼容的排序规则区域。C 或 POSIX 有一个特殊的排序规则名称（字节顺序排序而不是字典顺序排序）。C 排序规则可以用于任何字符编码。

--lc-ctype= *locale*

类似于 `--locale`，但设置用于字符分类的语言环境（哪些字符序列是有效的，以及它们如何被解释）。在 Greenplum 数据库初始化之后，这是不能更改的，因此有必要选择一个与用户计划存储在 Greenplum 数据库中的数据兼容的字符分类区域。

--lc-messages= *locale*

类似于 `--locale`，但设置用于 Greenplum 数据库输出消息的语言环境。当前版本的 Greenplum 数据库不支持输出消息的多种区域（所有消息均为英文），所以更改此设置不会有任何效果。

--lc-monetary= *locale*

类似 `--locale`，但是设置用于格式化货币金额的区域。

--lc-numeric= *locale*
类似--locale, 但是设置用于格式化数字的区域。

--lc-time= *locale*
类似--locale, 但设置用于格式化日期和时间的区域。

-l *logfile_directory*
写入日志文件的目录, 默认为~/gpAdminLogs。

-m *number* | --max_connections=*number*
设置Master允许的最大客户端连接数。默认值是250。

-O *output_configuration_file*
当使用-O选项时, gpinit system不会创建一个 新的Greenplum集群, 而是会将提供的集群配置信息写入指定的 When used with the option, does not create a new Greenplum Database cluster but instead writes the supplied cluster configuration information to the specified *output_configuration_file*文件。该文件使用QD_PRIMARY_ARRAY和 PRIMARY_ARRAY参数来定义每个成员相关的主机名、端口号、数据目录、Segment前缀、 Segment ID和Content ID。集群配置的详细信息可以根据需要修改, 以匹配Greenplum数据库备份中的可用值, 活着可以简单的被用来重建相同集群配置。使用QD_PRIMARY_ARRAY、 PRIMARY_ARRAY和MIRROR_ARRAY的配置文件必须被传递给 gpinit system -I *input_configuration_file*以初始化集群。

-p *postgresql_conf_param_file*
可选。包含用户想要为Greenplum数据库设置的postgresql.conf参数 设置的文件名称。这些设置将在初始化单个Master和Segment实例时使用。用户也可以在 初始化后使用gpconfig工具来设置参数。

-q
以静默模式运行。命令行输出不显示在屏幕上, 但仍然写入日志文件。

-b *size* | --shared_buffers= *size*
设置Greenplum服务器实例用于共享内存缓冲区的内存量。用户可以指定以千字节 (KB) 、 兆字节 (MB) 或千兆字节 (GB) 为单位的大小。默认值是125MB。

-s *standby_master_host*
可选。如果用户希望配置备份Master主机, 请使用此选项指定主机名称。Greenplum数据库软件必须已经在该主机上安装和配置。

-P *standby_master_port*
如果使用-s配置了备用Master实例, 可以用该选项定义端口号。默认端口号与Master端口号相同。为了在同一主机上运行备用和主Master, 必须使用该选项 指定一个不同的端口号。Greenplum数据库软件必须已经在备用主机上安装和配置好。

-S *standby_master_datadir* | --standby_dir=*standby_master_datadir*
如果使用-s配置备用Master主机, 可以使用该选项指定数据目录。如果在同一台主机上配置备用Master和主Master实例, 那必须定义

该选项以指定不同的目录。

-e superuser_password | --su_password=superuser_password

使用此选项可指定为Greenplum数据库超级用户帐户（例如gpadmin）设置的密码。如果未指定此选项，则默认密码gparray会分配给超级用户。用户以后可以使用 ALTER ROLE命令更改密码。

推荐的最佳安全实践：

- 不要在生产环境中使用默认密码选项。
- 安装之后立即更改密码。

--mirror_mode={group|spread}

使用该选项指定镜像Segment实例所在的镜像主机。默认采用group模式，会将一台主机上的所有主Segment实例的镜像放到另外一台主机。spread会将一台主机上的所有实例散布在另外的主机上。spread仅在集群主机数量大于每台主机上的实例数量时可用。有关Greenplum数据库镜像策略的详细信息，请见[Segment镜像概述](#)。

-v | --version

显示gpinit system的版本。

-? | --help

显示gpinit system命令行参数，然后退出。

初始配置文件格式

gpinit system需要定义有以下参数的配置文件。示例初始化配置文件可以在

`$GPHOME/docs/cli_help/gpconfigs/gpinit_system_config`中找到。

为了避免Greenplum数据库和其他应用程序之间的端口冲突，Greenplum数据库端口号不应该在操作系统参数`net.ipv4.ip_local_port_range`指定的范围之内。例如，如果`net.ipv4.ip_local_port_range = 10000 65535`，则可以将Greenplum数据库基本端口号设置为以下这些值。

```
PORT_BASE = 6000
MIRROR_PORT_BASE = 7000
```

ARRAY_NAME

必需。用户正在配置的阵列的名称。用户可以使用任何用户喜欢的名字。如果名称包含空格，请将名称放在引号中。

MACHINE_LIST_FILE

可选。可以用来替代-h选项。这指定包含构成Greenplum数据库系统的Segment主机地址名称列表的文件。Master主机被假定为运行该工具的主机，并且不应该被包含在此文件中。如果用户的Segment主机有多个网络接口，则该文件将包含该主机的所有地址。给出该文件的绝对路径。

SEG_PREFIX

必需。这指定了一个前缀，用于命名Master和Segment实例上的数据目录。Greenplum数据库系统中数据目录的命名约定是SEG_PREFIX*number*，其中*number*对Segment实例从0开始（Master始终为-1）。因此，如果用户选择前缀gpseg，则用户的Master实例数据目录将被命名为gpseg-1，并且Segment实例将被命名为gpseg0、gpseg1、gpseg2、gpseg3等等。

PORt_BASE

必需。这指定计算主Segment端口用到的基础数字。主机上的第一个主Segment端口被设置为PORT_BASE，然后对该主机上的每个额外主Segment都加一，有效值为1-65535。

DATA_DIRECTORY

必需。这指定工具将创建主Segment数据目录的数据存储位置。列表中的位置数量决定了每台物理主机将创建的主Segment的数量（如果主机文件中列出了主机的多个地址，Segment的数量将被均匀分布在指定的接口地址间）。如果用户希望在同一位置创建数据目录，则可以多次列出相同的数据存储区域。运行gpinit system的用户（例如，gpadmin用户）必须具有写入这些目录的权限。例如，这将为每台主机创建六个主Segment：

```
declare -a DATA_DIRECTORY=( /data1/primary /data1/primary
                           /data1/primary /data2/primary /data2/primary
                           /data2/primary )
```

MASTER_HOSTNAME

必需。Master实例的主机名。这个主机名必须与机器上配置的主机名完全匹配（运行hostname命令以确定正确的主机名）。

MASTER_DIRECTORY

必需。这指定在Master主机上创建数据目录的位置。用户必须确保运行gpinit system的用户（例如gpadmin用户）有权写入此目录。

MASTER_PORT

必需。Master实例的端口号。这是访问Greenplum数据库系统时用户和客户端连接将使用的端口号。

TRUSTED_SHELL

必需。gpinit system工具用来在远程主机上执行命令的shell。允许的值为ssh。用户必须在运行gpinit system工具之前设置用户的可信主机环境（用户可以使用gpssh-exkeys来做这件事）。

CHECK_POINT_SEGMENTS

必需。日志文件段（每个段通常为 兆字节）中自动预写式日志（WAL）检查点之间的最大距离。这将在Greenplum数据库系统的每个Segment实例的`postgresql.conf`文件中设置`checkpoint_segments`参数。

ENCODING

必需。要使用的字符集编码。该字符集必须与所使用的--`locale`设置兼容，尤其是--`lc_collate`和--`lc_ctype`。Greenplum数据库 支持和PostgreSQL相同的字符集。

DATABASE_NAME

可选。系统初始化之后要创建的Greenplum数据库的名称。用户可以随后使用`CREATE DATABASE`命令或`createdb`工具创建数据库。

MIRROR_PORT_BASE

可选。这指定计算镜像Segment端口号用到的基数。主机上的第一个镜像Segment端口设置为`MIRROR_PORT_BASE`，然后对该主机上每个额外的镜像Segment加一。有效值范围从1到65535，不能与`PORT_BASE`计算的端口冲突。

MIRROR_DATA_DIRECTORY

可选。指定工具将创建镜像Segment数据目录的数据存储位置。必须为镜像Segment 实例声明与主Segment实例相同数量的数据目录（请参阅`DATA_DIRECTORY`参数）。运行`gpinit system`的用户（例如，`gpadmin`用户） 必须具有写入这些目录的权限。例如：

```
declare -a MIRROR_DATA_DIRECTORY=( /data1/mirror
/data1/mirror /data1/mirror /data2/mirror /data2/mirror
/data2/mirror)
```

QD_PRIMARY_ARRAY, PRIMARY_ARRAY, MIRROR_ARRAY

这些参数只能通过配置文件提供给`gpinit system -I input_configuration_file`。

`QD_PRIMARY_ARRAY`、`PRIMARY_ARRAY`和`MIRROR_ARRAY`定义Greenplum数据库Master主机、主实例和镜像实例，格式如下：

```
host~port~data_directory/seg_prefix<segment_id>~dbid~conte
```

Greenplum数据库Master会一直使用-1作为Segment ID和Content ID。例如：

```
QD_PRIMARY_ARRAY=127.0.0.1~5432~/gpmaster/gpsne-1~1~-1~-0
declare -a PRIMARY_ARRAY=( 
127.0.0.1~40000~/gpdata1/gpsne0~2~0
127.0.0.1~40001~/gpdata2/gpsne1~3~1
)
declare -a MIRROR_ARRAY=( 
127.0.0.1~50000~/gpmirror1/gpsne0~4~0
127.0.0.1~50001~/gpmirror2/gpsne1~5~1
)
```

客户可以使用`gpinit system -O output_configuration_file`来生成`OD_PRIMARY_ARRAY`、`PRIMARY_ARRAY`、`MIRROR_ARRAY`参数，包含主机、数据目录、Segment前缀和端口号信息。出于恢复的目的，您可以编辑Segment和Content ID的值以匹配现存Greenplum数据库备份的值。

HEAP_CHECKSUM

可选。该参数指定堆表的数据是否启用checksum。当启用时，所有数据库上的堆存储进行checksum，Greenplum数据库可以预防I/O系统损坏导致数据损坏。该选项只有在系统初始化时进行设置并且以后不能改变。

`HEAP_CHECKSUM`选项默认启用，强烈不建议将该选项设置为禁用。如果禁用该选项，存储的数据损坏不会被检测到，数据恢复会相当困难。

要检查是否Greenplum数据库启用了堆表checksum，可以通过`gpconfig`管理工具查询`data_checksums`参数：

```
$ gpconfig -s data_checksums
```

HBA_HOSTNAMES

可选。该参数控制`gpinit system`工具在`pg_hba.conf`中开放访问IP地址或者主机名的信息。默认值为0，该工具会使用IP地址更新`hba`文件。当初始化Greenplum数据库系统时，指定`HBA_HOSTNAMES=1`来让工具使用主机名更新`pg_hba.conf`文件。

更多有关Greenplum数据库解决`pg_hba.conf`文件中主机名的信息，请见[配置客户端认证](#)。

示例

通过提供配置文件和Segment主机地址文件来初始化Greenplum数据库阵列，并设置一个散布镜像（`--mirror-mode=spread`）配置：

```
$ gpinit system -c gpinit system_config -h hostfile_gpinit system --mirror-mode=spread
```

初始化Greenplum数据库阵列并设置超级用户远程口令：

```
$ gpinit system -c gpinit system_config -h hostfile_gpinit system --su-password=mypassword
```

初始化带有可选后备Master主机的Greenplum数据库阵列：

```
$ gpinit system -c gpinit system_config -h  
hostfile_gpinit system -s host09
```

代替初始化一个Greenplum数据库集群，将提供的配置文件信息写入到输出文件。该输出文件采用QD_PRIMARY_ARRAY和PRIMARY_ARRAY参数定义Master和Segment主机：

```
$ gpinit system -c gpinit system_config -h  
hostfile_gpinit system --mirror-mode=spread -O  
cluster_init.config
```

采用输入带有QD_PRIMARY_ARRAY和PRIMARY_ARRAY参数的配置文件（该文件定义Greenplum数据库集群）初始化Greenplum数据库：

```
$ gpinit system -I cluster_init.config
```

另见

[gpssh-exkeys](#)、[gpdelete system](#)

Greenplum数据库® 6.0文档

[工具指南](#)

[管理工具参考](#)

[analyzedb](#)[gpactivatestandby](#)[gpaddmirrors](#)[gpbackup](#)[gpcheck](#)[gpcheckcat](#)[gpcheckperf](#)[gpconfig](#)[gpdeletesystem](#)[gpexpand](#)[gpfdist](#)[gpinitstandby](#)[gpinitsystem](#)[gupload](#)[gplogfilter](#)[gpmapreduce](#)[gpmovemirrors](#)[gpperfmon_install](#)[gppkg](#)

在Greenplum数据库日志文件中搜索指定的项。

概要

```
gplogfilter [timestamp_options] [pattern_options]
    [output_options] [input_options] [input_file]
```

```
gplogfilter --help
```

```
gplogfilter --version
```

描述

gplogfilter工具可以被用来在一个Greenplum数据库日志文件中搜索匹配指定条件的项。如果没有提供输入文件，那么**gplogfilter**将使用环境变量\$MASTER_DATA_DIRECTORY 在标准日志位置定位Greenplum的Master日志文件。要从标准输入读取，可用一个破折号（-）作为输入文件名。输入文件可以被使用gzip压缩。在输入文件中，日志项由其YYYY-MM-DD [hh:mm[:ss]] 格式的时间戳标识。

如果通过[gpssh](#)工具运行 **gplogfilter**，用户还可以用它来一次搜索所有的Segment日志文件。例如，要显示每个Segment 日志文件中的最后三行：

```
gpssh -f seg_host_file
      => source /usr/local/greenplum-
db/greenplum_path.sh
      => gplogfilter -n 3 /gpdata/*/pg_log/gpdb*.csv
```

gplogfilter的输出默认被发送到标准输出。可以使用-o选项把输出发送到一个文件或者一个目录。如果用户提供了一个以.gz结束的输出文件名，该输出文件默认将被以最大压缩率压缩。如果输出目标是一个目录，输出文件将被命名为输入文件的名字。



选项

gprecoverseg

时间戳选项

-b *datetime* | --begin= *datetime*

以格式YYYY-MM-DD [hh:mm[:ss]]指定要搜索的日志项的开始日期和时间。

如果指定一个时间，日期和时间必须被封闭在单引号或者双引号中。这个例子将日期和时间封闭在单引号中：

```
gplogfilter -b '2013-05-23 14:33'
```

-e *datetime* | --end= *datetime*

指定一个YYYY-MM-DD [hh:mm[:ss]]格式的结束日期和时间来停止搜索日志信息。

-d *time* | --duration= *time*

以[hh] [:mm[:ss]]格式指定要搜索的日志项所在的时间长度。如果指定这个选项但没有指定-b或-e选项，将使用当前时间作为基础。

模式匹配选项

-c i [gnore] | r [espect] | --case=i [gnore] | r [espect]

除非前面放上--case=ignore选项，字母表符号的匹配默认是大小写敏感的。

-C '*string*' | --columns=' *string*'

从日志文件中选择特定的列。将想要的列指定为由逗号定界的列编号串，列编号从1开始，从左到右依次是1、2、3等等。关于日志文件格式的详情以及可用列的列表及其相关编号请见Greenplum数据库管理员指南中的“查看数据库服务器日志文件”部分。

-f '*string*' | --find=' *string*'

查找包含指定字符串的日志项。

-F '*string*' | --nofind=' *string*'

拒绝包含指定字符串的日志项。

-m *regex* | --match= *regex*

查找匹配指定的Python正则表达式的日志项。Python正则表达式语法请参考 <https://docs.python.org/library/re.html>

-M *regex* | --nomatch= *regex*

拒绝匹配指定Python正则表达式的日志项。Python正则表达式语法请参考 <https://docs.python.org/library/re.html>

-t | --trouble

只查找在第一行有ERROR:、FATAL:或者PANIC:的日志项。

输出选项

-n *integer* | --tail= *integer*

限制输出为找到的符合条件日志项中的最后integer项。

-s offset [limit] | --slice=offset [limit]

从符合条件的日志项列表中，从offset项处开始返回limit个项，其中为零（0）的offset指示结果集中的第一项并且任何不超过零的offset表示从结果集的末尾开始数。

-o output_file | --out=output_file

将输出写到指定的文件或者目录位置而不是STDOUT。

-z 0-9 | --zip=0-9

使用gzip把输出文件压缩到指定的压缩级别，其中0是不压缩而9是最大压缩。如果用户提供一个以.gz结束的输出文件名，该输出文件默认将被使用最大压缩级别压缩。

-a | --append

如果输出文件已经存在，会追加到该文件而不是覆盖它。

输入选项

input_file

要在其中搜索的输入日志文件的名称。如果没有提供输入文件，gplogfilter将使用环境变量\$MASTER_DATA_DIRECTORY来定位Greenplum数据库的Master日志文件。要从标准输入读取，可使用一个破折号（-）作为输入文件名。

-u | --unzip

使用gunzip解压输入文件。如果输入文件名以.gz结束，它默认将被解压。

--help

显示在线帮助。

--version

显示这个工具的版本。

示例

显示Master日志文件中的最后三个错误消息：

```
gplogfilter -t -n 3
```

显示Master日志文件中时间戳位于最后10分钟内的所有日志消息：

```
gplogfilter -d :10
```

显示Master日志文件中包含字符串|con6 cmd11|的日志消息：

```
gplogfilter -f '|con6 cmd11|'
```

使用[gpssh](#)在Segment主机上运行gplogfilter并且在Segment日志文件中搜索含有字符串 con6的日志消息，把输出保存到一个文件。

```
gpssh -f seg_hosts_file -e 'source  
/usr/local/greenplum-db/greenplum_path.sh ;  
gplogfilter -f  
con6 /gpdata/*/pg_log/gpdb*.csv' > seglog.out
```

另见

[gpssh](#)、[gpscp](#)

[工具指南](#)

[管理工具参考](#)

[analyzedb](#)

[gpactivatestandby](#)

[gpaddmirrors](#)

[gpbackup](#)

[gpcheck](#)

[gpcheckcat](#)

[gpcheckperf](#)

[gpconfig](#)

[gpdeletesystem](#)

[gpexpand](#)

[gpfdist](#)

[gpinitstandby](#)

[gpinitsystem](#)

[gupload](#)

[gplogfilter](#)

[gmapreduce](#)

[gpmovemirrors](#)

[gpperfmon_install](#)

[gppkg](#)

按照YAML规范文档中的定义运行Greenplum的MapReduce作业。

概要

```
gmapreduce -f yaml_file [dbname [username]]
           [-k name=value | --key name=value]
           [-h hostname | --host hostname] [-p port| --port
           port]
           [-U username | --username username] [-W] [-v]

           gmapreduce -x | --explain

           gmapreduce -x | --explain-analyze

           gmapreduce -v | --version

           gmapreduce -h | --help
```

先决条件

在运行此程序之前，需要执行以下操作：

- 用户必须在YAML文件中定义用户的MapReduce作业。有关Greenplum MapReduce规范的信息，请参阅[Greenplum数据库参考指南](#)。
- 用户必须是Greenplum数据库超级用户才能运行使用不可信Perl或Python 编写的MapReduce作业。
- 用户必须是Greenplum数据库超级用户才能运行带有EXEC 和FILE输入的MapReduce作业。
- 用户必须是Greenplum数据库超级用户才能运行带有GPFDIST 输入的MapReduce作业，除非用户具有适当的权限。有关更多信息，请参阅[Greenplum数据库参考指南](#)。

描述

[MapReduce](#) 是由Google开发的用于在商用服务器阵列上处理和生成大型数据集的编程模型。Greenplum MapReduce允许熟



gprecoverseg

悉MapReduce范式的编程人员编写map和reduce函数，并将它们提交给Greenplum数据库并行引擎处理。

为了使Greenplum能够处理MapReduce函数，需要在YAML文档中定义函数，然后将其传递到Greenplum MapReduce程序gpmapreduce，以供Greenplum数据库并行引擎执行。Greenplum系统负责分布输入数据、在一组机器上执行程序、处理机器故障以及管理所需的机器间通信。

选项

-f *yaml_file*

必须。包含Greenplum MapReduce作业定义的YAML文件。请参阅 *Greenplum*数据库参考指南。

-? | --help

显示帮助，然后退出。

-V | --version

显示版本信息，然后退出。

-v | --verbose

显示详细输出。

-x | --explain

不运行MapReduce作业，而是生成解释计划。

-X | --explain-analyze

运行MapReduce作业并生成解释 - 分析计划。

-k | --keyname= *value*

设置一个YAML变量。需要一个数值。如果未指定变量名称，则默认为“key”。

连接选项

-h *host* | --host *host*

指定运行Greenplum的Master数据库服务器的机器的主机名。如果未指定，则从环境变量 PGHOST读取或默认为localhost。

-p *port* | --port *port*

指定Greenplum的Master数据库服务器正在侦听连接的TCP端口。如果未指定，则从环境变量PGPORT读取或默认为5432。

-U *username* | --username *username*

T要连接的数据库角色名称。如果未指定，则从环境变量PGUSER 读取或默认为当前系统用户名。

-W | --password

强制口令提示。

示例

运行my_yaml.txt中定义的MapReduce作业，并连接到数据库mydatabase：

```
gpmapreduce -f my_yaml.txt mydatabase
```

另见

*Greenplum*数据库参考指南中的Greenplum MapReduce规范。

Greenplum数据库® 6.0文档

安装Greenplum Command Center使用的gpperfmon数据库，并可选择启用数据收集代理。

□ 工具指南

□ 管理工具参考

analyzedb

```
gpperfmon_install --port gpdb_port
    [--enable --password gpmon_password [ --pgpass
    path_to_file]]
    [--verbose]
```

gpactivatestandby

```
gpperfmon_install --help | -h | -?
```

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gpload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

aprecoversea

概要

```
gpperfmon_install --port gpdb_port
    [--enable --password gpmon_password [ --pgpass
    path_to_file]]
    [--verbose]

gpperfmon_install --help | -h | -?
```

描述

gpperfmon_install工具自动执行启用数据收集代理程序所需的步骤。用户必须是Greenplum数据库系统用户（gpadmin）才能运行此工具。--port选项是必须的。使用--enable选项时，还需要--password选项。使用--port选项来提供Greenplum数据库的Master实例的端口。如果使用--enable选项，必须在工具完成后重新启动Greenplum数据库。

在没有--enable选项时，此工具只创建gpperfmon数据库（用于存储由数据收集代理收集的系统度量的数据库）。在使用--enable选项运行时，该工具还会运行启用性能监视器数据收集代理所需的以下附加任务：

1. 在Greenplum数据库中创建gpmon超级用户角色。数据收集代理需要连接到数据库并写入数据。gpmon超级用户角色默认使用MD5加密口令认证。使用--password选项来设置gpmon超级用户的口令。
2. 更新\$MASTER_DATA_DIRECTORY/pg_hba.conf文件。该工具将这些行添加到基于主机的认证文件（pg_hba.conf）中：

local	gpperfmon	gpmon	md5	
host	all	gpmon	127.0.0.1/28	md5
host	all	gpmon	::1/128	md5

第二和第三行的host项，让gpmon能访问所有Greenplum数据库

Note: 可能有必要在运行gpperfmon_install工具以限制gpmon角色访问数据库或更改身份验证方法之后再编辑pg_hba.conf文件中的行。运行gpstop -u重新加载Greenplum数据库的pg_hba.conf文件。

- 要将gpmon的访问限制为仅访问gpperfmon数据库，编辑pg_hba.conf文件中host项。对于gpmon用户，将第二个字段从all更改为gpperfmon：

local	gpperfmon	gpmon	md5
host	gpperfmon	gpmon	127.0.0.1/28

md5				
	host	gpperfmon	gpmon	::1/128
md5				

- gpperfmon_install工具采用默认的MD5认证方法。Greenplum数据库可以选择性地配置为使用SHA-256哈希算法来计算保存在系统目录中的口令哈希值。这与MD5认证方法不兼容，MD5认证方法在系统目录中需要MD5哈希或明文口令。因此，如果在数据库中启用了SHA-256哈希算法，则必须在运行gpperfmon_install工具后编辑pg_hba.conf文件。对于host项，将gpmon角色的身份认证从md5改为password：

local	gpperfmon	gpmon	md5
	host	all	gpmon 127.0.0.1/28
password			
	host	all	gpmon ::1/128
password			

password认证方法提交用户的明文口令进行验证，不应在不可信的网络上使用。请参阅Greenplum数据库管理员指南中的“保护Greenplum数据库中的密码”。

- 更新口令文件(.pgpass)。为了允许数据收集代理作为gpmon角色进行连接而无需口令提示，用户必须拥有一个包含gpmon 用户项的口令文件。该工具将以下项添加到用户的口令文件（如果该文件不存在，该工具将创建它）：

```
*:5432:gpperfmon:gpmon:gpmon_password
```

如果用户的口令文件不在默认位置(~/.pgpass)，使用--pgpass选项指定文件位置。

- 设置Greenplum Command Center的服务器配置参数。数据收集代理必须启用以下参数才能开始收集数据。该工具在Greenplum数据库的postgresql.conf配置文件中设置以下参数：

- gp_enable_gpperfmon=on (在集群中所有的postgresql.conf文件中)
- gpperfmon_port=8888 (在集群中所有的postgresql.conf文件中)
- gp_external_enable_exec=on (在Master的postgresql.conf文件中)

数据收集代理可以通过在gpperfmon.conf配置文件中设置参数进行配置。参阅[数据收集代理配置](#)获取更多详细信息。

选项

--enable

除了创建gpperfmon数据库外，还执行启用数据收集代理所需的附加步骤。指定--enable时，工具还将创建并配置gpmon超级用户帐户，并在postgresql.conf文件中设置Command Center服务器配置参数。

--password gpmon_password

如果必须指定了--enable，设置gpmon超级用户的口令。如果未指定--

`--enable`则不允许使用。

`--port gpdb_port`
必需。指定Greenplum数据库Master的连接端口。

`--pgpass path_to_file`
在`--enable`选项被指定时可选。如果口令文件不在`~/.pgpass`的默认位置，则这个选项指定口令文件的位置。

`--verbose`
将日志记录级别设置为详细。

`--help | -h | -?`
显示在线帮助。

数据收集代理配置

`$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf`文件存储数据收集代理的配置参数。要使这些选项的配置更改生效，必须保存`gpperfmon.conf`然后重新启动Greenplum数据库服务器 (`gpstop -r`)。

`gpperfmon.conf`文件包含以下配置参数。

参数	描述
<code>log_location</code>	指定 <code>gpperfmon</code> 日志文件的目录位置。默认是 <code>\$MASTER_DATA_DIRECTORY/gpperfmon/logs</code> 。
<code>min_query_time</code>	<p>指定统计信息收集的最短查询运行时间（秒）。所有运行时间超过此值的查询都将记录在<code>queries_history</code>表中。对于运行时间较短的查询，不收集历史数据。默认为20秒。</p> <p>如果用户知道要为所有查询收集数据，则可以将此参数设置为较低值。但是，如果将最小查询运行时间设置为零，即使对于Greenplum Command Center运行的大量查询也会收集数据，从而创建大量可能无用的数据。</p>
<code>max_log_size</code>	<p>这个参数不包含在<code>gpperfmon.conf</code>中，但是它可能被添加到这个文件中。</p> <p>为防止日志文件增长过大，可以将<code>max_log_size</code>参数添加到<code>gpperfmon.conf</code>中。该参数的值以字节为单位。例如：</p> <pre>max_log_size = 10485760</pre> <p>使用此设置，日志文件将在系统转到新的日志文件之前增长到10MB。</p>
<code>partition_age</code>	<code>gperfmon</code> 统计数据将被保留的月数。它的默认值是0，

	这意味着我们不会丢失任何数据。
quantum	<p>指定所有Segment上的数据收集代理更新之间的时间（以秒为单位）。有效值为10、15、20、30和60。缺省值为15秒。</p> <p>如果用户偏好较低粒度的性能视图，或者想要收集和分析系统度量的最小量，请选择较高的quantum。要更频繁地收集数据，请选择一个较低的值。</p>
harvest_interval	数据捕获时间段，以秒为单位。一次数据捕获会将gpperfmon外部表（_tail）的近期数据移动到对应的历史文件。默认为120。最小值为30。
ignore_qexec_packet	（弃用）。设置为true时，数据收集代理不会收集gpperfmon数据库的queries_*表中的性能数据：rows_out、cpu_elapsed、cpu_currpct、skew_cpu和skew_rows 缺省设置true可以减少gpmon进程所消耗的内存量。如果用户需要这些额外的性能数据，请将此参数设置为false。
smdw_aliases	<p>此参数允许用户为后备Master指定额外的主机名。例如，如果后备Master有两个NIC，则可以输入：</p> <pre>smdw_aliases=smdw-1,smdw-2</pre> <p>如果Greenplum Command Center失去与后备Master的连接，则这个可选的容错参数非常有用。它不会持续重试连接到主机smdw，而是尝试连接到基于NIC的别名smdw-1或smdw-2。这确保了Command Center控制台可以持续调查和监控后备Master。</p>

注解

gpperfmon数据库和Greenplum Command Center需要gpmon角色。在创建gpperfmon数据库和gpmon角色后，用户可以更改 gpmon角色的口令并更新Greenplum Command Center信息用来连接到gpperfmon 数据库：

1. 以超级用户身份登录到Greenplum数据库，并使用ALTER ROLE命令更改 gpmon的口令。

```
# ALTER ROLE gpmon WITH PASSWORD 'new_password' ;
```

2. 更新Greenplum Command Center使用的.pgpass文件中的口令。默认文件位置是gpadmin 的主目录（~/ .pgpass）。.pgpass文件包含一行gpmon口令。

```
*:5432:gpperfmon:gpmon:new_password
```

3. 使用Command Center的gpcmdr工具重新启动Greenplum Command Center。

```
$ gpcmdr --restart
```

gpperfmon监控系统在启动后需要一些初始化。监控信息在几分钟后出现，而不是在安装和启动gpperfmon系统之后立即出现。

示例

只创建gpperfmon数据库：

```
$ su - gpadmin
$ gpperfmon_install --port 5432
```

创建gpperfmon数据库，创建gpmon超级用户，并启用数据收集代理：

```
$ su - gpadmin
$ gpperfmon_install --enable --password changeme --port 5432
$ gpstop -r
```

另见

[gpstop](#)

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

在整个集群中安装.gppkg格式的Greenplum数据库扩展（例如PL/Java、PL/R和MADlib）及其依赖项。

概要

```
gppkg [-i package | -u package | -r name-version | -c]
          [-d master_data_directory] [-a] [-v]

gppkg --migrate GPHOME_old GPHOME_new [-a]
          [-v]

gppkg [-q | --query] query_option

gppkg -? | --help | -h

gppkg --version
```

描述

Greenplum软件包管理器(gppkg)工具在集群中的所有主机上安装Greenplum数据库扩展以及任何依赖项。在系统扩展和Segment恢复的情况下，它也将自动在新主机上安装扩展。

Note: 在Greenplum数据库重要升级之后，用户必须再次下载并安装所有gppkg扩展。

选项

-a (不要提示)

不要提示用户确认。

-c | --clean

协调集群的包状态以匹配Master主机的状态。在失败或部分的安装/卸载后运行此选项可确保软件包安装状态在集群间保持一致。

-d master_data_directory

如果未指定，则使用为\$MASTER_DATA_DIRECTORY设置的值。



`gprecoverseg``-i package | --install= package`

安装给定的包。这包括任何前/后安装步骤以及任何依赖关系的安装。

`--migrate GPHOME_old GPHOME_new`

从单独的\$GPHOME迁移软件包。将包从一个版本的Greenplum数据库运输到另一个版本。

例如: `gppkg --migrate /usr/local/greenplum-db-<old-version> /usr/local/greenplum-db-<new-version>`

当迁移包时，要求满足以下条件。

- 至少目标数据库集群的Master实例要启动（实例安装在*GPHOME_new*）。在运行gppkg命令之前执行gpstart -m命令启动 Master实例。
- 运行*GPHOME_new*安装目录下的gppkg工具。

`-q | --query query_option`

提供有关已安装软件包的*query_option*指定的信息。一次只能指定一个*query_option*。下表列出了*query_option*的可能值。*<package_file>*是一个包的名字。

Table 1. gppkg的查询选项

<i>query_option</i>	返回
<i><package_file></i>	是否安装了指定的软件包。
<code>--info <i><package_file></i></code>	关于指定软件包的名称、版本和其他信息。
<code>--list <i><package_file></i></code>	指定包的文件内容。
<code>--all</code>	列出所有已安装的包。

`-r name-version | --remove=name-version`

删除指定的包。

`-u package | --update= package`

更新给定的包。

Warning: 更新程序包的过程包括删除与程序包相关的系统对象的所有先前版本。例如，以前版本的共享库被删除。更新过程之后，如果函数引用已被删除的包文件，则数据库函数在调用时将失败。

`--version (显示工具版本)`

显示此工具的版本。

`-v | --verbose`

将日志记录级别设置为详细。

`-? | -h | --help`

显示在线帮助。

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

恢复已标记为down的主Segment实例或镜像Segment实例（如果启用了镜像）。

概要

```
gprecoverseg [-p new_recover_host[,...]] | -i
recover_config_file] [-d master_data_directory]
[-B parallel_processes] [-F [-s]] [-a] [-q]
[--no-progress] [-l logfile_directory]

gprecoverseg -r

gprecoverseg -o output_recover_config_file
[-p new_recover_host[,...]]

gprecoverseg -? | --help

gprecoverseg --version
```

描述

在启用了镜像的系统中，**gprecoverseg**工具会重新激活故障的Segment实例，并识别需要重新同步的已更改的数据文件。一旦**gprecoverseg**完成这个过程，系统进入*resynchronizing*模式，直到被恢复的Segment更新为最新为止。在重新同步期间系统处于在线状态并且能被正常操作。

在增量恢复期间（未指定-F选项），如果**gprecoverseg**在启用了镜像的系统中检测到禁用镜像的Segment实例，则工具将报告该Segment禁用了镜像，不会尝试恢复该Segment实例，并继续恢复过程。

Segment实例可能由于多种原因故障，如主机故障、网络故障或磁盘故障。当一个Segment实例故障时，其状态在Greenplum数据库系统目录中被标记为*down*并且在*change tracking*模式下激活其镜像。为了使发生故障的Segment实例重新运行，首先必须纠正使其故障的问题，然后使用**gprecoverseg**在Greenplum数据库中恢复Segment实例。

使用**gprecoverseg**进行Segment恢复需要有一个活动镜像来从其中恢复。对于没有启用镜像的系统，或者在发生双重故障的情况下

gprecoverseg

(主Segment和镜像Segment同时故障) , 必须采取 手动步骤恢复出现故障的Segment实例, 然后执行系统重新启动让Segment重新联机。例如, 该命令重新启动系统。

```
gpstop -r
```

缺省情况下, 将恢复出现故障的Segment, 这意味着系统将该Segment重新联机到与最初配置的主机和 数据目录位置相同的位置。在这种情况下, 请使用以下格式的恢复配置文件 (使用-i) 。

```
<failed_host_address>:<port>:<data_directory>
```

在某些情况下, 这可能是不可能的 (例如, 如果主机物理损坏, 无法恢复) 。gprecoverseg 允许用户将失败的Segment恢复成全新的主机 (使用-p) , 在剩余的活动Segment主机上的 备用数据目录位置 (使用-s) 或通过下面的格式提供恢复配置文件 (使用-i) 。

SPACE关键字表示所需空间的位置。不要添加额外的空间。

```
<failed_host_address>:<port>:<data_directory>SPACE
<recovery_host_address>:<port>:<data_directory>
```

有关恢复配置文件的详细信息和示例, 请参阅下面的-i选项。

gp_segment_configuration系统目录表可以帮助用户确定当前的Segment配置, 以便用户可以规划镜像恢复配置。例如, 运行以下查询:

```
=# SELECT dbid, content, address, port, datadir
   FROM gp_segment_configuration
 ORDER BY dbid;
```

新恢复的Segment主机必须预先安装Greenplum数据库软件, 并且配置与现有Segment主机完全相同。所有 当前配置的Segment主机上必须存在备用数据目录位置, 并且有足够的磁盘空间来容纳故障的Segment。

恢复过程会在Greenplum数据库系统目录中再次标记该Segment, 然后启动重新同步过程, 以使Segment的 事务状态处于最新状态。在重新同步期间系统在线并且可用。要检查重新同步进程运行的状态:

```
gpstate -m
```

选项

-a (不提示)

不要提示用户确认。

-B *parallel_processes*

并行恢复的Segment数。如果未指定，则实用程序将启动最多16个并行进程，具体取决于需要恢复多少个Segment实例。

-d *master_data_directory*

可选。Master主机的数据目录。如果未指定，则使用为\$MASTER_DATA_DIRECTORY设置的值。

-F (完全恢复)

可选。执行活动Segment实例的完整副本以恢复出现故障的Segment。默认情况下，仅复制Segment关闭时发生的增量更改。

Warning: 全量恢复会在从活动（当前主）Segment实例复制数据目录到故障Segment之前将故障Segment的数据目录清空。在执行全量恢复前，确保Segment故障不会引起数据损坏，并且任何Segment磁盘故障问题均被修复。

全量恢复会持续很长时间，所以gprecoverseg会显示每一个Segment的运行处理过程。每个Segment的处理过程每秒更新一次，采用ANSI逃逸符更新每一个Segment行，如果您将gprecoverseg工具执行日志输出到文件或您的终端不支持ANSI逃逸符，可以采用-s选项停用ANSI逃逸符。该操作会使每个Segment每秒输出一个新行。带有--no-progress时会禁用处理报告。

-i *recover_config_file*

指定文件的名称以及有关故障Segment要恢复的详细信息。文件中的每一行都是以下格式。**SPACE**关键字表示所需空间的位置。不要添加额外的空间。

```
<failed_host_address>:<port>:<data_directory>SPACE
<recovery_host_address>:<port>:
<data_directory>
```

注释

以#开始的行被视为注释并被忽略。

要恢复的Segments

第一行之后的每一行指定要恢复的Segment。这一行可以有两种格式之一。在就地恢复的情况下，在该行中输入一组冒号分隔的字段。例如：

```
failedAddress:failedPort:failedDataDirectory
```

要恢复到新位置，请在行中输入由空格分隔的两组字
段。**SPACE** 表示不要添加额外的空间。

```
failedAddress:failedPort:failedDataDirectory SPACE newAddr
```

```
newPort:newDataDirectory
```

示例

单个镜像的就地恢复

```
sdw1-1:50001:/data1/mirror/gpseg16
```

将单个镜像恢复到新主机

```
sdw1-1:50001:/data1/mirror/gpseg16 SPACE sdw4-  
1:50001:/data1/recover1/gpseg16
```

获取示例文件

用户可以使用-*o*选项输出一个恢复配置文件的样例以便从它开
始。

-l *logfile_directory*

写入日志文件的目录。默认为~/gpAdminLogs。

-o *output_recover_config_file*

指定文件名称和位置以输出示例恢复配置文件。输出文件以-
*i*选项所需的格式列出当前故障的Segment及其默认恢复位置。
与-p选项一起使用可输出用于在不同主机上恢复的示例文件。

如果需要，可以编辑此文件以提供备用恢复位置。

-p *new_recover_host[,...]*

在当前配置的Greenplum数据库阵列之外指定一个备用主机，用
于恢复故障的Segment。在多个故障的Segment 主机的情况下，
用户可以指定一个逗号分隔的列表。备用主机必须安装和配
置Greenplum数据库软件，并具有与当前 网段主机相同的硬件和
操作系统配置（相同的OS版本、语言环境、gpadmin用户帐
户、创建的数据目录位置、交换的ssh密钥、网络接口数量、网
络接口命名约定等）。

-q (无屏幕输出)

以静默模式运行。命令输出不显示在屏幕上，但仍然写入日志文
件。

-r (重新平衡Segment)

在Segment恢复之后，Segment实例可能并未回到系统初始化时为它给定的优先角色。这可能会让系统陷入一种潜在的非平衡状态，因为某些Segment主机拥有的活动Segment数量可能超过最高系统性能时的数量。这一选项通过将主Segment和镜像Segment返回到其优先角色来重新平衡主Segment和镜像Segment。在运行gprecoverseg -r之前，所有的Segment都必须有效且同步。如果有任何正在进行的查询，它们将被取消并回滚。

-s

展示pg_basebackup顺序处理过程而不是当前情况。此选项在需要写入文件或tty不支持转义字符时有用。默认是展示当前处理过程。

--no-progress

禁用产生于pg_basebackup的处理过程报告。默认配置为显示基础备份的处理过程。

-v (详细模式)

将日志记录输出设置为verbose。

--version (版本)

显示此工具的版本。

-? (帮助)

显示在线帮助。

示例

恢复所有故障的Segment实例：

```
$ gprecoverseg
```

恢复后，重新平衡用户的Greenplum数据库系统，将所有Segment重置为其首选角色。首先检查所有Segment已启动并同步。

```
$ gpstate -m
$ gprecoverseg -r
```

将任何故障的Segment实例恢复到新配置的空闲Segment主机：

```
$ gprecoverseg -i recover_config_file
```

输出默认恢复配置文件：

```
$ gprecoverseg -o /home/gpadmin/recover_config_file
```

另见

[gpstart](#)、[gpstop](#)

[工具指南](#)

[管理工具参考](#)

[analyzedb](#)

[gpactivatestandby](#)

[gpaddmirrors](#)

[gpbackup](#)

[gpcheck](#)

[gpcheckcat](#)

[gpcheckperf](#)

[gpconfig](#)

[gpdeletesystem](#)

[gpexpand](#)

[gpfdist](#)

[gpinitstandby](#)

[gpinitsystem](#)

[gupload](#)

[gplogfilter](#)

[gpmapreduce](#)

[gpmovemirrors](#)

[gpperfmon_install](#)

[gppkg](#)

重新装载Greenplum数据库表数据，根据指定的列对数据进行排序。

概要

```
gpreload -d database [-p port] {-t | --table-file}
path_to_file [-a]
```

```
gpreload -h
```

```
gpreload --version
```

描述

gpreload工具重新装载列数据被排序过的表数据。对于使用表存储选项 `appendoptimized=TRUE`并启用压缩创建的表，使用排序数据重新装载数据可以改善表压缩。用户在一个文本文件中指定要重新装载的表要排序的表列列表。

当列中的数据的不同值与行总数相比来说相对较小时，压缩可以通过对数据进行排序来改进。

对于正在重新装载的表，要排序的列顺序可能会影响压缩。具有最少不同值的列应该首先列出。例如，列出州和城市一般会获得比列出城市和州更好的压缩。

```
public.cust_table: state, city
public.cust_table: city, state
```

有关**gpreload**使用的文件格式的信息，请参阅--table-file选项。

注解

为了提高重载性能，在重载数据之前，应该删除正在重载表上的索引。



gprecoverseg

重新装载表数据后运行ANALYZE命令可能会因为重新装载 数据的数据分布发生更改而查询性能。

对所有的表，该工具都会将数据复制到临时表，然后将实际表清空，再从临时表按照指定顺序插入到实际表。每个表的重载操作都在一个单一的事务中处理。

对于分区表来说，可以单独重载子分区的数据。但是无论如何，数据都会从根分区表插入，这会在整个表上产生 ROW EXCLUSIVE锁。

选项

-a (不提示)

可选。如果指定，则gpreload工具不会提示用户进行确认。

-d database

包含要重新装载的表的数据库。gpreload工具以运行该工具的用户的身份连接到数据库。

-p port

Greenplum数据库的Master端口。如果未指定，则使用PGPORT环境变量的值。如果该值不可用，则返回错误。

{-t | --table-file } path_to_file

包含要重新装载的方案限定的表名列表的文件位置和名称，该文件中还有要从Greenplum数据库重新排序的列名。仅支持用户定义的表。视图或系统目录表不受支持。

如果在文件中列出的表上定义有索引，gpreload会提示继续。

每一行都指定一个表名和要排序的列的列表。这是文件中每一行的格式：

```
schema.table_name: column [desc] [, column2 [desc] ... ]
```

表名后跟一个冒号 (:)，然后至少一个列名。如果指定多个列，请使用逗号分隔列名称。列按升序排序。在列名后面指定关键字desc以降序对列进行排序。

通配符不受支持。

如果文件中有错误，gpreload会报告第一个错误并退出。

以下例子展示了重载三个表：

```
public.clients: region, state, rep_id desc
public.merchants: region, state
test.lineitem: group, assy, whse
```

在第一个表public.clients中，rep_id列中的数据被按照

降序排序。其他列中的数据被按照升序排序。
--version (显示工具版本)
列出工具版本。
-? (帮助)
提供在线帮助。

示例

此示例命令将重新装载文件data-tables.txt中列出的数据库mytest中的表。

```
gpreload -d mytest --table-file data-tables.txt
```

另见

*Greenplum*数据库参考指南中的CREATE TABLE。

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

aprecoversea

恢复由gpbackup创建的Greenplum数据库备份。默认gprestore 使用位于Greenplum Master主机数据目录下的元数据文件和DDL文件，使用存储于Segment主机上的表数据CSV 文件。

概要

```
gprestore --timestamp YYYYMMDDHHMMSS
  [--backup-dir directory]
  [--create-db]
  [--debug]
  [--exclude-schema schema_name]
  [--exclude-table schema.table]
  [--exclude-table-file file_name]
  [--include-schema schema_name]
  [--include-table schema.table]
  [--include-table-file file_name]
  [--data-only | --metadata-only]
  [--jobs int]
  [--on-error-continue]
  [--plugin-config config_file_location]
  [--quiet]
  [--redirect-db database_name]
  [--verbose]
  [--version]
  [--with-globals]
  [--with-stats]
```

gprestore --help

描述

使用gprestore从备份集恢复时，必须包含--timestamp选项来指定确切一个来自备份集的恢复时间值（YYYYMMDDHHMMSS）。如果在备份时指定了--backup-dir，那么gprestore同样要指定--backup-dir来获取备份文件。

如果指定的是增量备份，那么还需要一个完全备份的文件集合（一个全备和所有需要的增量备份）。gprestore会在开始恢复之前验证备份集合是可用的。

Important: 对于增量备份集，备份必须在单独设备上。例如，备份集的文件必须全部在 Data Domain系统。

有关增量备份的详细信息，请见[使用gpbackup和gprestore创建增量备份](#)。

当从备份集恢复时，gprestore会将数据恢复到创建备份时指定的数据库。如果目标数据库存在，并且数据库中存在该表，恢复操作会失败。如果目标数据库不存在，可以通过指定--create-db选项来创建数据库。也可以通过指定--redirect-db 选项来将数据恢复到不同的数据库。

当恢复分区表的子分区时，分区表和数据会一同被恢复。例如，使用`gpbackup`的`--include-table-file`选项创建了一个备份。恢复数据时会恢复创建分区表并只将数据恢复到列表文件中指定的分区。

`Greenplum`数据库系统对象会自动包含在`gpbackup`备份集中，但是只有在用`gprestore`进行恢复时指定`--with-globals`选项才会进行恢复。相似的情况还有，如果备份时使用`--with-stats`选项备份了查询计划统计信息，那么在使用`gprestore`进行恢复时必须指定`--with-stats`才能对这部分数据进行恢复。默认情况下，只有备份集中的数据库对象会被恢复。

恢复操作的性能可以通过提高创建多个并行连接的方式来同时恢复多个表和元数据。默认`gprestore`使用1个连接，但是我们可以通过`--jobs`选项来为大的恢复过程增加处理进程数。

当一个操作完成后，`gprestore`会返回一个状态码。详见[Return Codes](#)。

`gprestore`在恢复操作完成后可以发送邮件通知。可以指定发送邮件和接收邮件的配置文件。详见[配置邮件通知](#)。

Note: 该工具在系统内部采用SSH连接执行各项操作任务。在大型Greenplum集群、云部署或每台主机部署了大量的segment实例时，可能会遇到超过主机最大授权连接数限制的情况。此时需要考虑更新SSH配置参数`MaxStartups`以提高该限制。更多关于SSH配置的选项，请参考您的Linux分发版的SSH文档。

选项

--timestamp YYYYMMDDHHMMSS

必须。指定一个用来恢复`gpbackup`备份集的时间戳。默认`gprestore`会尝试从Master主机的`$MASTER_DATA_DIRECTORY/backups/YYYYMMDD/YYYYMMDDhhmmss/`路径下定位时间戳对应的元数据文件，从每个Segment主机的`<seg_dir>/backups/YYYYMMDD/YYYYMMDDhhmmss/`路径下定位CSV数据文件。

--backup-dir directory

可选。从指定的路径下找所有备份文件（元数据文件和数据文件）。必须指定`directory`为一个绝对路径（不能是相对路径）。如果没有指定该选项，`gprestore`会尝试从Master主机的`$MASTER_DATA_DIRECTORY/backups/YYYYMMDD/YYYYMMDDhhmmss/`路径下定位时间戳对应的元数据文件，从每个Segment主机的`<seg_dir>/backups/YYYYMMDD/YYYYMMDDhhmmss/`路径下定位CSV数据文件。如果`gpbackup`操作指定了该选项，那么恢复时请同时指定该选项。

该选项不能和`--plugin-config`选项一起使用。

--create-db

可选。在恢复数据库对象元数据之前先创建数据库。
通过复制空的系统标准数据库`template0`来创建指定的数据库。

--data-only

可选。仅恢复`gpbackup`创建的表数据，不恢复创建数据库表的操作。该选项

假定数据库中已经存在对应的表。如果要恢复指定的集合，可以通过指定选项来包含表/模式或排除表/模式。指定--with-stats选项来从备份集恢复表的统计信息。

备份集必须包含要被恢复的表数据。例如，gpbackup采用选项--metadata-only备份的备份集不包含表数据，不能用来恢复数据。

如果要仅恢复数据库表，不恢复表里的数据，参见选项 [--metadata-only](#)。

--debug

可选。显示操作期间的详细信息和调试日志。

--exclude-schema schema_name

可选。指定恢复操作期间要排除的数据库模式。可以多次指定以排除多个模式。该选项不能和--include-schema或表过滤选项（例如：--include-table）一起使用。

--exclude-table schema.table

可选。指定恢复操作期间要排除的表。指定的格式必须为<schema-name>. <table-name>。如果表名或模式名使用了非小写字母、数字或下划线，那改名字可以用双引号包裹。客户可以多次指定该选项。如果表不在备份集中，恢复操作会失败。不能指定分区表的子分区。

该选项不能与--exclude-schema或表过滤选项（例如：--include-table）一起使用。

--exclude-table-file file_name

可选。指定恢复期间要排除的表的列表文件。文件为text格式，并且每行都必须定义一个单独的表，格式为 <schema-name>. <table-name>。文件不能包含多余的行。如果表名或模式名使用了非小写字母、数字或下划线，那改名字可以用双引号包裹。客户可以多次指定该选项。如果表不在备份集中，恢复操作会失败。不能指定分区表的子分区。

该选项不能与--exclude-schema或表过滤选项（例如：--include-table）一起使用。

--include-schema schema_name

可选。指定要恢复的数据库模式。可以多次指定该选项以包含多个模式。如果指定了该选项，那定义的任何模式都必须在备份集中。没在--include-schema选项中指定的所有模式都会被忽略。

如果指定的模式在目标数据库中存在，该工具会产生一个错误然后继续操作。如果被恢复的表在数据库中存在，该工具会失败。

如果备份集在多个模式下存在依赖关系，那不能使用该选项。

更多信息请见[过滤备份和恢复的内容](#)。

--include-table schema.table

可选。指定一个要恢复的表。格式必须为<schema-name>. <table-name>。如果任何表名或模式名使用非小写字母、数字或下划线，那么可以用双引号包裹名字。该选项可以多次指定。不能指定分区表的子分区。也可以指定一个有效的模式或视图。

如果指定了该选项，工具不会自动恢复依赖对象。必须指定需要的依赖对象。例如，如果恢复一个视图，那么必须也恢复它对应的表。如果恢复的表使用到了一个序列，那么也必须恢复该序列。这些对应的对象也必须都在备份集中存在。

不能将该选项与--include-schema或表过滤选项（例如--exclude-table-file）一起使用。

--include-table-file file_name

可选。指定一个包含恢复表名的列表文件。每行代表一张单独的表，格式必须为 <schema-name>. <table-name>。该文件不能包含多余的行。如果表

名或模式名使用了非小写字母、数字或下划线，那改名字可以用双引号包裹。客户可以多次指定该选项。如果表不在备份集中，恢复操作会失败。不能指定分区表的子分区。

也可以指定一个有效的模式或视图。

如果指定了该选项，工具不会自动恢复依赖对象。必须指定需要的依赖对象。例如，如果恢复一个视图，那么必须也恢复它对应的表。如果恢复的表使用到了一个序列，那么也必须恢复该序列。这些对应的对象也必须都在备份集中存在。

如果使用--include-table-file选项，`gprestore`不会创建表的角色或用户集合。工具会恢复表的索引和规则。触发器也会被恢复，但是Greenplum目前不支持触发器。

更多信息请见[过滤备份或恢复的内容](#)。

--jobs int

可选。指定恢复表数据和元数据的并行连接的数量。默认`gprestore`使用1个连接。增加该参数会提高数据恢复的速度。

Note: 如果使用`gpbackup --single-data-file`选项合并表备份为每个Segment节点1个单独的文件，那么不能在恢复过程中指定--jobs超过1来进行并行恢复。

--metadata-only

可选。从一个`gpbackup`创建的备份集中创建数据库表，不恢复数据。该选项假设目标数据库中不存在这些表。要从备份集创建指定的表集合，可以通过指定对应的选项来选择或排除相应的表或模式。指定选项--with-globals来恢复Greenplum数据库系统对象。

备份集必须包含要恢复的表的DDL数据。例如，`gpbackup`选项--data-only生成的备份集就不包含表的DDL信息。

创建完数据库表后再恢复表数据，请见选项[--data-only](#)。

--on-error-continue

可选。指定该选项以在创建数据库元数据（例如表、对象或函数）或恢复数据过程中出现SQL错误时，忽略错误继续执行。如果有其他类型错误出现，工具会退出。工具会显示错误信息汇总并将错误信息写入到`gprestore`日志文件，并继续恢复操作。

默认为第一次出现错误即退出。

--plugin-config config-file_location

指定`gpbackup`插件配置文件位置，它是一个YAML格式的文本文件。该文件包含`gprestore`在恢复操作期间使用的插件的配置信息。

如果备份数据库时指定了--plugin-config选项，那么在从该备份集恢复数据时也必须指定该配置文件。

该选项不能与--backup-dir同时使用。

有关存储插件应用的详细信息，请见[使用gpbackup存储插件](#)。

--quiet

可选。禁止任何非告警、非错误日志输出。

--redirect-db database_name

可选。恢复到指定的`database_name`，而不是备份时的默认数据库名。

--verbose

可选。显示恢复操作期间的详细日志信息。

--version

可选。打印版本号并退出。

--with-globals

可选。恢复备份集合中的Greenplum数据库系统对象，和其余数据库对象。详见[备份或还原中包含的对象](#)。

--with-stats

可选。从备份集合恢复查询计划统计信息。

--help

显示在线帮助信息。

Return Codes

`gprestore`完成后会返回如下返回码之一

- **0** – 成功完成恢复。
- **1** – 恢复完成，没有严重错误。具体信息见日志文件。
- **2** – 恢复失败，有严重错误。具体信息见日志文件。

示例

创建demo数据库并恢复备份集合中指定时间的所有表名和模式名：

```
$ dropdb demo
$ gprestore --timestamp 20171103152558 --create-db
```

恢复备份集到"demo2"数据库而不是默认备份的"demo"数据库：

```
$ createdb demo2
$ gprestore --timestamp 20171103152558 --redirect-db demo2
```

恢复数据库全局对象和查询计划统计信息，而不是默认的全部数据：

```
$ gprestore --timestamp 20171103152558 --create-db --with-globals --
with-stats
```

使用创建在`/home/gpadmin/backup`下的文件恢复数据，创建8个并行连接：

```
$ gprestore --backup-dir /home/gpadmin/backups/ --timestamp
20171103153156 --create-db --jobs 8
```

仅恢复备份集中的"wikipedia"模式：

```
$ dropdb demo
$ gprestore --include-schema wikipedia --backup-dir
/home/gpadmin/backups/ --timestamp 20171103153156 --create-db
```

如果从增量备份集中恢复，所有需要的备份文件都必须对`gprestore`可用。例如，以下时间戳指定增量备份集。20170514054532时增量备份集合的全备份。

```
20170514054532 (full backup)
20170714095512
20170914081205
20171114064330
20180114051246
```

以下gprestore命令指定时间戳20121114064330。时间戳为20120714095512和20120914081205的增量备份和全量备份必须都能被恢复程序访问。

```
gprestore --timestamp 20121114064330 --redirect-db mystest --create-db
```

另见

[gpbackup](#)、[使用gpbackup和gprestore并行备份](#)和[使用带有gpbackup和gprestore的S3存储插件](#)

工具指南

管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfldist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

一次在多个主机之间复制文件。

概要

```
gpscp { -f hostfile_gpssh | -h hostname [-h hostname ...] }
       [-J character] [-v]
[[user@]hostname:]file_to_copy [...]
[[user@]hostname:]copy_to_path

gpscp -?

gpscp --version
```

描述

gpscp工具允许用户使用SCP（安全复制）在一个命令中将一个或多个文件从指定的主机 复制到其他指定的主机。例如，用户可以同时将文件从Greenplum数据库的Master主机复制到所有的Segment主机。

要指定SCP会话中涉及的主机，请使用-f选项指定包含主机名列表的文件，或使用-h选项在命令行上命名单个主机名。至少需要一个主机名(-h) 或主机文件(-f)。-J选项允许用户指定单个字符来替代copy from和copy to的目标字符串中的`hostname`。如果未指定-J，则默认的替代字符是等号(=)。例如，以下命令 将从本地主机将.bashrc复制到hostfile_gpssh中指定的所有主机上的/home/gpadmin：

```
gpscp -f hostfile_gpssh .bashrc =:/home/gpadmin
```

如果未在主机列表中指定用户名，或者在文件路径中指定了`user@`，则gpscp将以当前登录用户的身份复制文件。要确定当前登录的用户，请执行`whoami`命令。默认情况下，登录后gpscp将转到远程主机上会话用户的\$HOME。为确保将文件复制到远程主机上的正确位置，建议使用绝对路径。

在使用gpscp之前，用户必须在涉及SCP会话的主机之间建立可信的主机设置。用户可以使用工具`gpssh-exkeys`更新已知的主机文件并



gprecoverseg

在主机之间交换公钥（如果尚未这样做的话）。

选项

-f *hostfile_gpssh*

指定包含将参与此SCP会话的主机列表的文件的名称。主机文件的语法是每行一台主机，如下所示：

```
<hostname>
```

-h *hostname*

指定将参与此SCP会话的单个主机名。用户可以多次使用-h选项来指定多个主机名。

-J *character*

-J选项允许用户指定单个字符来替代copy from和copy to目标字符串中的*hostname*。如果未指定-J，则默认的替代字符是等号(=)。

-v (详细模式)

可选。除SCP命令输出外还报告其他消息。

file_to_copy

必需。要复制到其他主机（或文件位置）的文件的文件名（或绝对路径）。这可以是本地主机上的文件或另一台被提及主机上的文件。

copy_to_path

必需。用户希望将文件复制到指定主机上的路径。如果没有使用绝对路径，文件将被复制到相对于会话用户的\$HOME的位置。

用户也可以使用等号'='（或使用-J选项）指定的另一个字符代替*hostname*。这将随后在所提供的主机文件（-f）或-h选项中指定的每个主机名中进行替换。

-? (帮助)

显示在线帮助。

--version

显示此工具的版本。

示例

将名为installer.tar的文件复制到文件hostfile_gpssh 中的所有主机上的/路径下。

```
gpscp -f hostfile_gpssh installer.tar ::/
```

将名为*myfuncs.so*的文件复制到名为sdw1和 sdw2的主机上的指定位置：

```
gpscp -h sdw1 -h sdw2 myfuncs.so =:/usr/local/greenplum-db/lib
```

另见

[gpssh](#)、[gpssh-exkeys](#)

Greenplum数据库® 6.0文档

[工具指南](#)

[管理工具参考](#)

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

在Segment主机上安装Greenplum数据库。

概要

```
gpseginstall -f hostfile [-u gpdb_admin_user] [-p password]
[-c u|p|c|s|E|e|l|v]
```

```
gpseginstall --help
```

描述

gpseginstall工具提供了一种在主机列表文件中指定的Segment主机上快速安装 Greenplum数据库的简单方法。该工具不会在Master主机上安装或更新Greenplum数据库。用户可以以 `root` 身份或以非`root`用户身份运行`gpseginstall`。`gpseginstall`不执行数据库初始化。有关初始化数据库的更多信息，请参阅 `gpinitsystem`。

以`root`身份运行时，`gpseginstall`默认操作是添加一个系统用户（默认是`gpadmin`），创建口令（默认是`changeme`），并在部分主机上部署和安装Greenplum数据库。为此，`gpseginstall`从当前用户的环境变量（`$GPHOME`）中的安装路径下查找Master上的当前Greenplum 数据库二进制文件。它将Greenplum数据库软件压缩成`.tar.gz`文件并执行MD5 校验和以验证文件完整性。

然后，它将Greenplum数据库拷贝到Segment主机，安装（解压缩）Greenplum数据库，并将 Greenplum数据库安装的所有权更改为用户使用`-u`选项指定的系统用户。最后，它将作为根用户以及用`-u`选项指定的系统用户在所有Greenplum数据库主机之间进行密钥交换。`gpseginstall`还会执行用户限制检查并验证所有Segment上的Greenplum数据库的版本号。

如果以非`root`用户的身份运行`gpseginstall`，则`gpseginstall`仅会在Segment主机上压缩、复制并 安装Greenplum数据库。它还可以为当前系统用户在Greenplum数据库主机之间交换密钥，并验证所有Segment上的Greenplum数据库的版本号。

gprecoverseg

选项

-c | --commands *option_list*

可选。这使用户可以自定义gpseginstall操作。请注意，如果不在 gpseginstall语法中指定-c选项，则默认执行这些命令选项。

- u: 添加系统用户（仅限root使用）。
- p: 为系统用户更改口令（仅限root使用）。
- s: 将Greenplum数据库压缩、复制、解压（安装到）所有Segment。
- c: 更改Segment主机上的Greenplum数据库安装目录的所有权（仅限root使用）。
- E: 为root用户在Greenplum数据库的Master主机和Segment 主机之间进行密钥交换（仅限root使用）。
- e: 为非root系统用户在Greenplum数据库的Master主机和 Segment主机之间进行密钥交换。
- l: （仅适用于Linux）将新用户添加到Segment主机时，检查并修改用户限制配置文件（/etc/security/limits.conf）（仅限root使用）。
- v: 验证在所有Segment上运行的Greenplum数据库的版本。gpseginstall检查由\$GPHOME环境变量引用的Greenplum数据库安装的版本号以及到安装目录的符号链接。如果存在版本号不匹配或者无法找到Greenplum数据库 安装目录，则会发生错误。

-f | --file *hostfile*

必需。指定包含要安装Greenplum数据库的Segment主机的文件。主机列表文件的每行必须包含一个主机名，并包含Greenplum系统中每个主机的主机名。确保没有空行或多余的空格。如果主机有多个配置的主机名，则每个主机只能使用一个主机名。例如：

```
sdw1-1
sdw2-1
sdw3-1
sdw4-1
```

如果可用，用户可以使用在Greenplum数据库主机间交换密钥时使用的同一个 gpssh-exkeys主机列表。

-p | --password *password*

可选。使用-u选项设置用户指定用户的口令。默认口令是changeme。此选项仅在以root用户身份运行gpsetinstall 时

使用。

推荐的安全最佳实践：

- 一直使用口令。
- 不要使用默认口令。
- 安装后立即更改默认口令。

-u | --user *user*

可选。指定系统用户。此用户也是Greenplum数据库管理用户。此用户拥有Greenplum数据库安装 并且管理数据库。这也是Greenplum数据库启动/初始化的用户。此选项仅在以root 身份运行gpseginstall时可用。默认是gpadmin。

--help (帮助)

显示在线帮助。

示例

以root身份，在所有Segment上安装Greenplum数据库，将系统用户保留为默认值（gpadmin）并将gpadmin的口令设置为secret123：

```
# gpseginstall -f my_host_list_file -p secret123
```

作为非root用户，将Greenplum数据库二进制文件压缩并复制到所有Segment（作为gpadmin）：

```
$ gpseginstall -f host_file
```

以root身份，添加用户（gpadmin2），为该用户设置口令（secret1234），作为新用户在主机之间交换密钥，检查用户限制，验证版本号，但不要更改Greenplum二进制文件的所有权，压缩/复制/在Segment上安装Greenplum数据库，或者以root身份交换密钥。

```
$ gpseginstall -f host_file -u gpadmin2 -p secret1234 -c
upelv
```

另见

[gpinit system](#)、[gpssh-exkeys](#)

[工具指南](#)

[管理工具参考](#)

[analyzedb](#)

[gpactivatestandby](#)

[gpaddmirrors](#)

[gpbackup](#)

[gpcheck](#)

[gpcheckcat](#)

[gpcheckperf](#)

[gpconfig](#)

[gpdeletesystem](#)

[gpexpand](#)

[gpfdist](#)

[gpinitstandby](#)

[gpinitsystem](#)

[gupload](#)

[gplogfilter](#)

[gpmapreduce](#)

[gpmovemirrors](#)

[gpperfmon_install](#)

[gppkg](#)

一次提供对多台主机的SSH访问。

概要

```
gpssh { -f hostfile_gpssh | -h hostname [-h hostname ...]
} [-s] [-e]
      [-d seconds] [-t multiplier] [-v]
      [bash_command]

gpssh -?

gpssh --version
```

描述

gpssh工具允许用户使用SSH（安全shell）一次在多台主机上运行bash shell命令。用户可以通过在命令行上指定一个命令来执行单个命令，也可以省略该命令以进入交互式命令行会话。

要指定参与SSH会话的主机，请使用-f选项指定包含主机名列表的文件，或使用-h选项在命令行上指明单个主机名。至少需要一个主机名(-h)或主机文件(-f)。请注意，当前主机默认不包含在会话中。要包含本地主机，用户必须在会话中涉及的主机列表中明确声明它。

在使用gpssh之前，用户必须在涉及SSH会话的主机之间建立可信的主机设置。用户可以使用工具gpssh-exkeys更新已知的主机文件并在主机之间交换公钥（如果尚未这样做的话）。

如果用户没有在命令行上指定命令，gpssh将进入交互模式。在gpssh命令提示符(=>)处，用户可以像在常规bash终端命令行中那样输入命令，并且该命令将在会话涉及的所有主机上执行。要结束交互式会话，请按键盘上的CTRL+D或键入exit或quit。

如果主机文件中没有指定用户名，则gpssh将以当前登录的用户身份执行命令。要确定当前登录的用户，请执行whoami命令。默认情况下，登录后gpssh将转到远程主机上的会话用户的\$HOME。为确保所有远程主机上的命令都能正确执行，应始终输入绝对路径。

gprecoverseg

如果在使用gpssh时遇到网络超时问题，可以使用-d和-t选项或者在gpssh.conf文件中设置参数来控制 gpssh在验证初始ssh连接时使用的时间。有关配置文件的信息，请参阅gpssh配置文件[gpssh配置文件](#)。

选项

bash_command

在此会话中涉及的所有主机上执行的bash shell命令（可选择封闭在引号中）。如果未指定，则gpssh启动交互式会话。

-d (延迟) *seconds*

可选。指定用ssh开始gpssh交互开始时等待的时间（以秒为单位）。默认值是0.05。此选项将覆盖gpssh.conf 配置文件中指定的delaybeforesend的值。

增加此值可能会导致在gpssh启动过程中等待很长时间。

-e (回显)

可选。以非交互模式运行时，回显传递给每个主机的命令及其结果输出。

-f *hostfile_gpssh*

指定包含将参与此SSH会话的主机列表的文件的名称。主机文件的语法是每行一个主机。

-h *hostname*

指定将参与此SSH会话的单个主机名。用户可以多次使用-h 选项来指定多个主机名。

-s

可选。如果指定，则在执行目标主机上的任何命令之前，gpssh将在 \$GPHOME环境变量指定的目录中引用文件greenplum_path.sh。

该选项对交互模式和单命令模式都有效。

-t *multiplier*

可选。大于0的十进制数，它是gpssh在验证ssh 提示时使用的超时的倍数。默认值为1。此选项将覆盖gpssh.conf 配置文件中的prompt_validation_timeout值。

增加此值对gpssh启动过程影响不大。

-v (详细模式)

可选。在非交互模式下运行时，除了输出命令外，还会报告其他消息。

--version

显示此工具的版本。

-? (帮助)

显示在线帮助。

gpssh配置文件

`gpssh.conf`文件包含的参数可让用户调整gpssh在验证初始 ssh连接时使用的时间。gpssh会话使用ssh 执行命令之前，这些参数会影响网络连接。该文件的位置由环境MASTER_DATA_DIRECTORY 指定。如果未定义环境变量或`gpssh.conf`文件不存在，gpssh将使用默认值或使用-d和-t选项设置的值。有关环境变量的信息，请参考*Greenplum*数据库参考指南。

`gpssh.conf`文件是由[gpssh]部分和参数组成的文本文件。#（井号）代表注释的开始。这是一个`gpssh.conf`文件的例子。

```
[gpssh]
delaybeforesend = 0.05
prompt_validation_timeout = 1.0
sync_retries = 5
```

这些是`gpssh.conf`参数。

`delaybeforesend = seconds`

指定用ssh启动gpssh交互时等待的时间（以秒为单位）。默认是0.05。增加此值可能会导致gpssh启动过程中等待很长时间。-d选项覆盖此参数。

`prompt_validation_timeout = multiplier`

大于0的十进制数，它是gpssh在验证ssh 提示时使用的超时的倍数。增加这个值对gpssh启动过程影响不大。默认值是1。-t选项覆盖此参数。

`sync_retries = attempts`

一个非负整数，指定gpssh尝试连接远程Greenplum数据库主机的最大次数。缺省值为3。如果值为0，则如果初始连接尝试失败，gpssh将返回错误。增加尝试次数也增加了重试次数之间的时间。此参数不能使用命令行选项进行配置。
-t选项也影响重试尝试之间的时间。

增加此值可以弥补网络性能下降或部分主机性能问题，例如繁重的CPU或I/O负载。但是，当无法建立连接时，增加的值也会增加错误被返回时的延迟。

示例

与文件`hostfile_gpssh`中列出的所有主机启动交互式SSH会话组：

```
$ gpssh -f hostfile_gpssh
```

在gpssh交互式命令提示符处，在此会话中涉及的所有主机上运行shell命令。

```
=> ls -a /data/primary/*
```

退出交互式会话：

```
=> exit  
=> quit
```

使用名为sdw1和sdw2的主机启动非交互式SSH会话组，并将包含多个名为command_file的命令的文件传递给gpssh：

```
$ gpssh -h sdw1 -h sdw2 -v -e < command_file
```

在主机sdw2和localhost上以非交互模式执行单个命令：

```
$ gpssh -h sdw2 -h localhost -v -e 'ls -a /data/primary/*'  
$ gpssh -h sdw2 -h localhost -v -e 'echo  
$GPHOME'  
$ gpssh -h sdw2 -h localhost -v -e 'ls -l |  
wc -l'
```

另见

[gpssh-exkeys](#)、[gpscp](#)

Greenplum数据库® 6.0文档

[工具指南](#)

[管理工具参考](#)

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

在主机之间交换SSH公钥。

概要

```
gpssh-exkeys -f hostfile_exkeys | -h hostname [-h hostname  
...]
```

```
gpssh-exkeys -e hostfile_exkeys -x  
hostfile_gpexpand
```

```
gpssh-exkeys -?
```

```
gpssh-exkeys --version
```

描述

`gpssh-exkeys` 工具在指定的主机名（或主机地址）之间交换SSH密钥。这允许 Greenplum 主机和网络接口之间的 SSH 连接，而不需要口令提示。该工具用于最初将 Greenplum 数据库系统准备好用于无口令的 SSH 访问，以及在扩展 Greenplum 数据库系统时添加额外的 ssh 密钥。

要指定参与初始SSH密钥交换的主机，请使用`-f`选项指定包含主机名列表的文件（推荐），或使用`-h`选项在命令行上指出单个主机名。至少需要一个主机名（`-h`）或主机文件。请注意，本地主机默认包含在密钥交换中。

指定要添加到现有Greenplum数据库系统的新扩展主机，请使用`-e`和`-x`选项。`-e`选项指定一个文件，其中包含系统中已有SSH密钥的现有主机列表。`-x`选项指定一个文件，其中包含需要参与SSH密钥交换的新主机列表。

密钥要作为当前登录的用户交换。用户应该执行两次密钥交换过程：一次作为`root`用户，一次作为`gpadmin`用户（拥有Greenplum数据库安装的用户）。Greenplum数据库管理要求在Greenplum数据库系统的所有主机上创建相同的非`root`用户，并且这些工具必须能以该用户的身份连接到所有主机而无需口令。

`gpssh-exkeys`工具使用以下步骤执行密钥交换：

gprecoverseg

- 为当前用户创建一个RSA标识密钥对（如果尚不存在）。该密钥对中的公钥被添加到当前用户的*authorized_keys*文件中。
- 使用-h、-f、-e以及-x选项指定的每台主机的主机密钥更新当前用户的*known_hosts*文件。
- 使用ssh连接到每个主机，并获取*authorized_keys*、*known_hosts*和*id_rsa.pub*文件以设置无口令访问。
- 将从每个主机获取的*id_rsa.pub*文件中的密钥添加到当前用户的*authorized_keys*文件中。
- 使用新的主机信息（如果有）更新所有主机上的*authorized_keys*、*known_hosts*和*id_rsa.pub*文件。

选项

-e *hostfile_exkeys*

进行系统扩展时，这个选项指定包含当前Greenplum系统中每台主机（Master、后备Master和Segment）的所有已配置主机名和主机地址（接口名称）的文件的名称和位置，每行一个名称，其中没有空行或额外的空格。这个文件中指定的主机不能出现在-x使用的主机文件中。

-f *hostfile_exkeys*

指定包含Greenplum系统中每台主机（Master、后备Master和Segment）的所有已配置主机名和主机地址（接口名称）的文件的名称和位置，每行一个名称，其中没有空行或额外的空格。

-h *hostname*

指定将参与SSH密钥交换的单个主机名（或主机地址）。用户可以多次使用-h选项来指定多个主机名和主机地址。

--version

显示此工具的版本。

-x *hostfile_gpexpand*

进行系统扩展时，这是一个文件的名称和位置，该文件包含要添加到Greenplum系统的每台新Segment主机的所有已配置主机名和主机地址（接口名称），每行一个名称，其中没有空行或额外的空格。这个文件中指定的主机不能出现在-e使用的主机文件中。

-? (帮助)

显示在线帮助。

示例

在文件hostfile_exkeys中列出的所有主机名和地址之间交换SSH密钥：

```
$ gpssh-exkeys -f hostfile_exkeys
```

在主机sdw1、sdw2和sdw3之间交换SSH密钥：

```
$ gpssh-exkeys -h sdw1 -h sdw2 -h sdw3
```

在现有主机sdw1、sdw2和sdw3，新主机sdw4和sdw5作为系统扩展操作的一部分：

```
$ cat hostfile_exkeys
      mdw
      mdw-1
      mdw-2
      smdw
      smdw-1
      smdw-2
      sdw1
      sdw1-1
      sdw1-2
      sdw2
      sdw2-1
      sdw2-2
      sdw3
      sdw3-1
      sdw3-2
$ cat hostfile_gpexpand
      sdw4
      sdw4-1
      sdw4-2
      sdw5
      sdw5-1
      sdw5-2
$ gpssh-exkeys -e hostfile_exkeys -x
hostfile_gpexpand
```

另见

[gpssh](#)、[gpscp](#)

Greenplum数据库® 6.0文档

[工具指南](#)

[管理工具参考](#)

[analyzedb](#)[gpactivatestandby](#)[gpaddmirrors](#)[gpbackup](#)[gpcheck](#)[gpcheckcat](#)[gpcheckperf](#)[gpconfig](#)[gpdeletesystem](#)[gpexpand](#)[gpfdist](#)[gpinitstandby](#)[gpinitsystem](#)[gupload](#)[gplogfilter](#)[gpmapreduce](#)[gpmovemirrors](#)[gpperfmon_install](#)[gppkg](#)

启动一个Greenplum数据库系统。

概要

```
gpstart [-d master_data_directory] [-B parallel_processes]
[-R]
      [-m] [-y] [-a] [-t timeout_seconds] [-l
logfile_directory]
      [--skip-heap-checksum-validation]
      [-v | -q]

gpstart -? | -h | --help

gpstart --version
```

描述

gpstart工具用于启动Greenplum数据库服务器进程。当用户启动一个Greenplum数据库系统时，用户实际上是同时启动了几个postgres数据库服务器监听器进程（Master和所有的Segment实例）。**gpstart**工具处理各个实例的启动。每个实例都是并行启动的。

管理员第一次运行**gpstart**时，该工具将在用户的主目录中创建一个名为 .gphostcache的主缓存文件。随后，该工具使用此主机列表更有效地启动系统。如果将新主机添加到系统中，则必须手动从gpadmin用户的主目录中删除此文件。该工具将在下次启动时创建一个新的主机缓存文件。

作为启动过程的一部分，该工具会检查堆表checksum设置是否在集群上启用。如果堆表checksum启用情况在各个实例之间不同，Greenplum数据库不会启动，会返回一个错误信息。该验证选项可以通过指定选项 **--skip-heap-checksum-validation**来停用。更多关于堆表checksum的情况，请见Greenplum数据库管理员指南中的[启用高可用和数据一致性特性](#)部分。

Note: 在启动Greenplum数据库系统之前，用户必须首先使用gpinitsystem初始化系统。堆表checksum特性在系统初始化时配置启用和禁用，系统初始化后不能修改。



选项

-a

不要提示用户确认。

-B *parallel_processes*

并行启动的Segment数。如果未指定，则该工具将启动最多64个并行进程，具体取决于需要启动多少个Segment实例。

-d *master_data_directory*

可选。Master主机的数据目录。如果未指定，则使用为\$MASTER_DATA_DIRECTORY设置的值。

-l *logfile_directory*

写入日志文件的目录。默认为~/gpAdminLogs。

-m

可选。仅启动Master实例，这可能对维护任务有用。该模式只允许连接到utility模式下的Master。例如：

```
PGOPTIONS=' -c gp_session_role=utility' psql
```

此模式下不检查Master和Segment实例的堆表checksum设置一致性。

-q

以静默模式运行。命令输出不显示在屏幕上，但仍然写入日志文件。

-R

以受限模式启动Greenplum数据库（只允许数据库超级用户连接）。

--skip-heap-checksum-validation

启动时指定该选项，该工具不检查Master和Segment之间的堆表checksum一致性。默认情况下该设置在所有实例上是一样的，或者启用或者禁用。

Warning: 不进行该项验证启动Greenplum数据库会导致数据丢失。只有在必须要忽略堆表checksum验证错误以恢复数据或进行错误调试时，才能使用该选项。

-t *timeout_seconds*

指定等待Segment实例启动的超时时间（秒）。如果某个Segment实例异常关闭（例如由于电源故障或终止其postgres数据库监听器进程），由于数据库恢复和验证过程，启动可能需要较长的时间。如果未指定，则默认超时时间为60秒。

-v

显示工具输出的详细状态，进度和错误消息。

-y

可选。不启动后备Master主机。默认是启动后备Master主机和同步过程。

-? | -h | --help

显示在线帮助。

--version

显示工具的版本。

示例

启动Greenplum数据库系统：

```
gpstart
```

以受限模式启动Greenplum数据库系统（仅允许超级用户连接）：

```
gpstart -R
```

仅启动Greenplum主实例并以utility模式连接：

```
gpstart -m PGOPTIONS=''-c gp_session_role=utility' psql
```

另见

[gpstop](#)、[gpinitsystem](#)

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

显示正在运行的Greenplum数据库系统的状态。

概要

```
gpstate [-d master_data_directory] [-B parallel_processes]
          [-s | -b | -Q | -e] [-m | -c] [-p] [-i] [-f] [-v]
          | -q] | -x
          [-l log_directory]

gpstate -? | -h | --help
```

描述

gpstate 工具显示有关正在运行的Greenplum数据库实例的信息。由于Greenplum数据库系统由跨多台机器的多个PostgreSQL数据库实例（Segment）组成，因此用户可能需要了解关于Greenplum数据库系统的额外信息。**gpstate** 工具为Greenplum数据库系统提供了额外的状态信息，例如：

- 哪台Segment主机已被关闭？
- Master和Segment配置信息（主机、数据目录等）。
- 系统使用的端口。
- 主Segment到其相应的镜像Segment的映射。

选项

-b (简要状态)

可选。显示Greenplum数据库系统状态的简要概述。这是默认选项。

-B *parallel_processes*

并行检查的Segment数。如果未指定，则工具将根据需要检查多少个Segment实例启动最多60个并行进程。

-c (显示主Segment到镜像Segment的映射)

可选。显示主Segment到镜像Segment的映射。

-d *master_data_directory*



gprecoverseg

可选。Master的数据目录。如果未指定，则使用为\$MASTER_DATA_DIRECTORY设置的值。

-e (显示镜像状态出问题的Segment)

显示具有潜在问题的主要/镜像Master对的详细信息，例如1) 活动Segment正在Change Tracking模式下运行，表示Segment处于关闭状态；2) 活动Segment处于重新同步模式，这意味着它正在赶上对镜像的更改；3) 一个Segment不是它的首选角色，例如在系统初始化时作为一个主Segment的现在作为一个镜像Segment，这意味着用户可能有一台或多台Segment主机的处理负载不平衡。

-f (显示后备Master的详情)

显示后备Master主机的详细信息（如果配置）。

-i (显示Greenplum数据库的版本)

显示每个实例的Greenplum数据库软件版本信息。

-l *logfile_directory*

写入日志文件的目录。默认为~/gpAdminLogs。

-m (列出镜像)

可选。列出系统中的镜像Segment实例及其当前角色和同步状态。

-p (显示端口)

列出整个Greenplum数据库系统使用的端口号。

-q (没有屏幕输出)

可选。以静默模式运行。除了警告信息之外，屏幕上不显示命令输出。但是，这些信息仍然写入到日志文件中。

-Q (快速状态)

可选。在Master主机上的系统目录中检查Segment的状态。不直接向Segment征询状态。

-s (详细状态)

可选。显示Greenplum数据库系统的详细状态信息。

-v (详细输出)

可选。显示错误消息并输出详细的状态和进度信息。

-x (expand)

Optional. Displays detailed information about the progress and state of a Greenplum system expansion.

-? | -h | --help (help)

显示在线帮助。

输出字段定义

以下输出字段由主机的gpstate -s报告：

Table 1. gpstate为Master给出的输出数据

输出数据	描述
Master host	Master的主机名
Master postgres process ID	主数据库侦听进程的PID
Master data directory	主数据目录的文件系统位置
Master port	Master上postgres数据库监听器进程的端口
Master current role	dispatch = 常规操作模式 utility = 维护模式
Greenplum array configuration type	Standard = 每台主机一个NIC Multi-Home = 每台主机多个NIC
Greenplum init system version	系统第一次初始化时的Greenplum数据库版本
Greenplum current version	Greenplum数据库当前的版本
Postgres version	Greenplum数据库基于的PostgreSQL版本
Greenplum mirroring status	物理镜像或无镜像
Master standby	后备Master的主机名
Standby master state	后备Master的状态：活跃或被动

以下输出字段由gpstate -s为每个Segment报告：

Table 2. gpstate为Segment给出的输出数据

输出数据	描述
Hostname	系统配置的主机名
Address	网络地址主机名 (NIC名称)
Datadir	Segment数据目录的文件系统位置
Port	Segment的postgres数据库监听器进程的端口号
Current Role	Segment的当前角色： <i>Mirror</i> 或 <i>Primary</i>
Preferred Role	系统初始化时的角色： <i>Mirror</i> 或 <i>Primary</i>
Mirror Status	主/镜像Segment对的状态：

	<i>Synchronized</i> = 两者上的数据都是最新
	<i>Resynchronization</i> = 数据当前正被从一个拷贝到另一个
	<i>Change Tracking</i> = Segment停止并且活动Segment正在记录更改
Change tracking data size	当处于 <i>Change Tracking</i> 模式中时，更改日志文件的尺寸（如果应用压缩，尺寸可能增长或者收缩）
Estimated total data to synchronize	当处于 <i>Resynchronization</i> 模式中时，剩下要同步的数据的估计尺寸
Data synchronized	当处于 <i>Resynchronization</i> 模式中时，已经被同步的数据的估计尺寸
Estimated resync progress with mirror	当处于 <i>Resynchronization</i> 模式中时，完成百分比的估计值
Estimated resync end time	当处于 <i>Resynchronization</i> 模式中时，完成时间的估计
File postmaster.pid	postmaster.pid锁文件的状态： <i>Found</i> 或 <i>Missing</i>
PID from postmaster.pid file	postmaster.pid文件中找到的PID
Lock files in /tmp	为Segment的postgres会在/tmp中创建Segment端口锁文件（当Segment关闭时会移除该文件）
Active PID	Segment的活动进程ID
Master reports status as	系统目录中报告的Segment状态： <i>Up</i> 或 <i>Down</i>
Database status	Greenplum数据库对进入请求的状态： <i>Up</i> 、 <i>Down</i> 或 <i>Suspended</i> 。 <i>Suspended</i> 状态意味着当Segment正从一种状态转移到另一种状态时，数据库活动被临时暂停。

以下输出字段由gpstate -f为每个后备Master复制状态报告：

Table 3. gpstate对Master复制给出的输出数据

输出数据	描述
------	----

Standby address	后备Master的主机名
Standby data dir	后备Master的数据目录的文件系统位置
Standby port	后备Master的postgres数据库监听器进程的端口
Standby PID	后备Master的进程ID
Standby status	后备Master的状态: <i>Standby host passive</i>
WAL Sender State	预写式日志 (WAL) 流状态: <i>streaming</i> , <i>startup</i> , <i>backup</i> , <i>catchup</i>
Sync state	WAL发送者同步状态: <i>sync</i>
Sent Location	WAL发送者事务日志 (xlog) 记录发送位置
Flush Location	WAL接收者xlog记录刷入位置
Replay Location	后备上xlog记录重放位置

示例

显示Greenplum数据库系统的详细信息:

```
gpstate -s
```

在Master主机系统目录中快速检查下游Segment:

```
gpstate -Q
```

显示关于镜像Segment实例的信息:

```
gpstate -m
```

显示关于后备Master配置的信息:

```
gpstate -f
```

显示Greenplum软件版本信息:

```
gpstate -i
```

另见

[gpstart](#)、[gpexpandgplogfilter](#)

[工具指南](#)

[管理工具参考](#)

[analyzedb](#)

[gpactivatestandby](#)

[gpaddmirrors](#)

[gpbackup](#)

[gpcheck](#)

[gpcheckcat](#)

[gpcheckperf](#)

[gpconfig](#)

[gpdeletesystem](#)

[gpexpand](#)

[gpfdist](#)

[gpinitstandby](#)

[gpinitsystem](#)

gload

[gplogfilter](#)

[gpmapreduce](#)

[gpmovemirrors](#)

[gpperfmon_install](#)

[gppkg](#)

按照一个YAML格式的控制文件的定义运行一个装载作业。

概要

```
gload -f control_file [-l log_file] [-h hostname] [-p port]
[-U username] [-d database] [-w] [--gpfdist_timeout seconds]
[--no_auto_trans] [[-v | -V] [-q]] [-D]

gload -?

gload --version
```

先决条件

要执行gload命令的客户机必须具有下列要求：

- Python 2.6.2或更新版本，装有pygresql (Python的PostgreSQL接口包)，和pyyaml。注意Python及所需的Python库被包含在Greenplum数据库安装包中，因此如果在gload运行的机器上安装有Greenplum数据库，用户就不需要单独安装Python。
Note: Greenplum数据库的Windows装载客户端仅支持Python 2.5(您可以从<https://www.python.org> 获取)
-
- [gpfdist](#)并行文件分发程序被安装在\$PATH中。这个程序位于Greenplum 数据库的\$GPHOME/bin目录下。
- gload客户机可以访问（被访问）Greenplum数据库集群（Master和Segments）中所有主机。
- gload客户机可以访问（被访问）所有可能用来装载数据的主机（ETL服务器）。

描述

gload是一个数据装载工具，它扮演着Greenplum数据库外部表并行装载 特性的接口的角色。通过一个用YAML格式控制文件定义的装载



gprecoverseg

说明，`gupload` 调用Greenplum数据库的并行文件服务器 ([gpfdist](#)) 执行文件装载，基于源数据的定义创建一个外部表定义，并且指定`INSERT`、`UPDATE`或`MERGE`操作把源数据装载到数据库中的目标表中。

Note: `gpfdist`和`gupload`是在Greenplum的主版本级别有效的。例如，Greenplum 4.x版本的`gpfdist`不能用于Greenplum 5.x或6.x版本。

Note: 如果目标表的列名为保留关键字、有大写字母或包含任何双引号，那么`MERGE`和`UPDATE`操作不被支持。

在目标表上指定多个同时的装载操作时，操作包括在YAML控制文件(控制文件格式见[控制文件格式](#))的SQL集合中指定的任何SQL命令会在单个事务中执行以防止数据不一致。

选项

`-f control_file`

必选项。包含装载说明详情的YAML文件。请见[控制文件格式](#)。

`--gpfdist_timeout seconds`

为`gpfdist`并行文件分发程序发送响应设置超时时间。输入一个从0到30秒的值（输入"0"会禁用超时）。注意在高流量网络上可能需要增加这个值。

`-l log_file`

指定在哪里写日志文件。默认是`~/gpAdminLogs/gupload_YYYYMMDD`。有关日志文件的更多信息请见[日志文件格式](#)。

`--no_auto_trans`

如果用户在目标表上执行单个装载操作，可指定`--no_auto_trans`禁用把装载操作作为单个事务处理的特性。默认情况下，在一个目标表上执行多个同时的操作时，`gupload`把每个装载操作处理为单个事务以防止不一致的数据。

`-q` (无屏幕输出)

运行在静默模式中。命令输出不会被显示在屏幕上，但仍将被写入到日志文件。

`-D` (调试模式)

检查错误情况，但是不执行装载。

`-v` (详细模式)

在装载步骤被执行时，显示它们的详细输出。

`-V` (非常详细模式)

显示非常详细的输出。

`-?` (显示帮助)

显示帮助，然后退出。

--version

显示这个工具的版本，然后退出。

连接选项

-d *database*

要装载到的数据库。如果没有指定，则从装载控制文件、环境变量\$PGDATABASE 读取或者默认为当前系统用户名。

-h *hostname*

指定Greenplum的Master数据库服务器在其上运行的机器的主机名。如果没有指定，会从装载控制文件、环境变量\$PGHOST 读取或者默认为 localhost。

-p *port*

指定Greenplum的Master数据库服务器在其上监听连接的TCP端口。如果没有指定，会从装载控制文件、环境变量\$PGPORT 读取或者默认为5432。

-U *username*

要用其进行连接的数据库角色名。如果没有指定，会从装载控制文件、环境变量\$PGUSER 读取或者默认为当前系统用户名。

-W (强制口令提示)

强制口令提示。如果没有指定，会从环境变量\$PGPASSWORD、\$PGPASSFILE 指定的口令文件或~/.pgpass 中的口令文件中读取口令。如果这些都没有设置，即使没有提供-W，gload也将提示要求一个口令。

控制文件格式

gload控制文件使用 [YAML 1.1](#) 文档格式，然后为定义Greenplum数据库装载操作的多个步骤实现了其自身的模式。该控制文件必须是一个有效的YAML文档。

gload程序按顺序处理控制文件文档并且使用缩进（空格）来判断文档层次以及各个部分之间的关系。空格的使用是有意义的。不能把空格简单地用于格式化目的，并且不能使用制表符。

一个装载控制文件的基础结构是：

```
---
VERSION: 1.0.0.1
DATABASE: db_name
USER: db_username
HOST: master_hostname
PORT: master_port
GLOAD:
```

```

INPUT:
  - SOURCE:
    LOCAL_HOSTNAME:
      - hostname_or_ip
    PORT: http_port
    | PORT_RANGE: [start_port_range, end_port_range]
    FILE:
      - /path/to/input_file
    SSL: true | false
    CERTIFICATES_PATH: /path/to/certificates
  - FULLY_QUALIFIED_DOMAIN_NAME: true | false
  - COLUMNS:
    - field_name: data_type
  - TRANSFORM: 'transformation'
  - TRANSFORM_CONFIG: 'configuration-file-path'
  - MAX_LINE_LENGTH: integer
  - FORMAT: text | csv
  - DELIMITER: 'delimiter_character'
  - ESCAPE: 'escape_character' | 'OFF'
  - NULL_AS: 'null_string'
  - FORCE_NOT_NULL: true | false
  - QUOTE: 'csv_quote_character'
  - HEADER: true | false
  - ENCODING: database_encoding
  - ERROR_LIMIT: integer
  - LOG_ERRORS: true | false
  EXTERNAL:
    - SCHEMA: schema | '%'
  OUTPUT:
    - TABLE: schema.table_name
    - MODE: insert | update | merge
    - MATCH_COLUMNS:
      - target_column_name
    - UPDATE_COLUMNS:
      - target_column_name
    - UPDATE_CONDITION: 'boolean_condition'
    - MAPPING:
      target_column_name: source_column_name | expression
  PRELOAD:
    - TRUNCATE: true | false
    - REUSE_TABLES: true | false
    - STAGING_TABLE: external_table_name
    - FAST_MATCH: true | false
  SQL:
    - BEFORE: "sql_command"
    - AFTER: "sql_command"

```

VERSION

可选。gpload控制文件模式的版本。当前版本是1.0.0.1。

DATABASE

可选。指定Greenplum数据库系统要连接到哪个数据库。如果没有指定则默认为\$PGDATABASE。如果\$PGDATABASE也没有设置，则默认为当前系统用户名。用户还可以在命令行上用 -d选

项指定数据库。

USER

可选。指定用于连接的数据库角色。如果没有指定，默認為当前用户或者\$PGUSER（如果设置）。用户还可以在命令行上用-U选项指定数据库角色。

如果运行gpload的用户不是Greenplum数据库的超级用户，那么必须为该用户授予适当的权限。更多信息请见*Greenplum*数据库参考指南。

HOST

可选。指定Greenplum数据库的Master主机名。如果没有指定，默認為localhost或者\$PGHOST（如果设置）。用户还可以在命令行上用-h选项指定Master主机名。

PORt

可选。指定Greenplum数据库的Master端口。如果没有指定，默認為5432或者\$PGPORT（如果设置）。用户还可以在命令行上用-p选项指定Master端口。

GPOLOAD

必需。开始装载说明部分。GPOLOAD说明必须定义有一个INPUT小节和一个OUTPUT小节。

INPUT

必需。定义要装载的输入数据的位置和格式。gpload将在当前主机上启动gpfdist文件分布程序的一个或者更多实例并且在Greenplum数据库中创建指向源数据的外部表定义。注意在其上运行gpload的主机必须对所有的Greenplum数据库主机（Master和Segment）通过网络可访问。

SOURCE

必需。INPUT说明的SOURCE块定义源文件的位置。一个INPUT小节可以定义多个SOURCE块。每个定义的SOURCE块对应于将在本地机器上启动的一个gpfdist文件分布程序的实例。每个定义的SOURCE块必须有一个FILE说明。

更多关于使用gpfdist并行文件服务器和单个以及多个gpfdist实例的信息，请见*Greenplum*数据库管理员指南中的“装载和卸载数据”部分。

LOCAL_HOSTNAME

可选。指定gpload运行其上的本地机器的主机名或者IP地址。如果这个机器被配置有多个网络接口卡（NICs），用户可以指定每块NIC的主机名或者IP，以便允许网络流量同时使用所有的NIC。默认是仅使用本地机器的

主要主机名或者IP。

PORT

可选。指定`gpfdist`文件分布程序应该使用的特定端口号。用户还可以提供一个`PORT_RANGE`来从指定的范围中选择可用的端口。如果`PORT`和`PORT_RANGE`同时被定义，那么`PORT`优先。如果`PORT`和`PORT_RANGE`都没有定义，默認為在8000和9000之间选择一个可用端口。

如果在`LOCAL_HOSTNAME`中声明多个主机名，这个端口号被用于所有主机。如果用户想要使用所有的NICs装载一个给定目录位置的同一个文件或者文件集合，这种配置就是用户想要的。

PORT_RANGE

可选。可被用来代替`PORT`提供一个端口号范围，`gupload`可以从其中为这个`gpfdist`文件分布程序实例选择一个可用的端口。

FILE

必需。指定本地文件系统上的一个文件位置、命名管道或者目录位置，其中包含要被装载的数据。用户可以声明多个文件，只要所有指定文件中数据的格式相同。

如果这些文件被使用`gzip`或者`bzip2`（有`.gz`或者`.bz2`文件扩展名）压缩，这些文件将被自动解压缩（在用户路径中有`gunzip`或者`bunzip2`）。

在指定要装载哪些源文件时，用户可以使用通配符（`*`）或其他C风格的模式匹配来指示多个文件。被指定的文件假定在相对于`gupload`被执行的当前目录的位置（或者用户可以声明绝对路径）。

SSL

可选。指定SSL加密的使用。如果`SSL`被设置为`true`，`gupload`用`--ssl`启动`gpfdist`服务器并且使用`gpfdists://`协议。

CERTIFICATES_PATH

当`SSL`为`true`时必需；当`SSL`为`false`或者没有指定时不能指定这个参数。`CERTIFICATES_PATH`中指定的位置必须

包含下列文件：

- 服务器证书文件 `server.crt`
- 服务器私钥文件 `server.key`
- 可信证书授权 `root.crt`

根目录(/)不能被指定为
CERTIFICATES_PATH。

FULLY_QUALIFIED_DOMAIN_NAME

可选。指定gpload是否把主机名解析成完全限定的域名 (FQDN) 或者本地主机名。如果值被设置为`true`，名称会被解析到FQDN。如果该值被设置为`false`，则解析到本地主机名。默认是`false`。

在某些情况下可能要求一个完全限定的域名。例如，如果Greenplum 数据库系统在与ETL应用不同的域且该域能够被gpload访问。

COLUMNS

可选。以`field_name: data_type`这样的格式指定源数据文件的模式。源文件中的DELIMITER 字符是分隔两个数据值域 (列) 的东西。一行由一个换行字符 (0x0a决定)。

如果输入COLUMNS没有指定，则使用输出TABLE的模式，意味着源数据必须与目标表具有相同的列序、列数以及数据格式。

默认的source-to-target映射基于这一节定义的列名与目标TABLE 中列名之间的匹配。默认映射可以使用MAPPING小节覆盖。

TRANSFORM

可选。指定传递给gpload的输入转换的名字。有关XML转换的信息，请见Greenplum数据库管理员指南 中的“装载和卸载数据”。

TRANSFORM_CONFIG

当TRANSFORM被指定时，这个元素是必需的。指定在上面TRANSFORM参数中指定的转换的配置文件位置。

MAX_LINE_LENGTH

可选。一个整数，指定传递给gpload 的XML转换数据中一行的最大长度。

FORMAT

可选。指定源数据文件的格式：纯文本 (TEXT) 格

式，逗号分隔值 (CSV) 格式。如果没有指定，这个默认为 TEXT。更多有关源数据格式的信息，请见 *Greenplum数据库管理员指南* 中的“装载和卸载数据”。

DELIMITER

可选。指定在每行数据内分隔列的单个ASCII字符。在TEXT模式中默认是一个制表符，在CSV模式中默认是一个逗号。用户还可以指定一个非可打印ASCII字符或者非可打印 Unicode字符，例如："\x1B"或者"\u001B"。对于非可打印字符也支持转义字符串语法'E'character-code'。ASCII或Unicode字符必须被封闭在单引号中。例如：E'\x1B'或者E'\u001B'。

ESCAPE

指定用于C转义序列（例如\n、\t、\100等等）以及转义可能被当作行列定界符的数据字符的单个字符。确保选择一个在实际列数据中任何地方都没有使用的转义字符。文本格式文件的默认转义字符是一个\（反斜线），csv格式文件的默认转义字符是一个"（双引号）。不过可以指定另一个字符来表示转义。还可以在文本格式文件中通过指定'OFF'值作为转义值来禁用转义。这对于其中嵌有很多不准备作为转义字符的反斜线的文本格式的Web日志数据非常有用。

NULL_AS

可选。指定表示空值的字符串。TEXT模式中默认是\N（反斜线-N），csv模式中默认是没有引用的空的值。即便在TEXT模式中，对于想要把空值与空字符串区分开来的情况，用户也可以使用空字符串。任何匹配这个字符串的源数据项将被认为是一个空值。

FORCE_NOT_NULL

可选。在CSV模式中，处理每个被指定的列，仿佛它被引用并且因此不是一个NULL值。对于CSV模式中的默认空值字符串（两个定界符之间什么都没有），这导致缺失的值被计算为长度为零的字符串。

QUOTE

当FORMAT是CSV时，这个元素是必需的。为CSV模式指定引用字符。默认是双引号（"）。

HEADER

可选。指定数据文件中的第一行是一个头部行（包含列名）并且不应被包括在要被装载的数据中。如果使用多个数据源文件，所有的文件必须有一个头部行。默认是假定输入文件没有头部行。

ENCODING

可选。源数据的字符集编码。可指定一个字符串常量（例如'SQL_ASCII'）、一个整数编码编号，或者指定'DEFAULT'以使用默认客户端编码。如果没有指定，默认的客户端编码会被使用。有关支持的字符集的信息，请见*Greenplum*数据库参考指南。

ERROR_LIMIT

可选。为这个装载操作启用单行错误隔离模式。当被启用时，在输入被处理期间只要没有达到错误限制计数，任何Greenplum数据库 Segment会抛弃有格式错误的输入行。如果错误限制没有达到，所有好的行将被装载并且任何错误行都将被抛弃或者被捕获在错误日志信息中。默认是在遇到第一个错误时中止装载操作。注意单行错误隔离只适用于有格式错误的数据行，例如有额外或者缺失的属性、有错误数据类型的属性或者有无效的客户端编码序列。如果遇到约束错误（例如主键约束）仍将导致装载操作中止。有关处理装载错误的信息，请见*Greenplum*数据库管理员指南中的“装载和卸载数据”。

LOG_ERRORS

当ERROR_LIMIT被声明时，这个元素是可选的。值可以是true或者false。默认值是false。如果值是true，当运行在单行错误隔离模式中时，格式错误的行会被内部记录下来。用户可以用Greenplum数据库的内建SQL函数`gp_read_error_log('table_name')`检查格式错误。如果在装载数据时检测到格式错误，gpload会用包含错误信息的表的名字生成一个警告消息，看起来类似于这个消息。

```
timestamp|WARN|1 bad row, please use GPDB
built-in function gp_read_error_log('table-
name')
to access the detailed error row
```

如果LOG_ERRORS: true被指定，必须指定REUSE_TABLES: true以便在Greenplum数据库的错误日志中保留格式错误。如果没有指定REUSE_TABLES: true，错误信息会在gpload操作后被删除。只有关于格式错误的总结信息会被返回。用户可以用Greenplum数据库的函数`gp_truncate_error_log()`从错误日志中删除格式错误。
更多有关处理装载错误的信息，请见*Greenplum*数据库管理员指南中的“装载和卸载数据”。有

关于gp_read_error_log()函数的信息，请见Greenplum数据库参考指南 中的CREATE EXTERNAL TABLE命令。

EXTERNAL

可选。定义gpload创建的外部表数据库对象所属的方案。

默认是使用Greenplum数据库的search_path。

SCHEMA

当EXTERNAL被声明时，这个元素是必需的。

外部表所在的方案的名称。如果该方案不存在，会返回一个错误。

如果%（百分号字符）被指定，会使用OUTPUT小节中TABLE 指定的表名的方案。

如果这个表名没有指定一个方案，则会使用默认方案。

OUTPUT

必需。定义要被装载到数据库中的目标表和最终数据列值。

TABLE

必需。要装载到其中的目标表名。

MODE

可选。如果没有指定，则默认为INSERT。有三种可用的装载模式：

INSERT - 使用下列方法装载数据到目标表中：

```
INSERT INTO target_table SELECT * FROM
input_data;
```

UPDATE - 更新目标表中MATCH_COLUMNS 属性值等于输入数据并且UPDATE_CONDITION为true（可选条件）的行的UPDATE_COLUMNS。如果目标表的列为保留关键字、有大写字母或包含双引号（" "），那么不支持UPDATE。

MERGE - 插入新行并且更新FOOBAR 属性值等于相应输入数据而且MATCH_COLUMNS为true（可选条件）的已有行的UPDATE_COLUMNS。当源数据中的MATCH_COLUMNS值在目标表数据中没有相应值时会被标识成新行。在那种情况下，源文件中的整个行会被插入，而不仅仅是MATCH和UPDATE列。如果有多个相等的新MATCH_COLUMNS值，只有其中一个新行将被插入。使用UPDATE_CONDITION可过滤掉要抛弃的行。如果目标表的列名为保留关键字、

有大写字母或包含双引号 (" ") , 那么不支持MERGE。

MATCH_COLUMNS

如果MODE为UPDATE或者 MERGE, 则这个元素是必需的。指定被用作更新的连接条件的列。对于要在目标表中更新的行, 指定目标列中的属性值必须等于相应的源数据列值。

UPDATE_COLUMNS

如果MODE为UPDATE 或者MERGE, 则这个元素是必需的。指定对符合MATCH_COLUMNS条件和可选UPDATE_CONDITION的行要更新的列。

UPDATE_CONDITION

可选。指定目标表中要被更新的行 (在MERGE情况下是要被插入的行) 必须满足的一个布尔条件 (类似于在 WHERE子句中声明的那样) 。

MAPPING

可选。如果指定一个映射, 它会覆盖默认的source-to-target列映射。默认的source-to-target映射基于源COLUMNS 小节定义的列名与目标TABLE中列名之间的匹配。映射可以被指定为:

target_column_name: *source_column_name*
或者

target_column_name: '*expression*'

其中*expression*是在查询的SELECT 列表中指定的任意表达式, 例如常量值、列引用、操作符调用、函数调用等等。

PRELOAD

可选。指定在装载操作之前运行的操作。目前唯一的预装载操作是TRUNCATE。

TRUNCATE

可选。如果设置为true, gupload将在装载目标表之前 移除其中所有的行。

REUSE_TABLES

可选。如果设置为true, gupload将不会删除它创建的外部表 对象和阶段性对象。这些对象将被重用于未来使用同一装载说明的装载操作。这会 提高小型装载的性能 (正在进行的到同一目标表的小型装载) 。

如果LOG_ERRORS: true被指定, REUSE_TABLES: true 必须被指定以保留Greenplum数据库错误日志中的格式错误。如果REUSE_TABLES: true 没有被指定, 格式错误信息会在gupload操作之后被删除。

如果*external_table_name*存在, 工具会使用存在的表。

如果OUTPUT表结构与数据库中表结构不吻合, 工具会返

回错误。

STAGING_TABLE

可选。指定gpload操作过程中创建的临时外部表的名称。该外部表会被gpfldist使用。REUSE_TABLES: true必须被指定。如果REUSE_TABLES为false或没有指定，STAGING_TABLE会被忽略。默认情况下，gpload会采用随机名称创建一个临时外部表。

如果*external_table_name*包含点号(.)，gpload会返回错误。如果表已经存在，工具会使用该表。如果表的结构与OUTPUT表定义的结构不匹配，工具会返回错误。

工具会使用EXTERNAL部分定义的SCHEMA值作为*external_table_name*。如果SCHEMA的值为%，*external_table_name*的名字会和目标表相同，TABLE的表结构与OUTPUT部分定义的相同。

如果没有设置SCHEMA，工具会在数据库中搜索表(在search_path中定义的模式下)，如果找不到该表，*external_table_name*表会在PUBLIC模式下被创建。

FAST_MATCH

可选项。如果设置为true，在重用外部表时，gpload仅搜索匹配外部表对象的数据库。工具不会从pg_attribute检查外部表的列名和列类型。设置该值可以在重用外部表的前提下提高，gpload性能。但是如果实际的列不匹配，该工具会在实际执行时返回错误。

默认值为false，工具会去预先检查外部表定义的列名和列类型。

REUSE_TABLES: true也必须被定义。如果REUSE_TABLES为false或者没定义，并且FAST_MATCH: true被指定，gpload会返回告警信息。

SQL

可选。定义在装载操作之前或者之后要运行的SQL命令。用户可以指定多个BEFORE或者AFTER命令。按照想要的执行顺序列出命令。

BEFORE

可选。在装载操作开始之前要运行的一个SQL命令。将命令封闭在引号中。

AFTER

可选。在装载操作完成之后要运行的一个SQL命令。将命令封闭在引号中。

日志文件格式

gpload输出的日志文件具有下面的格式：

```
timestamp|level|message
```

其中`timestamp`的形式是：YYYY-MM-DD HH:MM:SS，`level`是DEBUG、LOG、INFO、ERROR中间的一个，而`message`是普通文本消息。

日志文件中可能让人感兴趣的一些INFO消息是（其中#对应于实际的秒数、数据的单位或者失败的行）：

```
INFO|running time: #.## seconds
INFO|transferred #.# kB of #.# kB.
INFO|gpload succeeded
INFO|gpload succeeded with warnings
INFO|gpload failed
INFO|1 bad row
INFO|# bad rows
```

注解

如果用户的数据库对象名使用双引号标识符（定界的标识符）创建，用户必须在gpload控制文件中用单引号指定定界的名称。例如，如果用户这样创建一个表：

```
CREATE TABLE "MyTable" ("MyColumn" text);
```

用户的YAML格式的gpload控制文件应该按如下方式引用上述表和列名：

```
- COLUMNS:
  - '"MyColumn"': text
OUTPUT:
  - TABLE: public.'"MyTable"'
```

如果YAML控制文件包含Greenplum数据库4.3.x的ERROR_TABLE元素，gpload会提示一个警告，显示表明ERROR_TABLE是不支持的，如果LOG_ERRORS和REUSE_TABLE被设置为true，加载错误信息会被正常处理。当运行在单行事务处理模式时，格式错误的行会被记录到内部数据库日志。

示例

按my_load.yml中的定义运行一个装载作业：

```
gpload -f my_load.yml
```

装载控制文件的例子：

```
---
VERSION: 1.0.0.1
DATABASE: ops
USER: gpadmin
HOST: mdw-1
PORT: 5432
GPOLOAD:
    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
            - descr: text
            - date: date
        - FORMAT: text
        - DELIMITER: '|'
        - ERROR_LIMIT: 25
        - LOG_ERRORS: true
    OUTPUT:
        - TABLE: payables.expenses
        - MODE: INSERT
    PRELOAD:
        - REUSE_TABLES: true
    SQL:
        - BEFORE: "INSERT INTO audit VALUES('start', current_timestamp)"
        - AFTER: "INSERT INTO audit VALUES('end', current_timestamp)"
```

另见

[gpfdist](#), *Greenplum数据库参考指南*中的CREATE EXTERNAL

TABLE。

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

aprecoversea

停止或者重启Greenplum数据库系统。

概要

```
gpstop [-d master_data_directory] [-B parallel_processes]
          [-M smart | fast | immediate] [-t
          timeout_seconds] [-r] [-y] [-a]
          [-l logfile_directory] [-v | -q]

gpstop -m [-d master_data_directory] [-y] [-
l logfile_directory] [-v | -q]

gpstop -u [-d master_data_directory] [-
l logfile_directory] [-v | -q]

gpstop --host host_name [-d
master_data_directory] [-l logfile_directory]
[-t timeout_seconds] [-a] [-v | -q]

gpstop --version

gpstop -? | -h | --help
```

描述

gpstop工具用于停止构成Greenplum数据库系统的数据库服务器。当用户停止一个Greenplum数据库系统时，用户实际上是一次停止几个postgres数据库服务器进程（Master和所有的Segment实例）。**gpstop**工具处理个别实例的关闭。每个实例被并行地关闭。

默认情况下，如果有任何客户端连接到数据库，则不允许关闭Greenplum数据库。使用-M fast选项来回滚所有正在进行的事务，并在关闭之前终止所有连接。如果有任何事务正在进行，则默认行为是在关闭之前等待它们提交。

使用-u选项时，该工具会上传对主pg_hba.conf文件所做的更改，或者在postgresql.conf文件中将运行时配置参数上载到服务上。请注意，任何活动的会话在重新连接到数据库之前都不会获取

选项

-a

不要提示用户确认。

-B *parallel_processes*

并行停止的Segment数。如果未指定，则工具将启动最多64个并行进程，具体取决于需要停止多少个Segment实例。

-d *master_data_directory*

可选。Master主机的数据目录。如果未指定，则使用为\$MASTER_DATA_DIRECTORY设置的值。

--host *host_name*

The utility shuts down the Greenplum Database segment instances on the specified host to allow maintenance on the host. Each primary segment instance on the host is shut down and the associated mirror segment instance is promoted to a primary segment if the mirror segment is on another host. Mirror segment instances on the host are shut down.

The segment instances are not shut down and the utility returns an error in these cases:

- Segment mirroring is not enabled for the system.
- The master or standby master is on the host.
- Both a primary segment instance and its mirror are on the host.

This option cannot be specified with the -m, -r, -u, or -y options.

Note: The gprecoverseg utility restores segment instances. Run gprecoverseg commands to start the segments as mirrors and then to return the segments to their preferred role (primary segments).

-l *logfile_directory*

写入日志文件的目录。默认为~/gpAdminLogs。

-m

可选。关闭在维护模式下启动的Greenplum主实例。

-M fast

快速关闭。任何正在进行的事务都会中断并回滚。

-M immediate

立即关闭。任何正在进行的事务都会中止。

该模式杀死所有postgres进程，而不允许数据库服务器完成事务处理或清理任何临时或进程内工作文件。

-M smart

智能关闭。如果存在活动连接，则此命令将失败并显示警告。这是默认的关闭模式。

-q

以静默模式运行。命令输出不显示在屏幕上，但仍然写入日志文件。

-r

关机完成后重新启动。

-t *timeout_seconds*

指定等待Segment实例关闭的超时阈值（以秒为单位）。如果Segment实例没有在指定的秒数内关闭，`gpstop`将显示一条消息，指示一个或多个Segment仍处于关闭过程中，并且直到Segment实例停止后才能重新启动Greenplum数据库。这个选项在`gpstop`被执行且有非常大的事务需要回滚的情况下非常有用。这些大型事务可能需要一分钟才能回滚，并超过600秒的默认超时时间。

-u

此选项将重新加载Master和Segment的`pg_hba.conf`文件以及`postgresql.conf`文件的运行时参数，但不会关闭数据库阵列。编辑`postgresql.conf`或`pg_hba.conf`之后，使用此选项可使新的配置设置处于活动状态。请注意，这仅适用于设计为运行时的配置参数。

-v

显示工具输出的详细状态，进度和错误消息。

-y

不要停止后备Master进程。默认是停止后备Master。

-? | -h | --help

显示在线帮助。

--version

显示工具的版本。

示例

在智能模式下关闭Greenplum数据库系统：

```
gpstop
```

在快速模式下关闭Greenplum数据库系统：

```
gpstop -M fast
```

停止所有的Segment实例，然后重新启动系统：

```
gpstop -r
```

停止在维护模式下启动的Master实例：

```
gpstop -m
```

在进行配置更改后重新加载`postgresql.conf`和`pg_hba.conf`文

件，但不要关闭Greenplum数据库阵列：

```
gpstop -u
```

另见

[gpstart](#)

Greenplum数据库® 6.0文档

[工具指南](#)

[管理工具参考](#)

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

显示有关操作系统的信息。

概要

```
gpsys1 [ -a | -m | -p ]
gpsys1 -?
gpsys1 --version
```

描述

gpsys1显示当前主机的平台和安装的内存（以字节为单位）。例如：

```
linux 1073741824
```

选项

-a (显示所有)

显示当前主机的平台和内存信息。这是默认选项。

-m (仅显示内存)

以字节为单位显示安装的系统内存。

-p (仅显示平台)

显示操作系统平台。可以是linux、darwin或sunos5。

-? (帮助)

显示在线帮助。

--version

显示此工具的版本。

示例

显示有关当前主机操作系统的信息：



gprecoverseg

gpsys1

另见

[gpcheckperf](#)

Greenplum数据库® 6.0文档

[工具指南](#)

[管理工具参考](#)

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfldist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

管理数据库连接池。

概要

```
pgbouncer [OPTION ...] pgbouncer.ini
```

OPTION

```
[ -d | --daemon ]
[ -R | --restart ]
[ -q | --quiet ]
[ -v | --verbose ]
[ { -u | --user }=username ]
```

```
pgbouncer [ -V | --version ] | [ -h | --help ]
```

描述

PgBouncer是Greenplum和PostgreSQL数据库的轻量级连接池管理器。PgBouncer为每个数据库用户和数据库组合创建一个池。PgBouncer或者为客户端创建一个新的数据库连接，或者重用一个现有的连接。当客户端断开连接时，连接被返回到池中以供重用。

PgBouncer支持PostgreSQL和Greenplum数据库共享的标准连接接口。Greenplum数据库客户端程序（例如`psql`）可以连接到PgBouncer正在运行的主机和端口号，而不是直接连接Greenplum数据库的主机或端口号。

可以通过配置文件声明PgBouncer和他的Greenplum数据库访问。提供的配置文件名通常为`pgbouncer.ini`，供`pgbouncer`命令使用。该配置文件还能提供Greenplum数据库位置信息。`pgbouncer.ini`配置文件也定义了线程数、连接池、授权用户和授权配置信息，另外还有一些其他信息。

默认情况下，`pgbouncer`作为一个前端进程运行。可以通过启动`pgbouncer`时指定`-d`选项来让其在后台执行。

`pgbouncer`进程被操作系统用户拥有。在启动`pgbouncer`时可以指定一个不同的用户名。

PgBouncer包括一个类似`psql`的管理控制台。授权用户可以连接到虚

gprecoverseg

拟数据库来监控和管理PgBouncer。也可以通过管理控制台来监控和管理后台进程。也可以使用控制台更新和重载PgBouncer配置文件，此时不需要通知或重启该进程。

有关PgBouncer的更多信息，请参考 [PgBouncer FAQ](#) □。

选项

-d | --daemon

运行PgBouncer作为守护进程（后台进程）。默认是作为前台进程运行。

作为守护进程启动时，PgBouncer显示启动消息到stdout。要禁止显示消息，请添加-q选项。

要关闭PgBouncer守护程序，请登录管理控制台并发出SHUTDOWN命令。

-R | --restart

使用指定的命令行参数重新启动PgBouncer。在重新启动期间，维护与数据库的非TLS连接，TLS连接被丢弃。

要作为守护程序重新启动PgBouncer，请指定选项-Rd。

Note: 仅在操作系统支持Unix套接字且PgBouncer配置没有 unix_socket_dir的情况下有效。

-q | --quiet

安静地运行，不在命令行（stdout）上显示消息。

-v | --verbose

增加消息详细度。显示额外的消息。可以多次使用。

{-u | --user}=username

PgBouncer进程假定的username的身份。

-V | --version

显示版本并退出。

-h | --help

显示帮助信息并退出。

另见

[pgbouncer.ini](#)、[pgbouncer-admin](#)

Greenplum数据库® 6.0文档

 工具指南 管理工具参考

```
[databases]
db = ...

[pgbouncer]
...
...

[users]
...
```

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

aprecoversea

PgBouncer的配置文件。

概要

```
[databases]
db = ...

[pgbouncer]
...
...

[users]
...
```

描述

可以在配置文件`pgbouncer.ini`中指定PgBouncer配置参数并指定用户相关的配置参数。

PgBouncer的配置文件（通常是`pgbouncer.ini`）是`.ini`格式。`.ini`格式的文件通常由小节、参数和参数值组成。小节名称封闭在方括号中，例如，`[section_name]`。参数和值通常以`key=value`的方式展示。以`";"`或`"#"`开头的行被注释或者忽略。当该`";"`和`"#"`字符出现在行后面时候不被识别。

PgBouncer配置文件包含`%include`路由符，它指定了用来读取和处理的其他文件。这让我们可以将配置文件拆分为多个部分。例如：

```
%include filename
```

如果提供的文件名不是绝对路径，当前操作系统工作路径被作为相对路径。

PgBouncer的配置文件包括以下小节，详细信息如下：

- [\[databases\]小节](#)
- [\[pgbouncer\]小节](#)
- [\[users\]小节](#)



[databases]小节

`[databases]`小节包含`key=value`键值对，其中键（`key`）是数据库名称，值（`value`）是`key=value`对的libpq连接字符串列表。

数据库名称可以包含不带引号的字符`[0-9A-Za-z_.-]`。包含其他字符的名称必须使用标准SQL标识符引用：

- 名字用双引号括起来（`" "`）。
- 用两个连续的双引号字符在一个标识符内表示一个双引号。

数据库名*是回退数据库。此键的值是所请求数据库的连接字符串。如果自动创建的数据项的空闲时间超过`autodb_idle_timeout`参数中指定的时间，则会自动清理这些数据项。

数据库连接参数

该值中可能包含以下参数以指定数据库的位置。

dbname
目标数据库名称。
默认值：与客户端数据库名称相同。

host
Greenplum的Master主机的名称或IP地址。主机名在连接时被解析。如果DNS返回多个结果，则以循环方式使用它们。DNS结果会被缓存，并且dns_max_ttl参数确定缓存项何时过期。
默认值：未设置，表示通过Unix套接字进行连接。

port
Greenplum数据库的Master端口。
默认：5432

user, password
如果设置了user=，则用指定的用户建立到目标数据库的所有连接。这意味着数据库将只有一个池。
如果未设置user=参数，将尝试使用客户端传递的用户名登录到目标数据库。这意味着每个连接到数据库的用户都会有一个池。

auth_user
如果设置了auth_user，任何未在auth_file文件中指定的用户，通过以auth_user身份查询数据库中的pg_shadow表来认证。auth_user的口令必须在auth_file中设置。

client_encoding
从服务器要求特定的client_encoding。

datestyle
从服务器要求特定的datestyle。

timezone
从服务器要求特定的timezone。

连接池配置

可以使用以下数据库连接池配置

pool_size
设置该数据库的最大池数。如果没有设置，则使用default_pool_size。

connect_query
在任何用户连接建立之后，允许执行查询。如果查询引发错误，则会被记录，否则将被忽略。

pool_mode
为此数据库设置池模式。如果未设置，则使用默认的pool_mode。

max_db_connections
为此数据库设置数据库范围内的最大连接数。此数据库的所有池连接总数不会超过此值。

[pgbouncer]小节

一般设置

logfile
指定日志文件的位置。日志文件保持打开状态。日志轮转后在PgBouncer管理控制台中执行kill -HUP pgbouncer或运行RELOAD;命令。

默认：未设置。

pidfile

pid文件的名字。没有pid文件，PgBouncer不能作为后台进程（守护进程）运行。

默认：未设置。

listen_addr

PgBouncer侦听TCP连接的接口地址列表。用户也可以使用*，表示在所有接口上进行监听。如果没有设置，则仅允许Unix套接字连接。

地址可以用数字（IPv4/IPv6）或名称来指定。

默认：未设置。

listen_port

要监听的端口。应用到TCP和UNIX套接字。

默认：6432

unix_socket_dir

指定Unix套接字的位置。适用于监听套接字和服务器连接。如果设置为空字符串，则禁用Unix套接字。在线重新启动（-R选项）才能正常工作。

默认值：/tmp

unix_socket_mode

Unix套接字的文件系统模式。

默认为：0777

unix_socket_group

用于Unix套接字的组名。

默认：未设置。

user

如果设置，则指定启动后要更改的Unix用户。这只有在PgBouncer作为root启动或user与当前用户相同时才有效。

默认：未设置。

auth_file

包含要加载的用户名和口令的文件的名称。文件格式与Greenplum数据库的pg_auth文件相同，因此可以将此参数设置为其中一个后端文件。

默认：未设置

auth_hba_file

当auth_type为hba时使用的HBA配置文件。HBA文件格式请参阅<https://pgbouncer.github.io/config.html#hba-file-format> PgBouncer文档中PgBouncer支持的HBA授权文件格式的讨论部分。

默认：未设置

auth_type

如何认证用户

pam

使用PAM作为客户授权。auth_file被忽略。该方法与auth_user选项一起使用。报告给PAM的服务名为“pgbouncer”。PAM在HBA配置文件中不支持。

hba

实际授权类型从auth_hba_file文件中加载。该设置允许不同的访问方法访问不同的路径。

cert

客户端必须使用有效的客户端证书连接到TLS。客户端的用户名取自证书中的CommonName字段。

md5

使用基于MD5的口令检查。`auth_file`可能包含MD5加密或纯文本的口令。这是默认的身份认证方法。

plain

通过网络发送明文口令。已废弃。

trust

不做认证。用户名仍然必须存在于`auth_file`中。

any

像`trust`方法一样，但提供的用户名被忽略。要求将所有数据库配置为使用特定用户登录。另外，控制台数据库允许任何用户以`admin`登录。

auth_query

从数据库加载用户口令的查询。如果用户不存在于`auth_file`中并且数据库项包含`auth_user`，则该查询作为`auth_user`在数据库中运行以查找用户。

注意查询会在每个数据库中都执行，如果要运行函数，那么该函数需要在每个数据库中都安装。

默认值: `SELECT username, passwd FROM pg_shadow WHERE username=$1`

auth_user

如果设置了`auth_user`，任何未在`auth_file`文件中指定的用户，通过以`auth_user`身份查询数据库中的`pg_shadow`表来认证。`auth_user`的口令必须在`auth_file`中设置。

直接访问`pg_shadow`要求有Greenplum数据库的管理员权限。推荐使用非管理员用户并调用函数来代替。

pool_mode

指定服务器连接何时可以被其他客户端重用。

session

当客户端断开连接时，连接返回到池。默认。

transaction

事务结束后，连接返回到池。

statement

当前查询完成后，连接返回到池。具有多个语句的长事务在此模式下被禁止。

max_client_conn

允许的最大客户端连接数。当增加时，文件描述符限制也应该增加。使用的文件描述符的实际数量大于`max_client_conn`。当每个用户使用自己的用户名连接到服务器时，使用的理论最大值是：

```
max_client_conn + (max_pool_size * total_databases * total_users)
```

如果在连接字符串中指定了数据库用户，则所有用户使用相同的用户名连接。那么理论上的最大连接是：

```
max_client_conn + (max_pool_size * total_databases)
```

(应该永远不要达到理论上的最大值，除非有人刻意为它加上负荷。但是，这意味着用户应该将文件描述符的数量设置为一个安全的较高数字。在用户的操作系统文档中搜索`ulimit`。)

默认：100

default_pool_size

允许每个用户/数据库对的服务器连接数量。这可以在每个数据库配置中被覆盖。

默认: 20

min_pool_size

当池中连接数低于这个数字时，将更多的服务器连接添加到池中。当平常的负载下降然后在一段时间的不活跃期之后突然恢复到平常的负载时，这会改善行为。

默认值: 0 (禁用)

reserve_pool_size

对池允许的额外连接的数量。0 禁用。

默认: 0 (禁用)

reserve_pool_timeout

如果客户端在这段时间内没有被服务，PgBouncer允许使用来自保留池的额外连接。0禁用。

默认值: 5.0

max_db_connections

每个数据库的最大连接数。如果达到限制，关闭到池的客户端连接不会立即允许对另一个池建立服务器连接，因为第一个池的服务器连接仍处于打开状态。一旦服务器连接关闭（由于空闲超时），将为等待池打开一个新的服务器连接。

默认: 没限制

max_user_connections

每个用户的最大连接数。当用户达到限制时，关闭到池的客户端连接不会立即允许对另一个池建立服务器连接，因为第一个池的服务器连接仍处于打开状态。一旦服务器连接关闭（由于空闲超时），将为等待池打开一个新的服务器连接。

server_round_robin

默认情况下，PgBouncer以LIFO（后进先出）顺序重用服务器连接，这样少数连接获得最大的负载。这在单个服务器提供数据库时提供最佳性能。但是如果数据库IP后面有TCP循环，那么最好是PgBouncer也以这种方式使用连接来实现均衡的负载。

默认值: 0

ignore_startup_parameters

默认情况下，PgBouncer只允许它在启动包中可以跟踪的参数：
client_encoding、datestyle、timezone和
standard_conforming_strings。

所有其他参数都会产生错误。要允许其他参数，在这里指定它们，这样PgBouncer可以忽略它们。

默认值: 空

disable_pqexec

禁用简单查询协议（PQexec）。与扩展查询协议不同，简单查询协议允许在一个数据包中进行多个查询，这允许某些种类的SQL注入攻击。禁用它可以提高安全性。这意味着只有专门使用扩展查询协议的客户端才能工作。

默认值: 0

application_name_add_host

将客户端主机地址和端口添加到连接启动时设置的应用程序名称设置。这有助于识别不良查询的来源。如果应用程序在连接后执行了SET application_name，则设置将被覆盖而不被检测到。

默认值: 1

conffile

显示当前配置文件位置。改变该参数会导致PgBouncer在下次RELOAD/SIGHUP后使用另外的配置文件。

默认值：文件来自命令行

service_name

仅在win32服务注册时使用。

默认值：pgbouncer

job_name

service_name的代称。

日志设置

syslog

切换syslog开启和关闭。

默认值：0

syslog_ident

以什么名字发送日志到syslog。

默认值：pgbouncer

syslog_facility

在什么设施下发送日志到系统日志。一些可能是：
auth、authpriv、daemon、user、local0-7

默认值：daemon

log_connections

记录成功的登陆。

默认值：1

log_disconnections

记录连接失败的原因。

默认值：1

log_pooler_errors

记录池中发送给客户端的错误消息。

默认值：1

stats_period

将汇总的统计信息写入日志的频率。

默认值：60

控制台访问控制

admin_users

允许连接并在控制台上运行所有命令的数据库用户列表，用逗号分隔。
当auth_type=any时候，忽略该列表。在这种情况下，任何用户名都被允许作为admin。

默认值：空

stats_users

逗号分隔的数据库用户列表，允许它们在控制台上连接并运行只读查询。这意味着除了SHOW FDS之外的所有SHOW命令。

默认值：空

C连接健康检查、超时

server_reset_query

在连接释放时且没有对其他客户端可用之前，发送到服务器的查询。在那一刻没有事务正在进行，所以它不应该包括ABORT或ROLLBACK。

该查询会清理任何改变模式为数据库会话，以保证下一个客户端连接获取更好的状态。默认为DISCARD ALL

当使用事务池时，`server_reset_query`应该是空的，因为客户端不能使用任何事务特性。如果客户端使用了事务特性，操作将会失败，因为事务池不能保证下次运行相同的连接。

默认值：DISCARD ALL;

server_reset_query_always

`server_reset_query`是否应该在所有池模式下运行。如果此设置处于关闭状态（默认），则`server_reset_query`将仅在处于会话池模式的池中运行。
事务池模式下的连接不应该有任何重置查询的需要。

默认值：0

server_check_delay

保持被释放连接可重用而不运行健康检查查询的时间。如果为0，则查询始终运行。

默认值：30.0

server_check_query

用来测试服务器连接的一个简单的无所作为的查询。

如果是一个空字符串，则禁用健康检查。

默认值：SELECT 1;

server_lifetime

池会尝试关闭连接时间超过这个秒数的服务器连接。将其设置为0意味着连接只能使用一次，然后关闭。

默认值：3600.0

server_idle_timeout

如果服务器连接闲置超过这么多秒钟，它将被丢弃。如果此参数设置为0，则禁用超时。

默认值：600.0

server_connect_timeout

如果在此秒数内连接和登录无法完成，则连接将被关闭。

默认值：15.0

server_login_retry

如果登陆由于connect()或认证失败而失败，则在重新尝试连接之前，池会等待几秒钟。

默认值：15.0

client_login_timeout

如果客户端在此秒数内连接但无法登录，则会断开连接。需要这来避免死连接卡住SUSPEND，从而导致在线重新启动。

默认值: 60.0

autodb_idle_timeout

如果（通过*）自动创建的数据库池已经有这么多秒没有使用，则它们将被释放。并且也放弃对它们的统计。

默认值: 3600.0

dns_max_ttl

缓存DNS查找结果多少秒。如果一个DNS查找返回几个答案，那么PgBouncer会在它们之间循环往复。实际的DNS TTL被忽略。

默认值: 15.0

dns_nxdomain_ttl

可以缓存错误和NXDOMAIN DNS查找多长时间。以秒为单位。

默认值: 15.0

dns_zone_check_period

检查区域序列号是否改变的周期。

PgBouncer可以从主机名（第一个点之后的所有内容）收集DNS区域，然后定期检查区域序列号是否更改。如果检测到更改，则会再次查找该区域中的所有主机名。如果任何主机IP发生更改，则其连接将失效。

只适用于UDNS后端（用--with-udns或--with-cares配置）。

默认值: 0.0 (禁用)

TLS设置

client_tls_sslmode

TLS模式主要给客户端连接使用。TLS连接默认不启用。当启用时，必须同时配置 client_tls_key_file 和 client_tls_cert_file 参数，以使PgBouncer能够接受客户端连接。

- disable: plain TCP。如果客户端请求TLS。他会被忽略。默认配置。
- allow: 如果客户端请求TLS，他被使用。如果没请求，使用plain TCP。如果客户端使用client-certificate，不会验证改值。
- prefer: 与allow相同
- require: 客户端必须使用TLS。如果不使用，客户端连接被拒绝。如果客户端使用client-certificate，他不被验证。
- verify-ca: 客户端必须使用TLS并验证客户端授权。
- verify-full: 与verify-ca相同。

client_tls_key_file

PgBouncer使用私有密钥接受客户端连接

默认值: 不设置。

client_tls_cert_file

根证书授权验证客户端授权。

默认值: 未设置。

client_tls_ca_file

根证书授权验证客户端授权。

默认值: 未设置。

client_tls_protocols

那种TLS协议被允许。

有效值为: tlsv1.0、tlsv1.1、tlsv1.2。

快捷方

式: all (tlsv1.0、tlsv1.1、tlsv1.2)、secure (tlsv1.2)、legacy (all)。

默认值: all

client_tls_ciphers

默认值: fast

client_tls_ecdhcurve

ECDH key交换使用的Elliptic Curve名称。

可用值: none (DH禁用)、auto (256-bit ECDH)、curve name。

默认值: auto

client_tls_dheparams

DHE key交换类型。

可用值: none (DH禁用)、auto (2048-bit ECDH)、legacy (1024-bit ECDH)。

默认值: auto

server_tls_sslmode

连接到Greenplum数据库和PostgreSQL数据库的TLS模式。TLS连接默认被禁用。

- disable: plain TCP。如果客户端请求TLS。他会被忽略。默认配置。

- allow: 如果客户端请求TLS，他被使用。如果没请求，使用plain TCP。如果客户端使用client-certificate，不会验证改值。

- prefer: 与allow相同

- require: 客户端必须使用TLS。如果不使用，客户端连接被拒绝。如果客户端使用client-certificate，他不被验证。

- verify-ca: 客户端必须使用TLS并验证客户端授权。

- verify-full: 与verify-ca相同。

server_tls_ca_file

用于验证Greenplum和PostgreSQL数据库服务证书的证书验证文件。

默认值: 不设置。

server_tls_key_file

依赖于Greenplum或PostgreSQL的PgBouncer私有密钥。

默认值: 不设置。

server_tls_cert_file

私有密钥证书。Greenplum或PostgreSQL数据库会验证它。

默认值: 不设置。

server_tls_protocols

允许使用的TLS协议版本。

有效值为: tlsv1.0、tlsv1.1、tlsv1.2。

快捷方

式: all (tlsv1.0、tlsv1.1、tlsv1.2)、secure (tlsv1.2)、legacy (all)。

server_tls_ciphers

默认值: fast

危险超时

设置以下超时可能会导致意外的错误。

query_timeout

运行时间超过这个秒数的查询被取消。此参数只能与较小的服务器端
`statement_timeout`一起使用，以捕获出现网络问题的查询。

默认值: 0.0 (禁用)

query_wait_timeout

允许查询等待执行的最长时间 (以秒为单位)。如果在此期间查询未分配连接，则客户端将断开连接。这用于防止无响应的服务器强占连接。

默认值: 0.0 (禁用)

client_idle_timeout

客户端连接空闲的时间超过了这么多秒钟就会被关闭。这应该大于客户端连接的生命周期设置，并且仅用于网络问题。

默认值: 0.0 (禁用)

idle_transaction_timeout

如果客户端处于“事务空闲”状态的时间超过这个秒数，则它被断开。

默认值: 0.0 (禁用)

低层网络设置

pkt_buf

包的内部缓冲区大小。影响发送的TCP包的尺寸和一般内存使用情况。实际的libpq包可能比这个大，所以没有必要把它设置的很大。

默认值: 4096

max_packet_size

PgBouncer接受的包的最大尺寸。包是一个查询或一个结果集行。完整的结果集可以更大。

默认值: 2147483647

listen_backlog

`listen(2)`系统调用的backlog参数。它表示有多少新的未答复的连接尝试被保留在队列中。当队列已满时，进一步的新连接尝试被丢弃。

默认值: 128

sbuf_looptcnt

在继续之前，在一个连接上处理数据多少次。没有这个限制，一个大的结果集的连接可能会使 PgBouncer停顿很长时间。一个循环处理一个`pkt_buf`数量的数据。0表示没有限制。

默认值: 5

suspend_timeout

等待SUSPEND或reboot (-R)期间的缓冲区刷写多少秒钟。如果刷写不成功，连接将被丢弃。

默认值: 10

tcp_defer_accept

有关此选项和其他TCP选项的详细信息，请参阅tcp (7) 手册页。

默认值: Linux上是45，其他为0

tcp_socket_buffer

默认值: 不设置

tcp_keepalive
用OS默认值启用基本的keepalive。

Linux上，系统默认为 `tcp_keepidle=7200`、`tcp_keepintvl=75`、`tcp_keepcnt=9`。

默认值：1

tcp_keepcnt

默认值：未设置

tcp_keepidle

默认值：未设置

tcp_keepintvl

默认值：未设置

[users]小节

这部分包含`key=value`对，其中`key`是用户名，`value`是`key=value`对的libpq连接字符串列表。

池配置

pool_mode

设置池模式用于来自此用户的所有连接。如果未设置，则使用数据库或默认`pool_mode`。

示例配置文件

最小配置

```
[databases]
postgres = host=127.0.0.1 dbname=postgres auth_user=gpadmin

[pgbouncer]
pool_mode = session
listen_port = 6543
listen_addr = 127.0.0.1
auth_type = md5
auth_file = users.txt
logfile = pgbouncer.log
pidfile = pgbouncer.pid
admin_users = someuser
stats_users = stat_collector
```

使用客户端传递的连接参数：

```
[databases]
* =

[pgbouncer]
listen_port = 6543
listen_addr = 0.0.0.0
auth_type = trust
auth_file = bouncer/users.txt
logfile = pgbouncer.log
pidfile = pgbouncer.pid
ignore_startup_parameters=options
```

数据库默认

```
[databases]

; foodb over unix socket
foodb =

; redirect bardb to bazdb on localhost
bardb = host=127.0.0.1 dbname=bazdb

; access to destination database will go with single user
forcedb = host=127.0.0.1 port=300 user=bar password=foo
client_encoding=UNICODE datestyle=ISO
```

另见

[pgbouncer](#)、[pgbouncer-admin](#)、[PgBouncer配置页面](#) ▾

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon_install

gppkg

PgBouncer管理控制台命令

概要

```
psql -p port pgbouncer
```

描述

PgBouncer管理控制台可以通过psql操作。通过连接到PgBouncer *port* 和虚拟数据库名pgbouncer来登陆控制台。

列在pgbouncer.ini配置文件中的配置参数admin_users和stats_users的用户有访问PgBouncer管理控制台的权限。

可以从控制台控制PgBouncer和Greenplum数据库之间的连接。也可以进行PgBouncer配置参数设置。

选项

-p port
PgBouncer端口号。

命令行语法

```
pgbouncer=# SHOW help;
NOTICE:  Console usage
DETAIL:
      SHOW
      HELP | CONFIG | DATABASES | POOLS | CLIENTS | SERVERS | VERSION
      SHOW FDS | SOCKETS | ACTIVE_SOCKETS | LISTS | MEM
      SHOW DNS_HOSTS | DNS_ZONES
      SHOW STATS | STATS_TOTALS | STATS_AVERAGES
      SET key = arg
      RELOAD
      PAUSE [ <db> ]
      RESUME [ <db> ]
```



gprecoverseg

```
DISABLE <db>
ENABLE <db>
KILL <db>
SUSPEND
SHUTDOWN
```

管理命令

以下PgBouncer管理命令控制pgbouncer进程。

PAUSE [*db*]

如果没有指定数据库，PgBouncer将尝试从所有服务器断开连接，首先等待所有查询完成。在所有查询完成之前，该命令不会返回。这个命令是用来准备重启数据库的。

如果指定了数据库名称，则只有该数据库被暂停。

如果运行一个PAUSE *db*数据库命令，然后运行PAUSE命令暂停所有数据库，则必须执行两个RESUME命令，一个用于所有数据库，一个用于指定的数据库。

SUSPEND

所有套接字缓冲区都被刷新，PgBouncer停止监听数据。在所有缓冲区为空之前，该命令不会返回。在线重新启动PgBouncer时使用。

RESUME [*db*]

从之前的PAUSE或者SUSPEND命令恢复工作。

如果用PAUSE命令指定了数据库，则还必须使用RESUME命令指定数据库。

使用PAUSE命令暂停所有数据库之后，不支持使用RESUME *db*恢复单个数据库。

DISABLE *db*

拒绝数据库上的所有新客户端连接。

ENABLE *db*

在数据库上允许新的客户端连接。

KILL *db*

立即删除所有的客户端和服务器到指定数据库的连接。

SHUTDOWN

停止PgBouncer进程。从psql命令会话中退出，请输入\q。

RELOAD

T PgBouncer进程重新加载当前的配置文件并更新可更改的设置。

SET *key*=*value*

覆盖指定的配置设置。请参见[SHOW CONFIG](#);命令。

SHOW命令

`SHOW category`命令显示不同类型的PgBouncer信息。 用户可以指定以下类别之一：

- [ACTIVE_SOCKETS](#)
- [CLIENTS](#)
- [CONFIG](#)
- [DATABASES](#)
- [DNS_HOSTS](#)
- [DNS_ZONES](#)
- [FDS](#)
- [POOLS](#)
- [SERVERS](#)
- [SOCKETS](#)
- [STATS](#)
- [STATS_TOTALS](#)
- [STATS_AVERAGES](#)
- [LISTS](#)
- [MEM](#)
- [USERS](#)
- [VERSION](#)

ACTIVE_SOCKETS

Table 1. 活动套接字信息

列	描述
type	S为服务器， C为客户端。
user	pgbouncer使用该用户名连接到服务器。
database	数据库名字。
state	服务器连接的状态，处于active状态、used状态或idle状态。
addr	PostgreSQL服务器的IP地址。
port	PostgreSQL 服务器端口。

local_addr	本地机器上的连接起始地址。
local_port	本地机器上的连接起始端口。
connect_time	连接建立的时间。
request_time	最后一次发出请求的时间。
wait	等待时间。
wait_us	等待时间。 (微秒)
ptr	该连接的内部对象的地址。用作唯一的ID。
link	服务器和客户端匹配的地址。
remote_pid	后台服务进程的定位符。
tls	TLS文本。
recv_pos	在I/O缓冲区中的接收位置。
pkt_pos	在I/O 缓冲区中的解析位置。
pkt_remain	套接字上剩余的包数量。
send_pos	包中的发送位置。
send_remain	剩余要发送的包的总长度。
pkt_avail	剩余要解析的I/O缓冲区量。
send_avail	剩余要发送的I/O缓冲区量。

CLIENTS

Table 2. 客户端

列	描述
type	C为客户端。
user	客户端连接的用户。
database	数据库名字。
state	客户端连接状态，状态可以为active、used、waiting或idle。
addr	客户端的IP地址，或套接字连接的unix域。
port	客户端连接的端口。
local_addr	本地机器上的连接结束地址。
local_port	本地机器上的连接结束端口。
connect_time	连接时间的时间戳。

request_time	上次客户端请求的时间戳。
wait	等待时间。
wait_us	等待时间。 (微秒)
ptr	该连接的内部对象的地址。用作唯一的ID。
link	客户端与服务器连接的地址。
remote_pid	进程ID，如果客户端连接到Unix套接字，并且操作系统支持获取它。
tls	客户端TLS内容。

CONFIG

当前PgBouncer参数设置的列表。

Table 3. 配置

列	描述
key	配置变量名
value	配置值
changeable	yes或no。显示变量在运行时是否可以改变。 如果为no，则该变量只能在启动时改变。

DATABASES

Table 4. 数据库

列	描述
name	配置好的数据库项的名字
host	pgbouncer连接到的主机。
port	pgbouncer连接到的端口
database	pgbouncer连接到的实际数据库名称。
force_user	当用户是连接字符串的一部分时，pgbouncer和数据库服务器之间的连接被强制为给定的用户，无论用什么客户端用户都是这样。
pool_size	服务器连接的最大数量。

reserve_pool	如果池达到pool_size，可以创建的额外连接数。
pool_mode	数据库的覆盖pool_mode，或者NULL使用默认值。
max_connections	此数据库的所有池的最大连接数。
current_connections	此数据库的所有池的连接总数。
paused	数据库的停顿/非停顿状态。
disabled	数据库的启用/停用状态。

DNS_HOSTS

Table 5. DNS主机

列	描述
hostname	主机名
ttl	下次查询间隔多少秒。
addrs	逗号分隔的地址列表。

DNS_ZONES

Table 6. 缓冲中的DNS区域

列	描述
zonename	区域名字
serial	当前DNS序列号
count	属于该区域的主机名。

FDS

SHOW FDS是用于联机重启的内部命令，例如升级到新的PgBouncer版本时。它显示正在使用的文件描述符列表以及附加其上的内部状态。这个命令阻塞内部事件循环，因此在PgBouncer被使用的时候不应该使用它。

当连接的用户拥有用户名“pgbouncer”，通过一个Unix套接字连接，并

具有与正在运行的 进程相同的UID时，实际的文件描述符将通过连接传递。

Table 7. FDS

列	描述
fd	文件描述符的数值。
task	任务为以下之一pooler、client或server。
user	使用文件描述符的连接的用户。
database	使用文件描述符的连接的数据库。
addr	使用文件描述符的连接的IP地址，如果使用Unix套接字，则为“unix”。
port	使用文件描述符的连接使用的端口。
cancel	该连接的取消键。
link	相应的服务器/客户端的文件描述符。空闲时为NULL。
client_encoding	数据库使用的字符集。
std_strings	这将控制普通字符串 ('...') 是否按照SQL标准中的规定按字面意思处理反斜线。
datestyle	显示日期和时间值的格式。
timezone	用于解释和显示时间戳的时区。
password	auth_user的密码。

LISTS

在两列中显示以下PgBouncer统计信息：项标签和值。

Table 8. PgBouncer项的计数

条目	描述
databases	数据库的计数。
users	用户的计数。
pools	池的计数。
free_clients	空闲客户端的计数。
used_clients	在用客户端的计数。
login_clients	处于login状态的客户端计数。

free_servers	空闲服务器的计数。
used_servers	在用服务器的计数。
dns_names	DNS名字的计数。
dns_zones	DNS区域的计数。
dns_queries	DNS查询的计数。
dns_pending	正在进行的DNS查询的计数。

MEM

显示这些PgBouncer缓存的缓存内存信息：

- user_cache
- db_cache
- pool_cache
- server_cache
- client_cache
- iobuf_cache

Table 9. 内存缓存

列	描述
name	缓存的名字。
size	缓存中单个缓存槽的大小。
used	缓存中在用的插槽数量。
free	缓存中可用插槽的数量。
memtotal	缓存使用的总字节数。

POOLS

为每对（数据库，用户）创建一个新的池项。

Table 10. 池

列	描述
database	数据库名字。
user	用户名。

cl_active	链接到服务器并且可以处理查询的客户端连接。
cl_waiting	已发送查询但尚未获得服务器连接的客户端连接。
sv_active	链接到客户端的服务器连接。
sv_idle	还没有使用且可立即用于客户端查询的服务器连接。
sv_used	空闲时间已经超过server_check_delay的服务器连接。在它们被使用前，必须先运行server_check_query。
sv_tested	当前正在运行server_reset_query或server_check_query的服务器连接。
sv_login	目前正在登陆的服务器连接。
maxwait	队列中第一个（最老的）客户端等待了多长时间，以秒为单位。如果这个开始增加，那么当前的服务器池不能足够快地处理请求。原因可能是服务器超载或pool_size设置太小。
maxwait_us	max_wait（微秒）。
pool_mode	正在使用的池模式。

SERVERS

Table 11. 服务器

列	描述
type	S, 为服务器。
user	pgbouncer用来连接到服务器的用户ID。
database	数据库名。
state	pgbouncer服务器连接的状态，可以是active、used或idle。
addr	Greenplum或PostgreSQL服务器的IP地址。
port	Greenplum或PostgreSQL服务器的端口。
local_addr	本地机器上的连接开始地址。
local_port	本地机器上的连接开始端口。
connect_time	连接建立的时间。
request_time	最后一次请求发送的时间。

wait	等待时间。
wait_us	等待时间。 (微秒)
ptr	此连接的内部对象的地址。用作唯一的ID。
link	与服务器配对的客户端连接的地址。
remote_pid	后端服务器进程的PID。如果连接是通过Unix套接字进行的，并且操作系统支持获取进程ID信息，则它是OS的pid。否则，它将从服务器发送的取消包中提取出来，在服务器是PostgreSQL的情况下应该是PID，但是如果服务器是另一个PgBouncer，则它是一个随机数。
tls	TLS内容。

STATS

显示统计数据。

Table 12. 统计

列	描述
database	为每个数据库呈现的统计信息。
total_xact_count	PgBouncer池化的SQL事务总数。
total_query_count	PgBouncer池化的SQL查询总数。
total_received	pgbouncer收到的网络流量的总字节数。
total_sent	pgbouncer发送的网络流量的总字节数。
total_xact_time	一个事务中，通过PgBouncer连接到Greenplum数据库的微秒数，包括idle事务和执行查询。
total_query_time	当主动连接到数据库服务器时，pgbouncer花费的总微秒数。
total_wait_time	客户端等待服务器的时间（微秒）。
avg_xact_count	PgBouncer池化的SQL事务平均数量。
avg_query_count	最后一个统计周期的每秒平均查询数。
avg_recv	每秒平均收到（来自客户端）的字节数。
avg_sent	每秒平均发送（到客户端）的字节数。
avg_xact_time	平均事务时长（微秒）。

avg_query_time	平均查询时长（微秒）。
avg_wait_time	平均等待时长（微秒）。

STATS_AVERAGES

SHOW STATS的子集，展示已选择的统计信息的平均值。

STATS_TOTALS

SHOW STATS的子集，展示已选择的统计信息的总值。

USERS

Table 13. 用户

列	描述
name	用户名。
pool_mode	用户的覆盖pool_mode，如果将使用默认值，则为NULL。

VERSION

显示PgBouncer版本的信息

Note: 该参考文档基于PgBouncer 1.8.1文档。

另见

[pgbouncer](#)、[pgbouncer.ini](#)

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

[PXF介绍](#)

[PXF管理手册](#)

[使用PXF访问hadoop](#)

[使用PXF访问Azure, Google云端存储, Minio和S3对象存储](#)

使用PXF访问SQL数据库
(JDBC)

PXF故障排除

[PXF实用程序手册](#)

`pxf cluster`

`pxf`

[Back to the Administrator Guide](#)

[Greenplum database 5管理员指南](#)

[Greenplum database 4管理员指南](#)

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

In this topic:

- [概要](#)
- [描述](#)
- [命令](#)
- [选项](#)
- [示例](#)
- [See Also](#)

在本地Greenplum数据库主机上管理PXF配置和PXF服务实例。

概要

`pxf <command> [<option>]`

<command> 支持的选项:

`cluster`
`help`
`init`
`reset`
`restart`
`start`
`status`
`stop`
`sync`
`version`

描述

`pxf` 工具管理本地Greenplum数据库主机上的PXF配置和PXF服务实例。

您可以在master, standby或特定segment主机上初始化或重置PXF。您还可以将PXF配置从master服务器同步到这些主机。

您可以在特定的segment主机上启动, 停止或重新启动PXF服务实例, 或者显示在segment主机上运行的PXF服务实例的状态。

(Use the `pxf cluster` command to initialize or reset PXF on all hosts, synchronize the PXF configuration to the Greenplum Database cluster, or to start, stop, or display the status of the PXF service instance on all segment hosts in the cluster.) (使用 `pxf cluster` 命令在所有主机上初始化或重置PXF，将PXF配置同步到Greenplum数据库集群，或者启动，停止或显示集群中所有segment主机上的PXF服务实例的状态。)

命令

cluster

在所有Greenplum数据库主机上管理PXF配置和PXF服务实例。
请参阅 [pxf群集](#)。

help

显示pxf帮助消息，然后退出

init

在主机上初始化PXF服务实例。初始化PXF时，必须通过名为 `$PXF_CONF` 的环境变量来标识PXF用户配置目录。如果在初始化PXF之前未设置 `$PXF_CONF`，则PXF会提示您在初始化过程中接受或拒绝默认用户配置目录 `$HOME/pxf`。参阅 [Options](#)。

reset

重置主机上运行的PXF服务实例。重置将删除PXF运行时文件和目录，并使PXF返回未初始化状态。在主机上重置PXF之前，必须停止在segment主机上运行的PXF服务实例。

restart

在segment主机上重新启动PXF服务

start

在segment主机上启动PXF服务

status

显示在segment主机上运行的PXF服务实例的状态

stop

停止在segment主机上运行的PXF服务实例

sync

将PXF配置从master服务器同步到特定的Greenplum数据
库standby服务器或segment主机。您必须在master主机上运行
`pxf sync`。请参阅 [选项](#)。

version

显示PXF的版本信息并退出

选项

`pxf init` 命令支持以下选项:

-y

如果未设置环境变量，则不提示使用默认的 `$PXF_CONF` 目录位置。

`pxf reset` 命令采用以下选项:

-f | -force

重置PXF服务实例之前不要提示；无需用户干预即可重置。

在Greenplum数据库master主机上运行的 `pxf sync` 命令具有以下选项:

<gphost>

将PXF配置同步到的Greenplum数据库主机。`<gphost>` 必须标识standby主机或segment主机。

示例

在segment主机本地启动PXF实例:

```
$ $GPHOME/pxf/bin/pxf start
```

See Also

[pxf cluster](#)

- Greenplum数据库® 6.0文档
- Greenplum平台扩展框架(PXF)
- [PXF介绍](#)
- [PXF管理手册](#)
- [使用PXF访问hadoop](#)
- [使用PXF访问Azure, Google云端存储, Minio和S3对象存储](#)
- [使用PXF访问SQL数据库\(JDBC\)](#)
- [PXF故障排除](#)
- [PXF实用程序手册](#)

In this topic:

- [概要](#)
- [描述](#)
- [Commands](#)
- [举例](#)
- [See Also](#)

在所有Greenplum数据库主机上管理PXF配置和PXF服务实例。

概要

```
pxf cluster <command>
```

where `<command>` is:

help
init
reset
start
status
stop
sync

描述

`pxf cluster` 工具命令在master,standby服务器以及所有Greenplum数据库segment主机上管理PXF。您可以使用该工具执行以下操作：

- 在Greenplum数据库集群中的所有主机上初始化PXF配置。
- 将所有主机上的PXF服务实例重置为其未初始化状态。
- 在所有segment主机上启动和停止PXF服务实例。
- 显示所有segment主机上的PXF服务实例的状态。
- 将PXF配置从Greenplum数据库master主机同步到standby和所有segment主机。

[pxf cluster](#)

[pxf](#)

[Back to the Administrator Guide](#)

[Greenplum database 5管理员指南](#)

[Greenplum database 4管理员指南](#)

[FAQ](#)

[如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息](#)

`pxf cluster` 需要一个正在运行的数据库集群。您必须在Greenplum数据库master主机上运行该程序。

(如果要在特定segment主机上管理PXF服务实例，请使用pxf工具。请参见 [pxf](#)。)

Commands

help

显示 `pxf cluster` 帮助消息，然后退出

init

在master,standby节点和所有segment主机上初始化PXF服务实例。在Greenplum数据库集群中初始化PXF时，必须通过名为 `$PXF_CONF` 的环境变量来标识PXF用户配置目录。如果在初始化PXF之前未设置 `$PXF_CONF`，则PXF返回错误。

reset

在master, standby和所有segment主机上重置PXF服务实例。重置将删除PXF运行时文件和目录，并使PXF返回未初始化状态。在Greenplum数据库集群中重置PXF之前，必须停止在每个segment主机上运行的PXF服务实例。

start

在所有segment主机上启动PXF服务实例。

status

显示所有segment主机上的PXF服务实例的状态。

stop

在所有段主机上停止PXF服务实例。

sync

将PXF配置从master主机同步到standby和所有Greenplum数据库segment主机。如果您更新了PXF用户配置或添加了JAR文件，则在同步PXF配置后还必须重新启动PXF。

举例

在所有segment主机上停止PXF服务

```
$ $GPHOME/bin/pxf cluster stop
```

See Also

[pxf](#)

Greenplum数据库® 6.0文档

工具指南

管理工具参考

客户端工具参考

客户端工具摘要

clusterdb

createdb

createlang

createuser

dropdb

droplang

dropuser

pg_config

pg_dump

pg_dumpall

pg_restore

psql

reindexdb

vacuumdb

附加程序

Greenplum database 5管理员
指南

重新群集之前用CLUSTER群集过的表。

概要

```
clusterdb [connection-option ...] [--verbose | -v] [--table
| -t table] [[--dbname | -d] dbname]
```

```
clusterdb [connection-option ...] [--all | -a] [--verbose |
-v]
```

```
clusterdb --? | --help
```

```
clusterdb -v | --version
```

描述

对表进行群集意味着根据索引对磁盘上的表进行物理重新排序，以便索引扫描操作可以按某种顺序访问磁盘上的数据，从而提高使用该索引的查询的索引查找性能。

clusterdb工具将在数据库中找到以前已使用CLUSTER SQL命令进行群集的任何表，并将它们再次群集在最后使用的同一索引上。从未群集过的表不受影响。

clusterdb是SQL命令CLUSTER的包装。尽管Greenplum数据库支持以这种方式对表进行群集，但是不建议这样做，因为CLUSTER操作本身非常慢。

如果确实需要以这种方式对表进行排序以提高查询性能，请使用CREATE TABLE AS语句对磁盘上的表进行重新排序，而不要使用CLUSTER。如果您以这种方式“群集”表，那么clusterdb将不相关。

选项

-a | --all

群集所有数据库。



[-d] *dbname* | [--dbname=] *dbname*

指定要群集的数据库的名称。如果未指定，则从环境变量PGDATABASE读取数据库名称。如果未设置，则使用为连接指定的用户名。

-e | --echo

输出clusterdb生成并发送到服务器的命令。

-q | --quiet

不显示响应。

-t *table* | --table= *table*

仅群集命名表。可以通过编写多个-t开关来群集多个表。

-v | --verbose

在处理过程中打印详细信息。

-V | --version

打印clusterdb版本并退出。

-? | --help

显示有关clusterdb命令行参数的帮助，然后退出。

连接选项

-h *host* | --host= *host*

运行Greenplum master数据库服务器的计算机的主机名。如果未指定，则从环境变量PGHOST读取或默认为localhost。

-p *port* | --port= *port*

Greenplum master数据库服务器正在侦听连接的TCP端口。如果未指定，则从环境变量PGPORT读取或默认为5432。

-U *username* | --username= *username*

要用作连接的数据库角色名称。如果未指定，则从环境变量PGUSER读取或默认为当前系统角色名称。

-w | --no-password

不发出密码提示。如果服务器要求密码验证，而其他方式（例如.pgpass文件）无法使用密码，则连接尝试将失败。此选项在没有用户输入密码的批处理作业和脚本中很有用。

-W | --password

强制输入密码提示。

--maintenance-db= *dbname*

指定要连接以发现还应群集哪些其他数据库的数据库的名称。如果未指定，将使用postgres数据库，如果不存在，将使用template1。

示例

群集数据库test：

```
clusterdb test
```

在名为xyzzy的数据库中群集单个表foo：

```
clusterdb --table foo xyzzyb
```

另见

[CLUSTER](#)

Greenplum数据库® 6.0文档

[工具指南](#)

[管理工具参考](#)

[客户端工具参考](#)

[客户端工具摘要](#)

clusterdb

createdb

createlang

createuser

dropdb

droplang

dropuser

pg_config

pg_dump

pg_dumpall

pg_restore

psql

reindexdb

vacuumdb

[附加程序](#)

Greenplum database 5管理员
指南

创建一个新的数据库。

概要

```
createdb [connection-option ...] [option ...] [dbname
['description']]
```

```
createdb -? | --help
```

```
createdb -V | --version
```

描述

`createdb`在Greenplum数据库系统中创建一个新数据库。

通常，执行此命令的数据库用户将成为新数据库的所有者。但是，如果执行用户具有适当的特权，则可以通过`-o`选项指定其他所有者。

`createdb`是SQL命令`CREATE DATABASE`的包装。

选项

dbname

要创建的数据库的名称。该名称在Greenplum系统中的所有其他数据库中必须唯一。如果未指定，则从环境变量`PGDATABASE`中读取，然后从`PGUSER`中读取，或者默认为当前系统用户。

description

与新创建的数据库关联的注释。包含空格的说明必须用引号引起来。

-D *tablespace* | --tablespace= *tablespace*

指定数据库的默认表空间。（此名称作为双引号标识符处理。）

-e echo

回显`createdb`生成并发送到服务器的命令。



-E *encoding* | --encoding *encoding*

在新数据库中使用的字符集编码。指定字符串常量（例

如'UTF8')，整数编码数字或DEFAULT以使用默认编码。有关支持的字符集的信息，请参阅《Greenplum数据库参考指南》。

-l *locale* | --locale *locale*

指定此数据库中要使用的语言环境。这等效于同时指定--lc-collate和--lc-ctype。

--lc-collate *locale*

指定要在此数据库中使用的LC_COLLATE设置。

--lc-ctype *locale*

指定要在此数据库中使用的LC_CTYPE设置。

--maintenance-db= *dbname*

指定在创建新数据库时要连接的数据库的名称。如果未指定，将使用postgres数据库。如果不存在（或者它是正在创建的新数据库的名称），将使用template1。

-O *owner* | --owner= *owner*

拥有新数据库的数据库用户的名称。默认为执行此命令的用户。（此名称作为双引号标识符处理。）

-T *template* | --template= *template*

从中创建新数据库的模板的名称。默认为template1。（此名称作为双引号标识符处理。）

-V | --version

打印createdb版本并退出。

-? | --help

显示有关createdb命令行参数的帮助，然后退出。

选项-D, -l, -E, -O和-T对应于基础SQL命令CREATE DATABASE的选项。有关它们的更多信息，请参见[CREATE DATABASE](#)。 for more information about them.

连接选项

-h *host* | --host= *host*

运行Greenplum master数据库服务器的计算机的主机名。如果未指定，则从环境变量PGHOST读取或默认为localhost。

-p *port* | --port= *port*

Greenplum master数据库服务器正在侦听连接的TCP端口。如果未指定，则从环境变量PGPORT读取或默认为5432。

-U *username* | --username= *username*

要用作连接的数据库角色名称。如果未指定，则从环境变量PGUSER读取或默认为当前系统角色名称。

-w | --no-password

不发出密码提示。如果服务器要求密码验证，而其他方式（例如.pgpass文件）无法使用密码，则连接尝试将失败。此选项在没有用户输入密码的批处理作业和脚本中很有用。

-W | --password

强制输入密码提示。

示例

使用默认选项创建数据库test：

```
createdb test
```

使用主机gpmaster上的Greenplum master（端口54321）并使用LATIN1编码方案来创建数据库demo：

```
createdb -p 54321 -h gpmaster -E LATIN1 demo
```

另见

[CREATE DATABASE](#) , [dropdb](#)

Greenplum数据库® 6.0文档

为数据库定义新的过程语言。

□ 工具指南

□ 管理工具参考

□ 客户端工具参考

□ 客户端工具摘要

clusterdb

createdb

createlang

createuser

dropdb

droplang

dropuser

pg_config

pg_dump

pg_dumpall

pg_restore

psql

reindexdb

vacuumdb

□ 附加程序

Greenplum database 5管理员
指南

概要

```
createlang [connection_option ...] [-e] langname [[-d] dbname]
```

```
createlang [connection-option ...] -l dbname
```

```
createlang -? | --help
```

```
createlang -v | --version
```

描述

createlang工具将新的过程语言添加到数据库。

createlang是SQL命令[CREATE EXTENSION](#)的包装。

Note: **createlang**已弃用，将来的发行版中可能会将其删除。建议直接使用CREATE EXTENSION命令。

标准Greenplum数据库分发中包含的过程语言包是：

- PL/pgSQL
- PL/Perl
- PL/Python

默认情况下，PL/pgSQL语言已在所有数据库中注册。

Greenplum数据库还具有用于PL/Java和PL/R的语言处理程序，但是Greenplum数据库尚未预安装这些语言。有关更多信息，请参阅文档中的[Greenplum PL/Java语言扩展](#)和[Greenplum PL/R语言扩展](#)部分。

选项

langname

指定要安装的过程语言的名称。（此名称为小写。）



[-d] *dbname* | [--dbname=] *dbname*

指定应向其中添加语言的数据库。默认设置是使用PGDATABASE环境变量设置，或与当前系统用户相同的名称。

-e | --echo

回显createlang生成并发送到服务器的命令。

-l *dbname* | --list *dbname*

显示目标数据库中已经安装的语言的列表。

-V | --version

打印createlang版本并退出。

-? | --help

显示有关createlang命令行参数的帮助，然后退出。

连接选项

-h *host* | --host= *host*

运行Greenplum master数据库服务器的计算机的主机名。如果未指定，则从环境变量PGHOST读取或默认为localhost。

-p *port* | --port= *port*

Greenplum主数据库服务器正在侦听连接的TCP端口。如果未指定，则从环境变量PGPORT读取或默认为5432。

-U *username* | --username= *username*

要用作连接的数据库角色名称。如果未指定，则从环境变量PGUSER读取或默认为当前系统角色名称。

-w | --no-password

不发出密码提示。如果服务器要求密码验证，而其他方式（例如.pgpass文件）无法使用密码，则连接尝试将失败。此选项在没有用户输入密码的批处理作业和脚本中很有用。

-W | --password

强制输入密码提示。

示例

要将语言plperl安装到数据库mytestdb中：

```
createlang plperl mytestdb
```

另见

[droplang](#), [CREATE EXTENSION](#), [CREATE LANGUAGE](#), [DROP](#)

[LANGUAGE](#)

Greenplum数据库® 6.0文档

[工具指南](#)

[管理工具参考](#)

[客户端工具参考](#)

[客户端工具摘要](#)

clusterdb

createdb

createlang

createuser

dropdb

droplang

dropuser

pg_config

pg_dump

pg_dumpall

pg_restore

psql

reindexdb

vacuumdb

[附加程序](#)

Greenplum database 5管理员
指南

创建一个新的数据库角色。

概要

```
createuser [connection-option ...] [role_attribute ...] [-e] role_name
```

```
createuser -? | --help
```

```
createuser -v | --version
```

描述

`createuser` 创建一个新的Greenplum数据库角色。 您必须是超级用户或具有`CREATEROLE`权限才能创建新角色。 您必须以超级用户身份连接到数据库才能创建新的超级用户。

超级用户可以绕过数据库中的所有访问权限检查，因此不应轻易授予超级用户特权。

`createuser` 是SQL命令`CREATE ROLE`的包装。

选项

role_name

要创建的角色的名称。 此名称必须不同于该Greenplum数据库安装中的所有现有角色。

-c *number* | **--connection-limit=** *number*

设置新角色的最大连接数。 默认设置为无限制。

-d | **--createdb**

新角色将被允许创建数据库。

-D | **--no-createdb**

不允许新角色创建数据库。 这是默认值。

-e | **--echo**

回显`createuser`生成并发送到服务器的命令。

-E | **--encrypted**



加密存储在数据库中的角色密码。如果未指定，则使用默认密码行为。

-i | --inherit

新角色将自动继承其成员角色的特权。这是默认值。

-I | --no-inherit

新角色将不会自动继承其成员角色的特权。

--interactive

如果在命令行上未指定用户名，则提示输入用户名，并提示在命令行中未指定选项-d/-D, -r/-R, -s/-S中的任何一个。

-l | --login

新角色将被允许登录Greenplum数据库。这是默认值。

-L | --no-login

新角色将不允许登录（组级别角色）。

-N | --unencrypted

不加密存储在数据库中的角色密码。如果未指定，则使用默认密码行为。

-P | --pwprompt

如果提供，则createuser将提示输入新角色的密码。如果您不打算使用密码身份验证，则没有必要。

-r | --createrole

新角色将被允许创建新角色（CREATEROLE特权）。

-R | --no-createrole

新角色将不允许创建新角色。这是默认值。

-s | --superuser

新角色将是超级用户。

-S | --no-superuser

新角色将不是超级用户。这是默认值。

-V | --version

打印createuser版本并退出。

--replication

新用户将具有REPLICATION特权，有关CREATE ROLE 的文档中对此进行了更详细的描述。

--no-replication

新用户将没有REPLICATION特权，有关CREATE ROLE 的文档中对此进行了更详细的描述。

-? | --help

显示有关createuser命令行参数的帮助，然后退出。

连接选项

-h host | --host= host

运行Greenplum master数据库服务器的计算机的主机名。如果未指定，则从环境变量PGHOST读取或默认为localhost。

-p port | --port= port

Greenplum master数据库服务器正在侦听连接的TCP端口。如果

未指定，则从环境变量PGPORT读取或默认为5432。

-U *username* | --username= *username*

要用作连接的数据库角色名称。如果未指定，则从环境变量PGUSER读取或默认为当前系统角色名称。

-w | --no-password

不发出密码提示。如果服务器要求密码验证，而其他方式（例如.pgpass文件）无法使用密码，则连接尝试将失败。此选项在没有用户输入密码的批处理作业和脚本中很有用。

-W | --password

强制输入密码提示。

示例

要在默认数据库服务器上创建角色joe：

```
$ createuser joe
```

要在默认数据库服务器上创建角色joe并提示一些其他属性：

```
$ createuser --interactive joe
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n) n
Shall the new role be allowed to create more new roles?
(y/n) n
CREATE ROLE
```

要使用连接选项来创建相同的角色joe，并显式指定属性，然后查看基本命令：

```
createuser -h masterhost -p 54321 -s -D -R -e joe
CREATE ROLE joe NOSUPERUSER NOCREATEDB NOCREATEROLE INHERIT
LOGIN;
CREATE ROLE
```

要将角色joe创建为超级用户，并立即分配密码admin123：

```
createuser -P -s -e joe
Enter password for new role: admin123
Enter it again: admin123
CREATE ROLE joe PASSWORD 'admin123' SUPERUSER CREATEDB
CREATEROLE INHERIT LOGIN;
CREATE ROLE
```

在上面的示例中，键入时实际上并未回显新密码，但是为了清楚起见，我们显示了键入的内容。但是，密码将显示在echoed命令中，如使用-e选项所示。

另见

[CREATE ROLE](#) , [dropuser](#)

删除数据库。

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

□ 客户端工具参考

□ 客户端工具摘要

clusterdb

createdb

createlang

createuser

dropdb

droplang

dropuser

pg_config

pg_dump

pg_dumpall

pg_restore

psql

reindexdb

vacuumdb

□ 附加程序

Greenplum database 5管理员
指南

概要

```
dropdb [connection-option ...] [-e] [-i] dbname
```

```
dropdb -? | --help
```

```
dropdb -v | --version
```

描述

dropdb销毁现有数据库。执行此命令的用户必须是超级用户或要删除的数据库的所有者。

dropdb是SQL命令DROP DATABASE的包装。有关DROP DATABASE的信息，请参阅《Greenplum数据库参考指南》。

选项

dbname

要删除的数据库的名称。

-e | --echo

回显dropdb生成并发送到服务器的命令。

-i | --interactive

在执行破坏性操作之前发出验证提示。

-V | --version

打印dropdb版本并退出。

--if-exists

如果数据库不存在，请不要报错。在这种情况下发出通知。

-? | --help

显示有关dropdb命令行参数的帮助，然后退出。

连接选项

-h *host* | --host= *host*

运行Greenplum master数据库服务器的计算机的主机名。如果未



指定，则从环境变量PGHOST读取或默认为localhost。

-p port | --port= port

Greenplum master数据库服务器正在侦听连接的TCP端口。如果未指定，则从环境变量PGPORT读取或默认为5432。

-U username | --username= username

要用作连接的数据库角色名称。如果未指定，则从环境变量PGUSER读取或默认为当前系统角色名称。

-w | --no-password

不发出密码提示。如果服务器要求密码验证，而其他方式（例如.pgpass文件）无法使用密码，则连接尝试将失败。此选项在没有用户输入密码的批处理作业和脚本中很有用。

-W | --password

强制输入密码提示。

--maintenance-db= dbname

指定要删除目标数据库要连接的数据库的名称。如果未指定，将使用postgres数据库。如果不存在（或者它是要删除的数据库的名称），则将使用template1。

示例

要使用默认连接参数销毁名为demo的数据库：

```
dropdb demo
```

要使用连接选项（带有验证）并查看基础命令来删除名为demo的数据库： To destroy the database named demo using connection options, with verification, and a peek at the underlying command:

```
dropdb -p 54321 -h masterhost -i -e demo
Database "demo" will be permanently deleted.
Are you sure? (y/n) y
DROP DATABASE "demo"
DROP DATABASE
```

另见

[createdb](#) , [DROP DATABASE](#)

删除过程语言。

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

□ 客户端工具参考

□ 客户端工具摘要

clusterdb

createdb

createlang

createuser

dropdb

droplang

dropuser

pg_config

pg_dump

pg_dumpall

pg_restore

psql

reindexdb

vacuumdb

□ 附加程序

Greenplum database 5管理员
指南

概要

```
droplang [connection-option ...] [-e] langname [[-d] dbname]
```

```
droplang [connection-option ...] [-e] -l dbname
```

```
droplang -? | --help
```

```
droplang -V | --version
```

描述

droplang从数据库中删除现有的过程语言。

droplang是SQL命令[DROP EXTENSION](#)的包装。

选项

langname

指定要删除的过程语言的名称。 (此名称为小写。)
[-d] *dbname* | [--dbname=] *dbname*

指定应从哪个数据库中删除语言。默认设置是使用PGDATABASE环境变量设置，或与当前系统用户相同的名称。

-e | --echo

回显**droplang**生成并发送到服务器的命令。

-l | --list

显示目标数据库中已经安装的语言的列表。

-V | --version

打印**droplang**版本并退出。

-? | --help

显示有关**droplang**命令行参数的帮助，然后退出。

连接选项



-h *host* | --host= *host*

运行Greenplum master数据库服务器的计算机的主机名。如果未指定，则从环境变量PGHOST读取或默认为localhost。

-p *port* | --port= *port*

Greenplum主数据库服务器正在侦听连接的TCP端口。如果未指定，则从环境变量PGPORT读取或默认为5432。

-U *username* | --username= *username*

要用作连接的数据库角色名称。如果未指定，则从环境变量PGUSER读取或默认为当前系统角色名称。

-w | --no-password

不发出密码提示。如果服务器要求密码验证，而其他方式（例如.pgpass文件）无法使用密码，则连接尝试将失败。此选项在没有用户输入密码的批处理作业和脚本中很有用。

-W | --password

强制输入密码提示。

示例

要从mydatabase数据库中删除语言pltcl：

```
droplang pltcl mydatabase
```

另见

[createlang](#) , [DROP EXTENSION](#) , [DROP LANGUAGE](#)

Greenplum数据库® 6.0文档

[工具指南](#)

[管理工具参考](#)

[客户端工具参考](#)

[客户端工具摘要](#)

clusterdb

createdb

createlang

createuser

dropdb

droplang

dropuser

pg_config

pg_dump

pg_dumpall

pg_restore

psql

reindexdb

vacuumdb

[附加程序](#)

Greenplum database 5管理员
指南

删除数据库角色。

概要

```
dropuser [connection-option ...] [-e] [-i] role_name
```

```
dropuser -? | --help
```

```
dropuser -V | --version
```

描述

dropuser从Greenplum数据库中删除现有角色。 只有超级用户和具有CREATEROLE特权的用户才能删除角色。 要删除超级用户角色，您必须自己是超级用户。

dropuser是SQL命令DROP ROLE的包装。

选项

role_name

要删除的角色的名称。 如果未在命令行上指定，则将提示您输入名称，并且使用了**-i**--interactive选项。

-e | --echo

回显**dropuser**生成并发送到服务器的命令。

-i | --interactive

在实际删除角色之前提示确认，如果在命令行上未指定角色名称，则提示输入角色名称。

--if-exists

如果用户不存在，请不要报错。在这种情况下发出通知。

-V | --version

打印**dropuser**版本并退出。

-? | --help

显示有关**dropuser**命令行参数的帮助，然后退出。

连接选项

-h host | --host= host

运行Greenplum master数据库服务器的计算机的主机名。如果未指定，则从环境变量PGHOST读取或默认为localhost。

-p port | --port= port

Greenplum master数据库服务器正在侦听连接的TCP端口。如果未指定，则从环境变量PGPORT读取或默认为5432。

-U username | --username= username

要用作连接的数据库角色名称。如果未指定，则从环境变量PGUSER读取或默认为当前系统角色名称。

-w | --no-password

不发出密码提示。如果服务器要求密码验证，而其他方式（例如.pgpass文件）无法使用密码，则连接尝试将失败。此选项在没有用户输入密码的批处理作业和脚本中很有用。

-W | --password

强制输入密码提示。

示例

要使用默认连接选项删除角色joe：

```
dropuser joe
DROP ROLE
```

要使用连接选项（带有验证）并查看基本命令来删除角色joe：

```
dropuser -p 54321 -h masterhost -i -e joe
Role "joe" will be permanently removed.
Are you sure? (y/n) y
DROP ROLE "joe"
DROP ROLE
```

另见

[createuser](#) , [DROP ROLE](#)

Greenplum数据库® 6.0文档

[工具指南](#)

[管理工具参考](#)

[客户端工具参考](#)

[客户端工具摘要](#)

clusterdb

createdb

createlang

createuser

dropdb

droplang

dropuser

pg_config

pg_dump

pg_dumpall

pg_restore

psql

reindexdb

vacuumdb

[附加程序](#)

Greenplum database 5管理员
指南

检索有关Greenplum数据库已安装版本的信息。

概要

```
pg_config [option ...]
```

```
pg_config --? | --help
```

```
pg_config --version
```

描述

`pg_config`工具显示当前安装的Greenplum数据库版本的配置参数。例如，它打算由希望与Greenplum数据库进行接口连接的软件包使用，以便于查找所需的头文件和库。请注意，由`pg_config`打印出的信息仅适用于Greenplum数据库master。

如果给出多个选项，则按该顺序打印信息，每行打印一项。如果未给出任何选项，则会打印所有可用信息以及标签。

选项

--bindir

打印用户可执行文件的位置。例如，使用它来查找`psql`程序。通常这也是`pg_config`程序所在的位置。

--docdir

打印文档文件的位置。

--includedir

打印客户端接口的C头文件的位置。

--pkgincludedir

打印其他C头文件的位置。

--includedir-server

打印C头文件的位置以进行服务器编程。

--libdir

打印对象代码库的位置。

--pkglibdir



打印动态可加载模块的位置，或服务器在其中搜索它们的位置。
(其他与体系结构相关的数据文件也可以安装在此目录中。)

--localedir
打印语言环境支持文件的位置。

--mandir
打印手册页的位置。

--sharedir
打印与体系结构无关的支持文件的位置。

--sysconfdir
打印系统范围的配置文件的位置。

--pgxs
打印扩展makefile的位置。

--configure
打印为Greenplum数据库配置构建时为配置脚本提供的选项。

--cc
打印用于构建Greenplum数据库的CC变量的值。这显示了使用的C编译器。

--cppflags
打印用于构建Greenplum数据库的CPPFLAGS变量的值。这显示了预处理时需要的C编译器开关。

--cflags
打印用于构建Greenplum数据库的CFLAGS变量的值。这显示了C编译器开关。

--cflags_sl
打印用于构建Greenplum数据库的CFLAGS_SL变量的值。这显示了用于构建共享库的其他C编译器开关。

--ldflags
打印用于构建Greenplum数据库的LDFLAGS变量的值。这显示了链接器开关。

--ldflags_ex
打印用于构建Greenplum数据库的LDFLAGS_EX变量的值。这显示了仅用于生成可执行文件的链接器开关。

--ldflags_sl
打印用于构建Greenplum数据库的LDFLAGS_SL变量的值。这显示了仅用于构建共享库的链接器开关。

--libs
打印用于构建Greenplum数据库的LIBS变量的值。它通常包含-l开关，用于链接到Greenplum数据库的外部库。

--version
打印Greenplum数据库的版本。

示例

要重现当前Greenplum数据库安装的构建配置，请运行以下命令：

```
eval ./configure 'pg_config --configure'
```

`pg_config --configure`的输出包含shell引号，因此带有空格的参数可以正确表示。因此，需要使用`eval`才能获得正确的结果。

Greenplum数据库® 6.0文档

工具指南

管理工具参考

客户端工具参考

客户端工具摘要

clusterdb

createdb

createlang

createuser

dropdb

droplang

dropuser

pg_config

pg_dump

pg_dumpall

pg_restore

psql

reindexdb

vacuumdb

附加程序

Greenplum database 5管理员
指南

Extracts a database into a single script file or other archive file.

概要

```
pg_dump [connection-option ...] [dump_option ...] [dbname]
```

```
pg_dump -? | --help
```

```
pg_dump -v | --version
```

描述

`pg_dump`是用于备份数据库的标准PostgreSQL工具，在Greenplum数据库中也受支持。它创建一个（非并行）转储文件。对于Greenplum数据库的常规备份，最好使用Greenplum数据库备份工具[gpbackup](#)以获得最佳性能。

如果要将数据迁移到另一个数据库供应商的系统或具有不同segment配置的另一个Greenplum数据库系统（例如，如果要迁移的系统具有更多或更少的segment实例），请使用`pg_dump`。要还原，必须使用相应的[pg_restore](#)工具（如果转储文件为归档格式），或者可以使用诸如[psql](#)之类的客户端程序（如果转储文件为纯文本格式）。

由于`pg_dump`与常规PostgreSQL兼容，因此可以用于将数据迁移到Greenplum数据库中。Greenplum数据库中的`pg_dump`工具与PostgreSQL `pg_dump`工具非常相似，但有以下例外和限制：

- 如果使用`pg_dump`备份Greenplum数据库数据库，请记住，对于大型数据库，转储操作可能需要很长时间（几个小时）。另外，必须确保您有足够的磁盘空间来创建转储文件。
- 如果要将数据从一个Greenplum数据库系统迁移到另一个系统，请使用`--gp-syntax`命令行选项在CREATE TABLE语句中包括DISTRIBUTED BY子句。这样可以确保在还原时使用正确的分发键列分发Greenplum数据库表数据。

即使同时使用数据库，`pg_dump`也会进行一致的备份。`pg_dump`不会阻止其他用户访问数据库（读或写）。

当与一种存档文件格式一起使用并与`pg_restore`结合使用时，`pg_dump`提供了一种灵活的归档和传输机制。`pg_dump`可用于备份整个数据库，然后`pg_restore`可用于检查存档和/或选择要还原数据库的哪些部分。最灵活的输出文件格式是自定义格式（`-Fc`）和目录格式（`-Fd`）。它们允许对所有已归档项目进行选择和重新排序，支持并行还原，并且默认情况下已压缩。目录格式是唯一支持并行转储的格式。

Note: `--ignore-version`选项已弃用，在以后的版本中将被删除。

选项

dbname

指定要转储的数据库的名称。如果未指定，则使用环境变量`PGDATABASE`。如果未设置，则使用为连接指定的用户名。

转储选项

`-a | --data-only`

仅转储数据，而不转储模式（数据定义）。表数据和序列值将转储。

此选项与`--section=data`相似，但由于历史原因不同。

`-b | --blobs`

在转储中包括大对象。这是默认行为，除非指定了`--schema`, `--table`或`--schema-only`。`-b`开关仅用于在选择性转储的时候添加大对象。请注意，`blob`被视为数据，因此，当使用`--data-only`时将包括这些`blob`，但在使用`--schema-only`时则不包括。

Note: Greenplum数据库不支持PostgreSQL[大对象工具](#)来流存储在大对象结构中的用户数据。

`-c | --clean`

在输出用于创建数据库对象的命令之前，将命令添加到文本输出文件中以清理（删除）数据库对象。（如果目标数据库中不存在任何对象，则恢复可能会生成一些无害的错误消息。）请注意，在转储操作开始之前不会删除对象，但是会将`DROP`命令添加到`DDL`转储输出文件中，以便在使用这些文件时要还原文件，要先执行`DROP`命令，再执行`CREATE`命令。此选项仅对纯文本格式有意义。对于归档格式，可以在调用`pg_restore`时指定该选项。

`-C | --create`

从命令开始输出，以创建数据库本身并重新连接到创建的数据。库。（使用这种形式的脚本，在运行脚本之前连接到目标安装中的哪个数据库都没有关系。）如果还指定了`--clean`，则脚

本会删除并在重新连接目标数据库之前重新创建它。此选项仅对纯文本格式有意义。对于归档格式，可以在调用`pg_restore`时指定该选项。

`-E encoding | --encoding= encoding`

以指定的字符集编码创建转储。默认情况下，转储以数据库编码创建。（获得相同结果的另一种方法是将`PGCLIENTENCODING`环境变量设置为所需的转储编码。）

`-f file | --file= file`

将输出发送到指定文件。对于基于文件的输出格式，可以省略此参数，在这种情况下，将使用标准输出。但是，必须为目录输出格式提供该格式，该格式指定目标目录而不是文件。在这种情况下，该目录是由`pg_dump`创建的，以前不能存在。

`-F p|c|d|t | --format=plain|custom|directory|tar`

选择输出格式。格式可以是以下之一：

`p | plain` — 输出纯文本SQL脚本文件（默认）。

`c | custom` — 输出适合输入到`pg_restore`的自定义存档。与目录输出格式一起使用时，这是最灵活的输出格式，因为它允许在还原过程中手动选择和重新排序已归档的项目。默认情况下，此格式为压缩格式，并且还支持并行转储。

`d | directory` — 输出适合于输入`pg_restore`的目录格式档案。

这将创建一个目录，其中包含要转储的每个表和blob的一个文件，以及一个所谓的目录文件，该表以`pg_restore`可以读取的机器可读格式描述了转储的对象。目录格式归档文件可以使用标准的Unix工具进行处理。例如，可以使用`gzip`工具压缩未压缩档案中的文件。默认情况下压缩此格式。

`t | tar` — 输出适合输入到`pg_restore`的tar格式的存档。tar格式与目录格式兼容；提取tar格式的存档会生成有效的目录格式的存档。但是，tar格式不支持压缩。同样，在使用tar格式时，在还原过程中不能更改表数据项的相对顺序。

`-i | --ignore-version`

Note: 此选项已被弃用，并将在以后的版本中删除。

忽略`pg_dump`和数据库服务器之间的版本不匹配。`pg_dump`可以从运行早期版本的Greenplum数据库（或PostgreSQL）的服务器中转储，但是可能不再支持非常旧的版本。如果您需要覆盖版本检查，请使用此选项。

`-j njobs | --jobs= njobs`

通过同时转储`njobs`表来并行运行转储。此选项减少了转储时间，但同时也增加了数据库服务器上的负载。您只能将此选项与目录输出格式一起使用，因为这是多个进程可以同时写入其数据的唯一输出格式。

Note: 使用`pg_dump`的并行转储仅在查询调度程序（master）节点上并行化，而不是在使用`gpbackup`时跨查询执行器（segment）节点并行化。

`pg_dump`将打开`njobs + 1`个到数据库的连接，因此请确保您

的`max_connections`设置足够高以容纳所有连接。

在运行并行转储时请求对数据库对象的排他锁可能导致转储失败。原因是`pg_dump`主进程对工作进程稍后将要转储的对象请求共享锁，以确保没有人删除它们并在转储运行时使它们消失。如果另一个客户端随后请求对表进行排他锁，则该锁将不会被授予，但将排队等待主进程的共享锁被释放。因此，对该表的任何其他访问也不会被授予，并且将在排他锁定请求之后排队。这包括尝试转储表的工作进程。如果没有任何预防措施，这将是典型的死锁情况。为了检测到这种冲突，`pg_dump`工作进程使用`NOWAIT`选项请求另一个共享锁。如果未向工作进程授予此共享锁，则其他人在此期间必须申请到独占锁，并且无法继续进行转储，因此`pg_dump`别无选择，只能中止转储。

为了保持一致的备份，数据库服务器需要支持同步快照，这是Greenplum数据库6.0中引入的功能。使用此特性，即使数据库客户端使用不同的连接，也可以确保他们看到相同的数据集。`pg_dump -j`使用多个数据库连接；它通过主进程一次连接到数据库，并针对每个工作者作业再次连接到数据库。如果没有同步快照特性，将无法保证不同的工作作业在每个连接中都看到相同的数据，这可能导致备份不一致。

如果要运行6.0之前版本服务器的并行转储，则需要确保从主服务器连接到数据库到最后一个工作者作业连接到数据库之间的时间里，数据库的内容没有变化。最简单的方法是在开始备份之前，停止所有访问数据库的数据修改过程（`DDL`和`DML`）。在6.0之前的Greenplum数据库服务器上运行`pg_dump -j`时，还需要指定`--no-synchronized-snapshots`参数。

`-n schema | --schema= schema`

仅转储与`schema`匹配的模式；这将同时选择模式本身及其所有包含的对象。如果未指定此选项，则将转储目标数据库中的所有非系统模式。通过编写多个`-n`开关可以选择多个模式。同样，根据`psql`的`\d`命令使用的相同规则，将`schema`参数解释为模式，因此也可以通过在模式中写入通配符来选择多个模式。使用通配符时，如果需要请小心引用该模式，以防止Shell扩展通配符。

注意：当指定`-n`时，`pg_dump`不会尝试转储所选模式可能依赖的任何其他数据库对象。因此，不能保证特定模式转储的结果可以自己成功地恢复到干净的数据库中。

Note: 指定`-n`时，不转储非模式对象（例如`blob`）。您可以使用`--blobs`开关将`blob`添加回转储。

`-N schema | --exclude-schema= schema`

不要转储任何与`schema`匹配的模式。根据与`-n`相同的规则解释该模式。`-N`可以多次给出，以排除与几种模式中的任何一种匹配的模式。当同时给出`-n`和`-N`时，行为是仅转储与至少一个`-n`开关匹配但不与`-N`开关匹配的模式。如果出现`-N`而没有`-n`，则与`-N`匹配的模式将从正常转储中排除。

-o | --oids

转储对象标识符 (OID) 作为每个表的数据的一部分。对于要还原到Greenplum数据库中的文件，建议不要使用此选项。

-O | --no-owner

不要输出命令来设置对象的所有权以匹配原始数据库。默认情况下，`pg_dump`发出`ALTER OWNER`或`SET SESSION AUTHORIZATION`语句来设置创建的数据库对象的所有权。除非由超级用户（或拥有脚本中所有对象的同一用户）启动脚本，否则运行脚本时这些语句将失败。要使脚本可以被任何用户恢复，但将赋予该用户所有对象的所有权，请指定`-o`。此选项仅对纯文本格式有意义。对于归档格式，可以在调用`pg_restore`时指定该选项。

-s | --schema-only

仅转储对象定义（模式），而不转储数据。

此选项与`--data-only`相反。它类似于`--section = pre-data` `--section = post-data`，但由于历史原因不同。

（不要将此与`--schema`选项混淆，该选项以不同的含义使用“schema”一词。）

要仅排除数据库中一部分表的表数据，请参见`--exclude-table-data`。

-S *username* | --superuser= *username*

指定禁用触发器时要使用的超级用户名。仅在使用`--disable-triggers`时才有意义。最好不要这样做，而是以超级用户身份启动生成的脚本。

Note: Greenplum数据库不支持用户定义的触发器。

-t *table* | --table= *table*

仅转储与表模式匹配的表（或视图，序列或外部表）。以`schema.table`格式指定表。

通过写入多个`-t`开关可以选择多个表。而且，根据`psql`\d命令使用的相同规则，将`table`参数解释为模式，因此也可以通过在模式中写入通配符来选择多个表。使用通配符时，如果需要请小心引用模式，以防止Shell扩展通配符。使用`-t`时`-n`和`-N`开关无效，因为`-t`选择的表将被转储，而与那些开关无关，并且非表对象也将不被转储。

Note: 当指定`-t`时，`pg_dump`不会尝试转储所选表可能依赖的任何其他数据库对象。因此，不能保证自己可以成功地将特定表转储的结果还原到干净的数据库中。

另外，`-t`不能用于指定子表分区。要转储分区表，必须指定父表名称。

-T *table* | --exclude-table= *table*

不要转储任何与表模式匹配的表。该模式根据与`-t`相同的规则进行解释。`-T`可以多次给出，以排除与几种模式中的任何一种

-t -T

匹配的表。当同时给出 `-t` 和 `-T` 时，行为是仅转储与至少一个 `t` 开关匹配但不与 `-T` 开关匹配的表。如果出现 `-T` 而没有 `-t`，则与 `-T` 匹配的表将从正常转储中排除。

`-v | --verbose`

指定详细模式。这将导致 `pg_dump` 向转储文件输出详细的对象注释和开始/停止时间，并向标准错误输出消息。

`-V | --version`

打印 `pg_dump` 版本并退出。

`-x | --no-privileges | --no-acl`

防止转储访问权限（`GRANT/REVOKE` 命令）。

`-Z 0..9 | --compress=0..9`

指定要使用的压缩级别。零表示无压缩。对于自定义存档格式，此选项指定压缩单个表数据段，并且默认设置为中等压缩。对于纯文本输出，设置非零压缩级别会使整个输出文件被压缩，就好像它是通过 `gzip` 馈送的一样。但默认设置为不压缩。`tar` 存档格式当前根本不支持压缩。

`--binary-upgrade`

此选项供就地升级工具使用。不建议或不支持将其用于其他目的。该选项的行为在将来的版本中可能会更改，恕不另行通知。

`--column-inserts | --attribute-inserts`

将数据转储为带有显式列名（(`INSERT INTO` *table* (*column*, ...)) `VALUES` ...) 的 `INSERT` 命令。这会使恢复非常缓慢。它主要用于制作可以装入非基于 PostgreSQL 的数据库的转储。但是，由于此选项为每一行生成一个单独的命令，因此在重新加载行时发生错误只会导致该行丢失，而不是整个表内容丢失。

`--disable-dollar-quoting`

此选项禁止对函数体使用美元引号，并强制使用 SQL 标准字符串语法对其进行引用。

`--disable-triggers`

仅当创建仅数据转储时，此选项才相关。它指示 `pg_dump` 包含一些命令，以在重新加载数据时临时禁用目标表上的触发器。如果不想在数据重装期间调用的表上有触发器，请使用此选项。为 `--disable-triggers` 发出的命令必须以超级用户身份执行。因此，您还应该使用 `-s` 指定超级用户名，或者最好小心地以超级用户身份启动生成的脚本。此选项仅对纯文本格式有意义。对于归档格式，可以在调用 `pg_restore` 时指定该选项。

Note: Greenplum 数据库不支持用户定义的触发器。

`--exclude-table-data=table`

不要为与 `table` 模式匹配的任何表转储数据。该模式根据与 `-t` 相同的规则进行解释。`--exclude-table-data` 可以多次给出，以排除与多种模式中的任何一种匹配的表。当您需要特定表的定义时，即使您不需要其中的数据，此选项也很有用。

要排除数据库中所有表的数据，请参见`--schema-only`。

`--if-exists`

清理数据库对象时，请使用条件命令（即添加`IF EXISTS`子句）。除非还指定了`--clean`，否则此选项无效。

`--inserts`

将数据转储为`INSERT`命令（而不是`COPY`）。这会使恢复非常缓慢。它主要用于制作可以装入非基于PostgreSQL的数据库的转储。但是，由于此选项为每一行生成一个单独的命令，因此在重新加载行时发生错误只会导致该行丢失，而不是整个表内容丢失。请注意，如果您重新排列了列顺序，则还原可能会完全失败。`--column-inserts`选项可以安全地防止列顺序更改，尽管速度更慢。

`--lock-wait-timeout=` *timeout*

不要在转储开始时一直等待获取共享表锁。相反，如果无法在指定的*timeout*时间内锁定表，则失败。将*timeout*指定为毫秒数。

`--no-security-labels`

不要转储安全标签。

`--no-synchronized-snapshots`

该选项允许在6.0之前版本的Greenplum数据库服务器上运行`pg_dump -j`。有关更多详细信息，请参见`-j`参数的文档。

`--no-tablespaces`

不输出命令以选择表空间。使用此选项，将在还原期间的默认表空间中创建所有对象。

此选项仅对纯文本格式有意义。对于归档格式，可以在调用`pg_restore`时指定选项。

`--no-unlogged-table-data`

不要转储未记录表的内容。此选项对是否转储表定义（模式）没有影响。它仅禁止转储表数据。从备用服务器转储时，始终排除未记录表中的数据。

`--quote-all-identifiers`

强制给所有标识符加上引号。当从Greenplum数据库主版本不同于`pg_dump`的中转储数据库时，或者打算将输出加载到其他主版本的服务器中时，建议使用此选项。默认情况下，`pg_dump`仅为在其主版本中为保留字的标识符加引号。在处理其他版本的保留字可能略有不同的服务器时，有时会导致兼容性问题。使用`--quote-all-identifiers`可以防止此类问题，但代价是难以阅读的转储脚本。

`--section=` *sectionname*

仅转储命名节。节名称可以是`pre-data`, `data`或`post-data`。可以多次指定此选项以选择多个节。默认为转储所有节。

`data`节包含实际的表数据和序列值。`post-data`项包括索引，触发器，规则和约束的定义，而不是经过验证的检查约束。

`pre-data`项包括所有其他数据定义项。

--serializable-deferrable

对转储使用可序列化的事务，以确保使用的快照与以后的数据库状态一致；但这要通过等待事务流中不会出现异常的点来完成，这样就不会存在转储失败或导致其他事务因`serialization_failure`而回滚的风险。

此选项对仅用于灾难恢复的转储无益。这对于在原始数据库继续更新的同时用于加载数据库副本以进行报告或其他只读负载共享的转储很有用。没有它，转储可能反映出与最终提交的事务的任何串行执行都不相符的状态。例如，如果使用批处理技术，则在转储中批次可能显示为已关闭，而批次中的所有项目都不会出现。

如果在启动`pg_dump`时没有活动的读写事务，则此选项没有任何区别。如果读写事务处于活动状态，则转储的开始可能会延迟不确定的时间长度。运行后，无论有无选项，性能都是相同的。

Note: 由于Greenplum数据库不支持可序列化的事务，因此`--serializable-deferrable`选项在Greenplum数据库中无效。

--use-set-session-authorization

输出SQL标准的`SET SESSION AUTHORIZATION`命令而不是`ALTER OWNER`命令来确定对象所有权。这使转储更加符合标准，但是依赖转储中对象的历史记录，可能无法正确还原。使用`SET SESSION AUTHORIZATION`进行的转储将需要超级用户权限才能正确还原，而`ALTER OWNER`则需要较少的权限。

--gp-syntax | --no-gp-syntax

使用`--gp-syntax`在`CREATE TABLE`语句中转储Greenplum数据库语法。这允许转储Greenplum数据库表的分发策略（`DISTRIBUTED BY`或`DISTRIBUTED RANDOMLY`子句），这对于还原到其他Greenplum数据库系统很有用。默认是在连接到Greenplum数据库系统时包括Greenplum数据库语法，而在连接到常规PostgreSQL系统时将其排除。

--function-oids *oids*

转储对象标识符的*oids*列表中指定的函数。

Note: 仅提供此选项供其他管理工具使用。不建议或不支持将其用于任何其他目的。该选项的行为在将来的版本中可能会更改，恕不另行通知。

--relation-oids *oids*

转储对象标识符的*oids*列表中指定的关系。

Note: 仅提供此选项供其他管理工具使用。不建议或不支持将其用于任何其他目的。该选项的行为在将来的版本中可能会更改，恕不另行通知。

-? | --help

显示有关`pg_dump`命令行参数的帮助，然后退出。

连接选项

-d *dbname* | --dbname= *dbname*

指定要连接的数据库的名称。这等效于在命令行上将*dbname*指定为第一个非选项参数。

如果此参数包含=符号或以有效的URI前缀

(`postgresql://`或`postgres://`) 开头，则将其视为*conninfo*字符串。有关更多信息，请参见PostgreSQL文档中的[连接字符串](#)。

-h *host* | --host= *host*

运行Greenplum数据库master数据库服务器的计算机的主机名。

如果未指定，则从环境变量PGHOST读取或默认为localhost。

-p *port* | --port= *port*

Greenplum master数据库服务器正在侦听连接的TCP端口。如果未指定，则从环境变量PGPORT读取或默认为5432。

-U *username* | --username= *username*

要用作连接的数据库角色名称。如果未指定，则从环境变量PGUSER读取或默认为当前系统角色名称。

-W | --password

强制输入密码提示。

-w | --no-password

不发出密码提示。如果服务器要求密码验证，而其他方式（例如.`pgpass`文件）无法使用密码，则连接尝试将失败。此选项在没有用户输入密码的批处理作业和脚本中很有用。

--role= *rolename*

指定用于创建转储的角色名称。此选项使`pg_dump`在连接到数据库后发出`SET ROLE rolename`命令。当通过身份验证的用户（由-U指定）缺少`pg_dump`所需的特权，但可以切换到具有所需权限的角色时，此功能很有用。某些安装有禁止直接以超级用户身份登录的策略，并且使用此选项可以在不违反策略的情况下进行转储。

注解

当选择了仅数据转储并且使用了选项`--disable-triggers`时，`pg_dump`发出命令以在插入数据之前禁用用户表上的触发器，并在插入数据后发出命令以重新启用它们。如果还原在中间停止，则系统catalog可能处于错误状态。

`pg_dump`生成的转储文件不包含优化器用于制定查询计划决策的统计信息。因此，从转储文件还原后运行`ANALYZE`是明智的，以确保最佳性能。

pg_dump的数据库活动通常由统计信息收集器收集。如果不希望这样做，则可以通过`PGOPTIONS`或`ALTER USER`命令将参数`track_counts`设置为`false`。

由于可以使用`pg_dump`将数据传输到较新的Greenplum数据库版本，因此可以预期`pg_dump`的输出将加载到比`pg_dump`的版本更高的Greenplum数据库版本中。`pg_dump`还可以从Greenplum数据库中转储早于其自身版本的版本。但是，`pg_dump`不能从Greenplum数据库版本中转储比其主要版本新的版本。它将拒绝尝试，而不是冒险进行无效的转储。另外，不能保证`pg_dump`的输出可以加载到主版本较旧的服务器上 - 即使转储是从该版本的服务器上获取的也不行。将转储文件加载到旧服务器中可能需要手动编辑转储文件，以删除旧服务器无法理解的语法。在交叉版本的情况下，建议使用`--quote-all-identifiers`选项，因为它可以防止由于不同的Greenplum数据库版本中的保留字列表不同而引起的问题。

示例

将名为`mydb`的数据库转储到SQL脚本文件中：

```
pg_dump mydb > db.sql
```

要将这样的脚本重新加载到名为`newdb`的（新创建的）数据库中：

```
psql -d newdb -f db.sql
```

以tar文件格式转储Greenplum数据库，并包含分发策略信息：

```
pg_dump -Ft --gp-syntax mydb > db.tar
```

要将数据库转储到自定义格式的存档文件中：

```
pg_dump -Fc mydb > db.dump
```

要将数据库转储到目录格式的存档中：

```
pg_dump -Fd mydb -f dumpdir
```

要将数据库与5个工作作业并行转储到目录格式的存档中，请执行以下操作：

```
pg_dump -Fd mydb -j 5 -f dumpdir
```

要将存档文件重新加载到名为newdb的（新创建的）数据库中：

```
pg_restore -d newdb db.dump
```

要转储名为mytab的单个表：

```
pg_dump -t mytab mydb > db.sql
```

要在-t和相关开关中指定大写或大小写混合的名称，您需要将名称加双引号；否则它将被折叠成小写。但是双引号对于shell来说是特殊的，因此反过来必须使用双引号。因此，要转储具有大小写混合名称的单个表，您需要执行以下操作：

```
pg_dump -t '"MixedCaseName"' mydb > mytab.sql
```

另见

[pg_dumpall](#), [pg_restore](#), [psql](#)

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

□ 客户端工具参考

□ 客户端工具摘要

clusterdb

createdb

createlang

createuser

dropdb

droplang

dropuser

pg_config

pg_dump

pg_dumpall

pg_restore

psql

reindexdb

vacuumdb

□ 附加程序

Greenplum database 5管理员
指南

将Greenplum数据库系统中的所有数据库提取到单个脚本文件或其他归档文件中。

概要

```
pg_dumpall [connection-option ...] [dump_option ...]
```

```
pg_dumpall -? | --help
```

```
pg_dumpall -v | --version
```

描述

`pg_dumpall`是一个标准的PostgreSQL工具，用于备份Greenplum数据库（或PostgreSQL）实例中的所有数据库，Greenplum数据库也支持该应用程序。它创建一个（非并行）转储文件。对于Greenplum数据库的常规备份，最好使用Greenplum数据库备份工具[gpbackup](#)以获得最佳性能。

`pg_dumpall`创建一个包含SQL命令的脚本文件，该命令可用作[psql](#)还原数据库的输入。它通过为每个数据库调用`pg_dump`来做到这一点。`pg_dumpall`还转储所有数据库通用的全局对象。

（`pg_dump`不会保存这些对象。）当前包括有关数据库用户和组的信息，以及整体上适用于数据库的访问权限。

由于`pg_dumpall`从所有数据库读取表，因此您很可能必须以数据库超级用户身份连接才能产生完整的转储。另外，您将需要超级用户特权才能执行保存的脚本，以便被允许添加用户和组以及创建数据库。

SQL脚本将被写入标准输出。使用`[-f | --file]`选项或shell运算符将其重定向到文件中。

`pg_dumpall`需要多次连接到Greenplum数据库master服务器（每个数据库一次）。如果您使用密码验证，则很可能每次都要求输入密码。在这种情况下，使用`~/.pgpass`文件很方便。

Note: `--ignore-version`选项已弃用，在以后的版本中将被删除。



选项

转储选项

-a | --data-only

仅转储数据，而不转储模式（数据定义）。此选项仅对纯文本格式有意义。对于归档格式，可以在调用[pg_restore](#)时指定该选项。

-c | --clean

在创建数据库对象（用于创建它们的命令）之前，输出用于清理（删除）数据库对象的命令。此选项仅对纯文本格式有意义。对于归档格式，可以在调用[pg_restore](#)时指定该选项。

-f *filename* | --file= *filename*

将输出发送到指定文件。

-g | --globals-only

仅转储全局对象（角色和表空间），而不转储数据库。

-i | --ignore-version

Note: 此选项已被弃用，并将在以后的版本中删除。

忽略[pg_dump](#)和数据库服务器之间的版本不匹配。[pg_dump](#)可以从运行早期版本的Greenplum数据库（或PostgreSQL）的服务器中转储，但是可能不再支持非常旧的版本。如果您需要覆盖版本检查，请使用此选项。

-o | --oids

转储对象标识符（OID）作为每个表的数据的一部分。对于要还原到Greenplum数据库中的文件，建议不要使用此选项。

-O | --no-owner

不要输出命令来设置对象的所有权以匹配原始数据库。默认情况下，[pg_dump](#)发出ALTER OWNER或SET SESSION AUTHORIZATION语句来设置创建的数据库对象的所有权。除非由超级用户（或拥有脚本中所有对象的同一用户）启动脚本，否则运行脚本时这些语句将失败。要使脚本可以被任何用户恢复，但将赋予该用户所有对象的所有权，请指定-o。此选项仅对纯文本格式有意义。对于归档格式，可以在调用[pg_restore](#)时指定该选项。

-r | --roles-only

仅转储角色，而不是数据库或表空间。

-s | --schema-only

仅转储对象定义（模式），而不转储数据。

-S *username* | --superuser= *username*

指定禁用触发器时要使用的超级用户名。仅在使用-- disable-triggers时才有意义。最好不要这样做，而是以超级用户身份启动生成的脚本。

Note: Greenplum数据库不支持用户定义的触发器。

-t | --tablespaces-only

仅转储表空间，而不是数据库或角色。

-v | --verbose

指定详细模式。这将导致[pg_dump](#)向转储文件输出详细的对象注释和开始/停止时间，并向标准错误输出消息。

-V | --version

打印pg_dumpall版本并退出。

-x | --no-privileges | --no-acl

防止转储访问权限（GRANT/REVOKE命令）。

--binary-upgrade

此选项供就地升级工具使用。不建议或不支持将其用于其他目的。该选项的行为在将来的版本中可能会更改，恕不另行通知。

--column-inserts | --attribute-inserts

将数据转储为带有显式列名（`INSERT INTO table (column, ...)` VALUES ...）的`INSERT`命令。这会使恢复非常缓慢。它主要用于制作可以装入非基于PostgreSQL的数据库的转储。同样，由于此选项为每一行生成一个单独的命令，因此在重新加载行时发生错误只会导致该行丢失，而不是整个表内容丢失。

--disable-dollar-quoting

此选项禁止对函数体使用美元引号，并强制使用SQL标准字符串语法对其进行引用。

--disable-triggers

仅当创建仅数据转储时，此选项才相关。它指示pg_dumpall包含一些命令，以在重新加载数据时临时禁用目标表上的触发器。如果不想在数据重装期间调用的表上有触发器，请使用此选项。为--disable-triggers发出的命令必须以超级用户身份执行。因此，您还应该使用-s指定超级用户名，或者最好小心地以超级用户身份启动生成的脚本。此选项仅对纯文本格式有意义。对于归档格式，可以在调用[pg_restore](#)时指定该选项。

Note: Greenplum数据库不支持用户定义的触发器。

--inserts

将数据转储为`INSERT`命令（而不是`COPY`）。这会使恢复非常缓慢。它主要用于制作可以装入非基于PostgreSQL的数据库的转储。但是，由于此选项为每一行生成一个单独的命令，因此在重新加载行时发生错误只会导致该行丢失，而不是整个表内容丢失。请注意，如果您重新排列了列顺序，则还原可能会完全失败。--column-inserts选项可以安全地防止列顺序更改，尽管速度更慢。

--lock-wait-timeout= *timeout*

不要在转储开始时一直等待获取共享表锁。相反，如果无法在指定的*timeout*时间内锁定表，则失败。可以使用`SET statement_timeout`接受的任何格式指定超时。允许的值因

要从中转储的服务器版本而异，但所有Greenplum数据库版本均接受整数毫秒。

--no-security-labels

不要转储安全标签。

--no-tablespaces

不输出命令以选择表空间。使用此选项，将在还原期间的默认表空间中创建所有对象。

--no-unlogged-table-data

不要转储未记录表的内容。此选项对是否转储表定义（模式）没有影响。它仅禁止转储表数据。从备用服务器转储时，始终排除未记录表中的数据。

--quote-all-identifiers

强制给所有标识符加上引号。当从Greenplum数据库主版本不同于pg_dumpall的中转储数据库时，或者打算将输出加载到其他主版本的服务器中时，建议使用此选项。默认情况下，pg_dumpall仅为在其主版本中为保留字的标识符加引号。在处理其他版本的保留字可能略有不同的服务器时，有时会导致兼容性问题。使用--quote-all-identifiers可以防止此类问题，但代价是难以阅读的转储脚本。

--resource-queues

转储资源队列定义。

--resource-groups

转储资源组定义。

--use-set-session-authorization

输出SQL标准的SET SESSION AUTHORIZATION命令而不是ALTER OWNER命令来确定对象所有权。这使转储更加符合标准，但是依赖转储中对象的历史记录，可能无法正确还原。使用SET SESSION AUTHORIZATION进行的转储将需要超级用户权限才能正确还原，而ALTER OWNER则需要较少的权限。

--gp-syntax

使用--gp-syntax在CREATE TABLE语句中转储Greenplum数据库语法。这允许转储Greenplum数据库表的分发策略(DISTRIBUTED BY或DISTRIBUTED RANDOMLY子句)，这对于还原到其他Greenplum数据库系统很有用。

--no-gp-syntax

不要在CREATE TABLE语句中输出表分配子句。

-? | --help

显示有关pg_dumpall命令行参数的帮助，然后退出。

连接选项

-d *connstr* | --dbname= *connstr*

指定用于连接到服务器的参数，作为连接字符串。有关更多信息，请参见PostgreSQL文档中的[连接字符串](#)。

为了与其他客户端应用程序保持一致，该选项称为--dbname，

但是由于pg_dumpall需要连接到许多数据库，因此连接字符串中的数据库名称将被忽略。使用-1选项可指定用于转储全局对象并发现应转储其他数据库的数据库的名称。

-h host | --host= host

运行Greenplum数据库master数据库服务器的计算机的主机名。如果未指定，则从环境变量PGHOST读取或默认为localhost。

-1 dbname | --database= dbname

指定要在其中连接以转储全局对象的数据库的名称。如果未指定，则使用postgres数据库。如果postgres数据库不存在，则使用template1数据库。

-p port | --port= port

Greenplum master数据库服务器正在侦听连接的TCP端口。如果未指定，则从环境变量PGPORT读取或默认为5432。

-U username | --username= username

要用作连接的数据库角色名称。如果未指定，则从环境变量PGUSER读取或默认为当前系统角色名称。

-w | --no-password

不发出密码提示。如果服务器要求密码验证，而其他方式（例如.pgpass文件）无法使用密码，则连接尝试将失败。此选项在没有用户输入密码的批处理作业和脚本中很有用。

-W | --password

强制输入密码提示。

--role= rolename

指定用于创建转储的角色名称。此选项使pg_dumpall在连接到数据库后发出SET ROLE rolename命令。当通过身份验证的用户（由-U指定）缺少pg_dumpall所需的特权，但可以切换到具有所需权限的角色时，此功能很有用。某些安装有禁止直接以超级用户身份登录的策略，并且使用此选项可以在不违反策略的情况下进行转储。

注解

由于pg_dumpall在内部调用pg_dump，因此某些诊断消息将引用pg_dump。

恢复后，明智的做法是在每个数据库上运行ANALYZE，以便查询优化器可以得到有用的统计信息。您也可以运行vacuumdb -a -z分析所有数据库。

pg_dumpall要求所有需要的表空间目录在还原之前都存在。否则，对于非默认位置的数据库，数据库创建将失败。

示例

转储所有数据库：

```
pg_dumpall > db.out
```

要从此文件重新加载数据库，可以使用：

```
psql template1 -f db.out
```

仅转储全局对象（包括资源队列）：

```
pg_dumpall -g --resource-queues
```

另见

[pg_dump](#)

Greenplum数据库® 6.0文档

- 工具指南

- 管理工具参考

- 客户端工具参考

- 客户端工具摘要

clusterdb

createdb

createlang

createuser

dropdb

droplang

dropuser

pg_config

pg_dump

pg_dumpall

pg_restore

psql

reindexdb

vacuumdb

- 附加程序

Greenplum database 5管理员
指南

从pg_dump创建的存档文件中恢复数据库。

概要

```
pg_restore [connection-option ...] [restore_option ...]  
filename
```

```
pg_restore -? | --help
```

```
pg_restore -v | --version
```

描述

pg_restore是一种工具，用于从[pg_dump](#)创建的非明文格式之一的存档中恢复数据库。它将发出将数据库重建到保存时所处状态的必要命令。存档文件还允许**pg_restore**对恢复的内容具有选择性，甚至可以在恢复之前对项目进行重新排序。

pg_restore可以在两种模式下运行。如果指定了数据库名称，则存档将直接还原到数据库中。否则，将创建一个脚本，该脚本包含重建数据库所需的SQL命令，并将其写入文件或标准输出。脚本输出等效于**pg_dump**的纯文本输出格式。因此，控制输出的某些选项类似于**pg_dump**选项。

pg_restore无法恢复存档文件中不存在的信息。例如，如果使用“将数据作为INSERT命令转储”选项进行归档，则**pg_restore**将无法使用COPY语句加载数据。

Note: **--ignore-version**选项已弃用，在以后的版本中将被删除。

选项

filename

指定要还原的档案文件的位置（或目录，对于目录，[归档](#)）。
如果未指定，则使用标准输入。

恢复选项

-a | --data-only

仅转储数据，而不转储模式（数据定义）。表数据和序列值将转储，如果在归档文件中有的话。

此选项与--section=data相似，但由于历史原因不同。

-c | --clean

在重新创建数据库对象之前，先对其进行清理（删除）。（如果目标数据库中不存在任何对象，则恢复可能会生成一些无害的错误消息。）

-C | --create

在还原到数据库之前先创建它。如果还指定了--clean，请在连接到目标数据库之前删除并重新创建目标数据库。
使用此选项时，以-d命名的数据库仅用于发出初始DROP DATABASE和CREATE DATABASE命令。所有数据都将还原到存档中显示的数据库名称中。

-d *dbname* | --dbname= *dbname*

连接到该数据库，然后直接还原到该数据库。与大多数其他Greenplum数据库工具一样，该工具也使用libpq支持的环境变量。但是，如果未提供数据库名称，它将不会读取PGDATABASE。

-e | --exit-on-error

如果在向数据库发送SQL命令时遇到错误，则退出。默认设置为继续并在还原结束时显示错误计数。

-f *outfilename* | --file= *outfilename*

为生成的脚本或与-1一起使用的列表指定输出文件。默认为标准输出。

-F c|d|t | --format={custom | directory | tar}

`pg_dump`生成的归档文件格式。无需指定格式，因为`pg_restore`将自动确定格式。格式可以是custom, directory或tar。

-i | --ignore-version

Note: 此选项已被弃用，并将在以后的版本中删除。
忽略数据库版本检查。

-I *index* | --index= *index*

仅还原命名索引的定义。

-j | --number-of-jobs | --jobs= *number-of-jobs*

使用多个并发作业运行`pg_restore`中最耗时的部分 - 加载数据，创建索引或创建约束的部分。此选项可以大大减少将大型数据库还原到多处理器计算机上运行的服务器的时间。

每个作业是一个进程或一个线程，具体取决于操作系统，并使用与服务器的单独连接。

此选项的最佳值取决于服务器，客户端和网络的硬件设置。影响因素包括CPU内核数和磁盘设置。一个不错的起点是服务器

上的CPU内核数量，但是在许多情况下，大于该数量的值也可以缩短还原时间。当然，太高的值会由于抖动而导致性能下降。此选项仅支持自定义存档格式。输入文件必须是常规文件（例如，不是管道）。发出脚本而不是直接连接到数据库服务器时，将忽略此选项。另外，不能将多个作业与`--single-transaction`选项一起使用。

`-l | --list`

列出归档的内容。此操作的输出可与`-L`选项一起使用，以限制和重新排序要还原的项目。

`-L list-file | --use-list= list-file`

仅还原*list-file*中的元素，并还原它们在文件中出现的顺序。请注意，如果将`-n`或`-t`之类的过滤开关与`-L`一起使用，它们将进一步限制恢复的项目。

通常，通过编辑先前的`-l`操作的输出来创建*list-file*。可以移动或删除行，也可以通过在行的开头放置分号（;）来注释掉行。请参阅下面的示例。

`-n schema | --schema= schema`

仅还原命名模式中的对象。可以将其与`-t`选项结合使用以仅还原特定表。

`-O | --no-owner`

不要输出命令来设置对象的所有权以匹配原始数据库。默认情况下，`pg_restore`发出`ALTER OWNER`或`SET SESSION AUTHORIZATION`语句来设置所创建模式元素的所有权。除非由超级用户（或拥有脚本中所有对象的同一用户）建立与数据库的初始连接，否则这些语句将失败。使用`-o`，可以将任何用户名用于初始连接，并且该用户将拥有所有创建的对象。

`-P 'function-name(argtype [, ...])' | --function='function-name(argtype [, ...])'`

仅还原命名函数。函数名称必须用引号引起。注意，函数名称和参数的拼写必须与转储文件的目录中的拼写完全相同（如`-list`选项所示）。

`-s | --schema-only`

如果模式条目存在于归档中，则仅还原模式（数据定义），而不还原数据。

此选项与`--data-only`相反。它类似于`--section=pre-data` `--section=post-data`，但由于历史原因不同。

（不要将此与`--schema`选项混淆，该选项以不同的含义使用“schema”一词。）

`-S username | --superuser= username`

指定禁用触发器时要使用的超级用户名。仅在使用`--`

`--disable-triggers`时才有意义。

Note: Greenplum数据库不支持用户定义的触发器。

-t *table* | --table= *table*

仅还原命名表的定义和/或数据。 可以使用多个-t开关指定多个表。 可以将它与-n选项结合使用以指定模式。

-T *trigger* | --trigger= *trigger*

仅还原命名触发器。

Note: Greenplum数据库不支持用户定义的触发器。

-v | --verbose

指定详细模式。

-V | --version

打印pg_restore版本并退出。

-x | --no-privileges | --no-acl

防止转储访问权限 (GRANT/REVOKE命令)。

-1 | --single-transaction

作为单个事务执行还原。 这样可以确保所有命令都成功完成，或者不应用任何更改。

--disable-triggers

仅当创建仅数据恢复时，此选项才相关。 它指示pg_restore在重载数据时执行命令以临时禁用目标表上的触发器。 如果不想在数据重装期间调用的表上有触发器，请使用此选项。 为--disable-triggers发出的命令必须以超级用户身份执行。 因此，您还应该使用-s指定超级用户名，或者最好以超级用户身份运行pg_restore。

Note: Greenplum数据库不支持用户定义的触发器。

--no-data-for-failed-tables

默认情况下，即使表的创建命令失败（例如，因为它已经存在），表数据也将还原。 使用此选项，将跳过此类表的数据。 当目标数据库可能已经包含所需的表内容时，此行为很有用。 指定此选项可防止加载重复或过时的数据。 该选项仅在直接还原到数据库中时才有效，在生成SQL脚本输出时无效。

--no-security-labels

即使存档包含安全标签，也不要输出命令以还原安全标签。

--no-tablespaces

不输出命令以选择表空间。 使用此选项，将在还原期间的默认表空间中创建所有对象。

--section= *sectionname*

仅还原命名节。 节名称可以是pre-data, data或post-data。 可以多次指定此选项以选择多个节。

默认为还原所有节。

--use-set-session-authorization

输出SQL标准的SET SESSION AUTHORIZATION命令而不是ALTER OWNER命令来确定对象所有权。 这使转储更加符合标准，但是依赖转储中对象的历史记录，可能无法正确还原。

-? | --help

显示有关pg_restore命令行参数的帮助，然后退出。

连接选项

-h host | --host host运行Greenplum数据库master数据库服务器的计算机的主机名。
如果未指定，则从环境变量PGHOST读取或默认为localhost。**-p port | --port port**

Greenplum master数据库服务器正在侦听连接的TCP端口。如果未指定，则从环境变量PGPORT读取或默认为5432。

-U username | --username username

要用作连接的数据库角色名称。如果未指定，则从环境变量PGUSER读取或默认为当前系统角色名称。

-w | --no-password

不发出密码提示。如果服务器要求密码验证，而其他方式（例如.pgpass文件）无法使用密码，则连接尝试将失败。此选项在没有用户输入密码的批处理作业和脚本中很有用。

-W | --password

强制输入密码提示。

--role=rolename

指定用于执行还原的角色名称。此选项使pg_restore在连接到数据库后发出SET ROLE rolename命令。当通过身份验证的用户（由-U指定）缺少pg_restore所需的特权，但可以切换到具有所需权限的角色时，此功能很有用。某些安装有禁止直接以超级用户身份登录的策略，并且使用此选项可以在不违反策略的情况下进行还原。

注解

如果您的安装对template1数据库有任何本地添加，请小心将pg_restore的输出加载到一个真正的空数据库中；否则，由于添加对象的重复定义，您很可能会出错。要创建没有任何本地添加的空数据库，请从template0而不是template1复制，例如：

```
CREATE DATABASE foo WITH TEMPLATE template0;
```

当将数据恢复到预先存在的表并使用选项--disable-triggers时，pg_restore在插入数据之前发出命令以禁用用户表上的触发器，然后在插入数据后发出命令以重新启用它们。如果还原在中间停止，则系统catalog可能处于错误状态。

另请参阅pg_dump文档以获取有关pg_dump限制的详细信息。

还原后，明智的做法是在每个还原的表上运行ANALYZE，以便查询优化器具有有用的统计信息。

示例

假设我们已经将名为mydb的数据库转储到了自定义格式的转储文件中：

```
pg_dump -Fc mydb > db.dump
```

删除数据库并从转储中重新创建它：

```
dropdb mydb
pg_restore -C -d template1 db.dump
```

要将转储重新加载到名为newdb的新数据库中。请注意，这里没有-C，而是直接连接到要还原到的数据库。还要注意，我们从template0而不是template1克隆新数据库，以确保它最初是空的：

```
createdb -T template0 newdb
pg_restore -d newdb db.dump
```

要重新排序数据库项，首先必须转储归档的目录：

```
pg_restore -l db.dump > db.list
```

清单文件由一个标题和每个项目的一行组成，例如，

```
; Archive created at Mon Sep 14 13:55:39 2009
;      dbname: DBDEMONS
;      TOC Entries: 81
;      Compression: 9
;      Dump Version: 1.10-0
;      Format: CUSTOM
;      Integer: 4 bytes
;      Offset: 8 bytes
;      Dumped from database version: 8.3.5
;      Dumped by pg_dump version: 8.3.8
;
;      Selected TOC Entries:
;
3; 2615 2200 SCHEMA - public pasha
```

```
1861; 0 0 COMMENT - SCHEMA public pasha
1862; 0 0 ACL - public pasha
317; 1247 17715 TYPE public composite pasha
319; 1247 25899 DOMAIN public domain0 pasha2
```

分号开始注释，行开头的数字指代分配给每个项目的内部档案ID。文件中的行可以被注释掉，删除和重新排序。例如：

```
10; 145433 TABLE map_resolutions postgres
;2; 145344 TABLE species postgres
;4; 145359 TABLE nt_header postgres
6; 145402 TABLE species_records postgres
;8; 145416 TABLE ss_old postgres
```

可以用作pg_restore的输入，并且只能按以下顺序恢复项目10和6：

```
pg_restore -L db.list db.dump
```

另见

[pg_dump](#)

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

□ 客户端工具参考

□ 客户端工具摘要

clusterdb

createdb

createlang

createuser

dropdb

droplang

dropuser

pg_config

pg_dump

pg_dumpall

pg_restore

psql

reindexdb

vacuumdb

□ 附加程序

Greenplum database 5管理员
指南

Greenplum数据库的交互式命令行接口

概要

psql [option ...] [dbname [username]]

描述

psql是Greenplum数据库的基于终端的前端。它使您能够以交互方式键入查询，将其发布到Greenplum数据库，并查看查询结果。或者，输入可以来自文件。此外，它提供了许多元命令和各种类似于shell的功能，以帮助编写脚本和自动化各种任务。

选项

-a | --echo-all

读取时将所有非空输入行打印到标准输出。（这不适用于以交互方式读取的行。）这等效于将变量**ECHO**设置为**all**。

-A | --no-align

切换到不对齐输出模式。（默认输出模式已对齐。）

-c 'command' | --command=' command'

指定**psql**将执行指定的命令字符串，然后退出。这在shell脚本中很有用。**command**必须是服务器可以完全解析的命令字符串，也可以是单个反斜杠命令。因此，您不能将SQL和**psql**元命令与此选项混合使用。为此，您可以将字符串通过管道传递到**psql**中，如下所示：

```
echo '\x \\ SELECT * FROM foo;' | psql
```

（\\是分隔符元命令。）

如果命令字符串包含多个SQL命令，则它们将在单个事务中进行处理，除非字符串中包含明确的**BEGIN/COMMIT**语句。这将将其分为多个事务。这与将相同的字符串提供给**psql**的标准输入时的行为不同。此外，仅返回最后一个SQL命令的结果。

-d dbname | --dbname= dbname

dbname

指定要连接的数据库的名称。这等效于在命令行上将为第一个非选项参数。

如果此参数包含=符号或以有效的URI前缀

(`postgresql://`或`postgres://`) 开头，则将其视为`conninfo`字符串。有关更多信息，请参见PostgreSQL文档中的[连接字符串](#)。

`-e | --echo-queries`

将所有发送到服务器的SQL命令复制到标准输出。

`-E | --echo-hidden`

回显由`\d`和其他反斜杠命令生成的实际查询。您可以使用它来研究`psql`的内部操作。这等效于将变量`ECHO_HIDDEN`设置为`on`。

`-f filename | --file=filename`

使用`filename`作为命令源，而不是交互读取命令。处理完文件后，`psql`终止。在许多方面，这等效于元命令`\i`。

如果文件名是`-`（连字符），则将读取标准输入，直到EOF指示或`\q`元命令为止。但是请注意，在这种情况下不使用`Readline`（就像已指定`-n`一样）。

使用此选项与编写`psql < filename`稍有不同。通常，两者都可以达到您的期望，但是使用`-f`可以启用一些不错的功能，例如带有行号的错误消息。使用此选项还可能会减少启动开销。

另一方面，使用shell的输入重定向的变体在理论上可以保证产生与您手工输入的所有内容完全相同的输出。

`-F separator | --field-separator= separator`

使用指定的分隔符作为未对齐输出的字段分隔符。

`-H | --html`

打开HTML表格输出。

`-l | --list`

列出所有可用的数据库，然后退出。其他非连接选项将被忽略。

`-L filename | --log-file=filename`

除正常输出目标外，还将所有查询输出写入指定的日志文件。

`-n | --no-readline`

不要将`Readline`用于行编辑，也不要使用命令历史记录。剪切和粘贴时，这对于关闭选项卡扩展很有用。

`-o filename | --output= filename`

将所有查询输出放入指定的文件。

`-P assignment | --pset= assignment`

允许您在命令行上以`\pset`样式指定打印选项。请注意，这里必须用等号（而不是空格）分隔名称和值。因此，要将输出格式设置为LaTeX，可以编写`-P format=latex`。

`-q | --quiet`

指定`psql`应该安静地工作。默认情况下，它打印欢迎消息和各种信息输出。如果使用此选项，则不会发生任何情况。这对于`-c`选项很有用。这等效于将变量`QUIET`设置为`on`。

`-R separator | --record-separator= separator`

指定

使用`separator`作为未对齐输出的记录分隔符。

`-s | --single-step`

以单步模式运行。这意味着在将每个命令发送到服务器之前都会提示用户，并且还可以选择取消执行。使用它来调试脚本。

`-S | --single-line`

在单行模式下运行，其中新行像分号一样终止SQL命令。

`-t | --tuples-only`

关闭列名和结果行计数页脚等的打印。此命令等效于`\pset tuples_only`，并且为方便起见而提供。

`-T table_options | --table-attr= table_options`

允许您指定要放置在HTML表格标记内的选项。有关详细信息，请参见`\pset`。

`-v assignment | --set= assignment | --variable= assignment`

执行变量分配，例如`\set meta`命令。请注意，必须在命令行上用等号分隔名称和值（如果有）。要取消设置变量，请取消等号。要将变量设置为空值，请使用等号，但不要使用该值。这些分配是在启动的非常早期阶段完成的，因此保留给内部使用的变量可能会在以后被覆盖。

`-V | --version`

打印psql版本并退出。

`-x | --expanded`

打开扩展表格式模式。

`-X | --no-psqlrc`

不要读取启动文件（系统范围内的`psqlrc`文件或用户的`~/.psqlrc`文件都不能读取）。

`-z | --field-separator-zero`

将未对齐输出的字段分隔符设置为零字节。

`-0 | --record-separator-zero`

将未对齐输出的记录分隔符设置为零字节。这对于例如与`xargs -0`配合非常有用。

`-1 | --single-transaction`

当psql执行脚本时，添加此选项会将`BEGIN/COMMIT`包装在脚本周围，以将其作为单个事务执行。这样可以确保所有命令都成功完成，或者不应用任何更改。

如果脚本本身使用`BEGIN`, `COMMIT`或`ROLLBACK`，则此选项将不会达到预期的效果。另外，如果脚本包含无法在事务块内执行的任何命令，则指定此选项将导致该命令（并因此导致整个事务）失败。

`-? | --help`

显示有关psql命令行参数的帮助，然后退出。

连接选项

`-h host | --host= host`

运行Greenplum数据库master数据库服务器的计算机的主机名。如果未指定，则从环境变量`PHOST`读取或默认为`localhost`。

在master主机上启动psql时，如果host值以斜杠开头，则它将用作UNIX域套接字的目录。

-p port | --port= port

Greenplum master数据库服务器正在侦听连接的TCP端口。如果未指定，则从环境变量PGPORT读取或默认为5432。

-U username | --username= username

要用作连接的数据角色名称。如果未指定，则从环境变量PGUSER读取或默认为当前系统角色名称。

-W | --password

强制输入密码提示。每当服务器请求密码验证时，psql都会自动提示输入密码。但是，当前密码请求检测并不完全可靠，因此此选项会强制提示。如果未发出密码提示，并且服务器要求密码验证，则连接尝试将失败。

-w --no-password

不发出密码提示。如果服务器要求密码验证，而其他方式（例如.pgpass文件）无法使用密码，则连接尝试将失败。此选项在没有用户输入密码的批处理作业和脚本中很有用。

注意：此选项在整个会话中保持设置状态，因此会影响元命令\connect的使用以及初始连接尝试。

退出状态

如果psql正常完成，它将返回0到shell程序；如果发生自身的致命错误（内存不足，找不到文件），则返回1；如果与服务器的连接断开并且会话不是交互式的，则返回2；如果脚本中发生错误，并设置了变量ON_ERROR_STOP，则返回3。

用法

连接到数据库

psql是Greenplum数据库的客户端应用程序。为了连接到数据库，您需要知道目标数据库的名称，Greenplum主服务器的主机名和端口号以及要用作连接的数据角色用户名。可以通过命令行选项（分别为-d，-h，-p和-U）告知psql这些参数。如果找到的参数不属于任何选项，它将被解释为数据库名称（或用户名，如果已经给出数据库名称）。并非所有这些选项都是必需的；有一些可用的默认值。如果省略主机名，则psql将通过UNIX域套接字连接到本地主机上的master服务器，或者通过TCP/IP连接到没有UNIX域套接字的计算机上的localhost。默认的master端口号是5432。如果为master使用其他端口，则必须指定端口。默认数据库用户名是您的操作系统用户名。

默认数据库名也是如此。请注意，您不能仅以任何用户名连接到任何数据库。您的数据库管理员应已将您的访问权限告知您。

当默认值不合适时，可以通过将任何或所有环境变量PGAPPNAME, PGDATABASE, PGHOST, PGPORT和PGUSER设置为适当的值来节省键入时间。

拥有~/.pgpass文件也很方便，可以避免定期输入密码。该文件应位于您的主目录中，并包含以下格式的行：

```
hostname:port:database:username:password
```

.pgpass的权限必须禁止任何其他用户或组的访问（例如：chmod 0600 ~/.pgpass）。如果权限不严格于此权限，则文件将被忽略。（但是，当前未在Microsoft Windows客户端上检查文件权限。）

指定连接参数的另一种方法是在conninfo字符串或URI中，而不是使用数据库名称。这种机制使您可以非常广泛地控制连接。例如：

```
$ psql "service=myservice sslmode=require"  
$ psql postgresql://gpmaster:5433/mydb?sslmode=require
```

这样，您还可以按照PostgreSQL文档中的[LDAP查找连接参数](#)中所述使用LDAP进行连接参数查找。有关所有可用连接选项的更多信息，请参见PostgreSQL文档中的[参数关键字](#)。

如果由于某种原因（权限不足，服务器未运行等）而无法建立连接，则psql将返回错误并终止。

如果标准输入或标准输出中至少有一个是终端，则psql会将客户端编码设置为auto，它将从语言环境设置（Unix系统上的LC_CTYPE环境变量）中检测适当的客户端编码。如果无法按预期进行，则可以使用环境变量PGCLIENTENCODING覆盖客户端编码。

输入SQL命令

在正常操作中，psql会提示您输入当前已连接psql的数据库的名称，后跟字符串=>（对于常规用户）或=#（对于超级用户）。例如：

```
testdb=>  
testdb=#
```

在提示符下，用户可以键入SQL命令。通常，当到达命令终止分号时，会将输入行发送到服务器。行尾不会终止命令。因此，为了清楚

起见，命令可以分布在几行上。如果命令已正确发送和执行，则命令的结果将显示在屏幕上。

如果不受信任的用户可以访问未采用[安全模式使用模式](#)的数据，[请通过从search_path中删除可公开写入的模式来开始会话](#)。您可以在连接字符串中添加options=--csearch_path=或在其他SQL命令之前发出SELECT pg_catalog.set_config('search_path', '' , false)。此注意事项并非特定于pgsql；它适用于执行任意SQL命令的每个接口。

元命令

在pgsql中输入的任何以不带引号反斜杠开头的内容都是pgsql元命令，该命令由pgsql本身处理。这些命令有助于使pgsql对管理或脚本化更加有用。元命令通常被称为斜杠或反斜杠命令。

pgsql命令的格式为反斜杠，后跟命令动词，然后是参数。参数与命令动词之间用任意数量的空格字符分隔。

要将空格包含在参数中，可以用单引号将其引起来。要将单引号包含在这样的参数中，请在单引号文本中写两个单引号。此外，单引号中包含的所有内容都会用类似C的方式替换\n（换行），\t（制表符），\b（退格键），\r（回车），\f（换页），\digits（八进制）和\xdigits（十六进制）。在单引号包围的文本中，反斜杠前导任何其他字符，都表示是该单个字符，不管它是什么。

在一个参数中，加上反勾号(`)的文本被看做是一个命令行，传递给shell。该命令的输出（删除结尾的新行）替换了加上反勾号的文本。

如果一个未加引号的冒号(:)后面跟着一个pgsql变量名，出现在一个参数中，那么它会被变量的值取代，就像[SQL代换](#)中描述的那样。

有些命令以一个SQL标识的名称（比如一个表名）为参数。这些参数遵循SQL语法关于双引号的规则：不带双引号的标识强制成小写，而双引号(")保护字母不进行大小写转换，并且允许在标识符中使用空白。在双引号中，成对的双引号在结果名字中分析成一个双引号。比如，FOO "BAR" BAZ解析成fooBARbaz；而"A weird" " name" 解析成A weird" name。

对参数的分析在行的末尾停止，或者在找到另一个不带引号的反斜杠时停止。一个不带引号的反斜杠会认为是一个新的元命令的开始。特殊的双反斜杠序列\\标识参数的结尾并将继续分析后面的SQL命令（如果存在的话）。这样SQL和pgsql命令可以自由的在一行里面混合。但

是在任何情况下，一条元命令的参数不能延续超过行尾。

下面是已定义的元命令：

\a

如果目前的表输出格式是不对齐的，则切换成对齐的。如果是对齐的，则切换成不对齐。这条命令是为了向后兼容。参阅\pset获取一个更通用的解决方法。

\c | \connect [*dbname* [*username*] [*host*] [*port*]] | *conninfo*

与一个PostgreSQL服务器建立一个新的连接。要使用的连接参数通过位置语法或者使用*conninfo*连接字符串（在[libpq连接字符串](#)描述）指定。

如果该命令省略了数据库名称，用户，主机或端口，则新连接可以重用先前连接中的值。默认情况下，除了处理*conninfo*字符串时，都将重用先前连接的值。传递`-reuse-previous=on`或`-reuse-previous=off`的第一个参数将覆盖该默认值。当命令既未指定也不重用特定参数时，将使用libpq缺省值。将*dbname*, *username*, *host*或*port*中的任何一个指定为`-`等同于省略该参数。

如果成功制作了新连接，那么关闭以前的连接。如果连接失败，那么仅当psql处于交互模式时将保留前面的连接。如果运行的是非交互的脚本，处理会马上停止并返回一个错误。设置这样的区别一方面是为用户使用方便考虑，另一方面也为了保证脚本不会碰巧操作了错误的数据库的安全机制考虑。

示例：

```
=> \c mydb myuser host.dom 6432
=> \c service=foo
=> \c "host=localhost port=5432 dbname=mydb
connect_timeout=10 sslmode=disable"
=> \c postgresql://tom@localhost/mydb?
application_name=myapp
```

\C [*title*]

把正在打印的表的标题设置为一个查询的结果或者取消这样的设置。这条命令等效于\pset title。

\cd [*directory*]

把当前工作目录改变到*directory*。没有参数则是改变到当前用户的家目录。使用\!pwd打印出当前工作目录。

\conninfo

显示有关当前连接的信息，包括数据库名称，用户名，连接类型（UNIX域套接字，TCP/IP等），主机和端口。

\copy { *table* [(*column_list*)] | (*query*) } {from | to} {*filename*' | program '*command*' | *stdin* | *stdout* | *pstdin* | *pstdout*} [with] (*option* [, ...])]

执行前端(客户端)拷贝。这是一个运行SQL [COPY](#)命令的操作，不同的是服务器在读写指明的文件，而psql读写文件并作为本

地的文件系统和服务器之间的跳板取出或写入数据。这意味着文件访问性和权限都是本地用户的，而不是服务器的，因此不需要SQL超级用户权限。

当指定了program时，command是通过psql执行的，并且来自或到达command的数据在服务器和客户端之间传送。再次，文件访问性和权限都是本地用户的，而不是服务器的，因此不需要SQL超级用户权限。

\copy ... from stdin | to stdout分别基于命令输入和输出进行读/写。从发出命令的同一源读取所有行，直到\.为止或流到达EOF。输出被发送到与命令输出相同的位置。要从psql的标准输入或输出读取/写入，请使用psstdin或psstdout。此选项对于在SQL脚本文件中内联填充表很有用。

该命令的语法与SQL COPY命令的语法相似，并且option必须指示SQL COPY命令的选项之一。请注意，因此，特殊的解析规则适用于\copy命令。特别是，变量替换规则和反斜杠转义不适用。

此操作不如SQL COPY命令有效，因为所有数据都必须通过客户端/服务器连接传递。

\copyright

显示Greenplum数据库所基于的PostgreSQL的版权和发行条款。

\d [relation_pattern] | \d+ [relation_pattern] | \dS [relation_pattern]

对于每个匹配pattern的关系(表、视图、索引、序列或外部表)或复合类型，显示所有列、它们的类型、表空间(如果不是缺省的)和任何特殊属性(诸如NOT NULL或缺省等)。相关的索引、约束、规则、触发器也同样显示出来。对于外部表，也显示相关的外部服务器。

- 对于一些关系类型，\d为每个字段显示了额外的信息：序列的字段值，索引的索引表达式和外部表的外部数据封装器选项。
- 命令形式\d+相同，除了显示更多信息：显示与表的列关联的所有注释，以及表中是否存在OID，如果关系是视图，则为视图定义。

对于分区表，用根分区表或子分区表指定的命令\d或\d+显示有关该表的信息，包括分区表当前级别上的分区键。命令\d+还显示表的直接子分区以及该子分区是外部表还是常规表。

对于追加优化的表和面向列的表，\d+显示表的存储选项。对于追加优化的表，将显示该表的选项。对于面向列的表，将为每列显示存储选项。

- 缺省的，只显示用户创建的对象；应用一个模式或s修饰符包含系统对象。

Note: 如果不带任何pattern参数调用\d，那么等效于\dtvsE，将显示一个所有可见表、视图、序列和外部表的

列表。

\da[S] [*aggregate_pattern*]

列出聚合函数，以及它们的返回类型和操作的数据类型。如果声明了pattern，那么只显示匹配模式的聚合函数。缺省的，只显示用户创建的对象；应用一个模式或s修饰符包含系统对象。

\db[+] [*tablespace_pattern*]

列出所有可用的表空间及其对应的路径。如果指定了模式，则仅显示名称与模式匹配的表空间。如果在命令名称后附加+，则会列出每个对象及其关联的权限。

\dc[S+] [*conversion_pattern*]

列出字符集编码之间的转换。如果指定了模式，则仅列出名称与该模式匹配的转换。默认情况下，仅显示用户创建的对象。提供一个模式或s修饰符以包含系统对象。如果在命令名称后附加+，则会列出每个对象及其相关描述。

\dC[+] [*pattern*]

列出类型转换。如果指定了模式，则仅列出其源或目标类型与该模式匹配的转换。如果在命令名称后附加+，则会列出每个对象及其相关描述。

\dd[S] [*pattern*]

显示类型constraint, operator class, operator family, rule和trigger的对象的描述。所有其他注释都可以通过相应的反斜杠命令查看那些对象类型。

\dd显示与模式匹配的对象的描述，或者如果没有给出参数，则显示适当类型的可见对象的描述。但无论哪种情况，仅列出具有描述的对象。默认情况下，仅显示用户创建的对象。提供一个模式或s修饰符以包含系统对象。

可以使用COMMENT SQL命令创建对象的描述。

\ddp [*pattern*]

列出默认的访问权限设置。将为每个角色（和模式，如果适用）显示一个条目，其默认权限设置已从内置默认值更改。如果指定了pattern，则仅列出角色名称或模式名称与模式匹配的条目。

[ALTER DEFAULT PRIVILEGES](#) 命令用于设置默认访问权限。特权显示的含义在[GRANT](#)下进行了说明。

\dD[S+] [*domain_pattern*]

列出域。如果指定了模式，则仅显示名称与该模式匹配的域。默认情况下，仅显示用户创建的对象。提供一个模式或s修饰符以包含系统对象。如果在命令名称后附加+，则会列出每个对象及其关联的权限和描述。

\dE\i\st\p\{S+\} [*external_table | index | sequence | table | parent table | view*]

这不是实际的命令名称：字母E, i, s, t, p和v分别代表外部表，索引，序列，表，父表和视图。您可以按任何顺序指定任何或所有这些字母，以获得这些类型的对象的列表。例如，\dit列出索引和表。如果在命令名称后附加+，则会列出

每个对象及其在磁盘上的物理大小及其相关的描述（如果有）。如果指定了模式，则仅列出名称与该模式匹配的对象。默认情况下，仅显示用户创建的对象。提供一个模式或s修饰符以包含系统对象。

`\des[+] [foreign_server_pattern]`

列出外部服务器。如果指定了模式，则仅列出名称与该模式匹配的服务器。如果使用\des+形式，则会显示每个服务器的完整描述，包括服务器的ACL，类型，版本，选项和描述。

`\det[+] [foreign_table_pattern]`

列出所有外部表。如果指定了模式，则仅列出其表名或模式名称与该模式匹配的条目。如果使用\det+形式，则还将显示通用选项和外部表描述。

`\deu[+] [user_mapping_pattern]`

列出用户映射。如果指定了模式，则仅列出其用户名与模式匹配的那些映射。如果使用\deu+形式，则会显示有关每个映射的其他信息。

Warning: \deu+可能也显示远程用户的用户名和密码，所以要小心不要透漏它们。

`\dew[+] [foreign_data_wrapper_pattern]`

列出外部数据包装器。如果指定了模式，则仅列出名称与该模式匹配的数据包装器。如果使用\dew+形式，则还将显示ACL，选项和外部数据包装程序的描述。

`\df[antwS+] [function_pattern]`

列出函数以及它们的参数，返回类型和函数类型，这些函数分为“agg”（聚合），“normal”，“trigger”或“window”。要仅显示特定类型的函数，请在命令中添加相应的字母a，n，t或w。如果指定了模式，则仅显示名称与该模式匹配的函数。如果使用\df+形式，则会显示有关每个函数的其他信息，包括安全性，易变性，语言，源代码和描述。默认情况下，仅显示用户创建的对象。提供一个模式或s修饰符以包含系统对象。

`\dF[+] [pattern]`

列出文本搜索配置。如果指定了模式，则仅显示名称与该模式匹配的配置。如果使用\dF+形式，则会显示每个配置的完整描述，包括基础文本搜索解析器和每种解析器令牌类型的字典列表。

`\dFd[+] [pattern]`

列出文本搜索词典。如果指定了模式，则仅显示名称与该模式匹配的词典。如果使用\dFd+形式，则会显示有关每个所选词典的其他信息，包括基础的文本搜索模板和选项值。

`\dFp[+] [pattern]`

列出文本搜索解析器。如果指定了模式，则仅显示名称与模式匹配的解析器。如果使用\dFp+形式，则会显示每个解析器的完整说明，包括基础函数和已识别令牌类型的列表。

`\dFt[+] [pattern]`

列出文本搜索模板。如果指定了模式，则仅显示名称与该模式匹

配的模板。如果使用`\dFt+`形式，则会显示有关每个模板的其他信息，包括基础函数名称。

`\dg[+] [role_pattern]`

列出数据库角色。（由于将“用户”和“组”的概念统一为“角色”，因此该命令现在等效于`\du`。）如果指定了模式，则仅列出名称与模式匹配的角色。如果使用`\dg+`形式，则将显示有关每个角色的其他信息；否则，将显示其他信息。目前，这会为每个角色添加注释。

`\dl`

这是`\lo_list`的别名，其中显示了大对象列表。

Note: Greenplum数据库不支持PostgreSQL 大对象工具 来流存储在大对象结构中的用户数据。

`\dL[S+] [pattern]`

列出过程语言。如果指定了模式，则仅列出名称与该模式匹配的语言。默认情况下，仅显示用户创建的语言。提供s修饰符以包括系统对象。如果在命令名称后附加+，则将列出每种语言及其调用处理程序，验证程序，访问特权以及是否为系统对象。

`\dn[S+] [schema_pattern]`

列出所有可用的模式（名称空间）。如果指定了模式，则仅列出名称与模式匹配的模式。默认情况下，仅显示用户创建的对象。提供一个模式或s修饰符以包含系统对象。如果在命令名后附加+，则列出每个对象及其相关的权限和描述（如果有）。

`\do[S] [operator_pattern]`

列出可用的运算符及其操作数和返回类型。如果指定了模式，则仅列出名称与模式匹配的运算符。默认情况下，仅显示用户创建的对象。提供一个模式或s修饰符以包含系统对象。

`\dO[S+] [pattern]`

列出排序规则。如果指定了模式，则仅列出名称与模式匹配的排序规则。默认情况下，仅显示用户创建的对象。提供一个模式或s修饰符以包含系统对象。如果在命令名称后附加+，则将列出每个排序规则及其关联的描述（如果有）。请注意，仅显示可与当前数据库的编码一起使用的排序规则，因此在同一安装的不同数据库中，结果可能会有所不同。

`\dp [relation_pattern_to_show_privileges]`

列出具有相关访问权限的表，视图和序列。如果指定了模式，则仅列出名称与该模式匹配的表，视图和序列。

`GRANT`和`REVOKE`命令用于设置访问权限。特权显示的含义在`GRANT`下进行了说明。

`\drds [role-pattern [database-pattern]]`

列出定义的配置设置。这些设置可以是角色特定的，数据库特定的或两者。`role-pattern`和`database-pattern`分别用于选择特定角色和要列出的数据库。如果省略，或者指定*，则列出所有设置，包括分别不是角色特定的或数据库特定的设置。

`ALTER ROLE`和`ALTER DATABASE`命令用于定义每个角色和每个数据库的角色配置设置。

\dT[S+] [*datatype_pattern*]

列出数据类型。如果指定了模式，则仅列出名称与模式匹配的类型。如果在命令名称后附加+，则将列出每种类型及其内部名称和大小，如果是枚举类型则允许的值及其关联的权限。默认情况下，仅显示用户创建的对象。提供一个模式或s修饰符以包含系统对象。

\du[+] [*role_pattern*]

列出数据库角色。（由于将“用户”和“组”的概念统一为“角色”，因此此命令现在等效于\dg。）如果指定了模式，则仅列出名称与模式匹配的角色。如果使用\du+形式，则会显示有关每个角色的其他信息；目前，这会为每个角色添加注释。

\dx[+] [*extension_pattern*]

列出已安装的扩展。如果指定了模式，则仅列出名称与该模式匹配的扩展名。如果使用\dx+形式，则列出属于每个匹配扩展的所有对象。

\dy[+] [*pattern*]

列出事件触发器。如果指定了模式，则仅列出其名称与该模式匹配的触发器。如果在命令名称后附加+，则会列出每个对象及其相关描述。

Note: Greenplum数据库不支持用户定义的触发器。

\e | \edit [*filename*] [*line_number*]

如果指定*filename*，则文件被编辑；编辑器退出后，其内容将复制回查询缓冲区。如果未提供*filename*，则将当前查询缓冲区复制到一个临时文件，然后以相同的方式对其进行编辑。

然后根据psql的常规规则重新解析新的查询缓冲区，其中整个缓冲区被视为一行。（因此，您不能以这种方式制作脚本。为此请使用\i。）这也意味着，如果查询以分号结尾（或包含分号），则会立即执行该查询。在其他情况下，它将仅在查询缓冲区中等待；输入分号或\g发送，或\r取消。

如果指定了行号，则psql将光标定位在文件或查询缓冲区的指定行上。请注意，如果给出单个全数字参数，则psql假定它是行号，而不是文件名。

有关配置和自定义编辑器的信息，请参见[环境变量](#)。

\echo *text* [...]

将参数输出到标准输出，中间用一个空格分隔，后跟换行符。这对于在脚本输出中散布信息很有用。如果第一个参数是未加引号的-n，则不写结尾的换行符。

Note: 如果使用\o命令重定向查询输出，则可能希望使用\qecho代替此命令。

\ef [*function_description* [*line_number*]]

该命令以CREATE OR REPLACE FUNCTION命令的形式获取并编辑命名函数的定义。编辑的方式与\edit相同。编辑器退出后，更新的命令在查询缓冲区中等待；输入分号或\g发送，或\r取消。

目标函数可以仅通过名称指定，也可以通过名称和参数指定，例

如`foo(integer, text)`。如果有多个具有相同名称的函数，则必须给出参数类型。

如果未指定任何函数，则会显示空白的CREATE FUNCTION模板以供编辑。

如果指定了行号，则psql将光标定位在函数主体的指定行上。

(请注意，函数主体通常不在文件的第一行开始。)

有关配置和自定义编辑器的信息，请参见[环境变量](#)。

`\encoding [encoding]`

设置客户端字符集编码。如果不带参数，此命令将显示当前编码。

`\f [field_separator_string]`

为未对齐的查询输出设置字段分隔符。默认值为竖线（|）。另请参阅\pset以获取设置输出选项的通用方法。

`\g [filename]`

`\g [| command]`

将当前查询输入缓冲区发送到服务器，并有选择地将查询的输出存储到`filename`中，或将输出通过管道传递给shell命令`command`。仅当查询成功返回零个或多个元组时，才写入文件或命令，如果查询失败或是不返回数据的SQL命令，则不写入文件或命令。

裸`\g`本质上等效于分号。带参数的`\g`是`\o`命令的一次性替代方案。

`\gset [prefix]`

将当前查询输入缓冲区发送到服务器，并将查询的输出存储到psql变量中。要执行的查询必须恰好返回一行。该行的每一列都存储在一个单独的变量中，该变量与该列相同。例如：

```
=> SELECT 'hello' AS var1, 10 AS var2;
-> \gset
=> \echo :var1 :var2
hello 10
```

如果指定`prefix`，则在查询的列名称之前添加该字符串以创建要使用的变量名称：

```
=> SELECT 'hello' AS var1, 10 AS var2;
-> \gset result_
=> \echo :result_var1 :result_var2
hello 10
```

如果列结果为NULL，那么对应的变量是未设置的。

如果查询失败或不返回一行，则不会更改任何变量。

`\h | \help [sql_command]`

提供有关指定SQL命令的语法帮助。如果未指定命令，则psql将列出所有可使用语法帮助的命令。如果`command`是星号（*），则会显示所有SQL命令的语法帮助。为了简化键入，

不必引用由几个单词组成的命令。

\H | \html

打开HTML查询输出格式。如果HTML格式已打开，则将其切换回默认的对齐文本格式。此命令是为了兼容性和方便起见，但有关设置其他输出选项的信息，请参见\pset。

\i | \include *filename*

从文件*filename*中读取输入，并像在键盘上键入文件一样执行它。

如果*filename*是-（连字符），则将读取标准输入，直到EOF指示或\q元命令为止。这可用于将交互式输入与文件输入进行穿插。请注意，只有在最外层处于活动状态时，才会使用Readline行为。

如果要在阅读时看到屏幕上的行，必须将变量ECHO设置为all。

\ir | \include_relative *filename*

\ir命令类似于\i，但是以不同的方式解析相对文件名。在交互模式下执行时，两个命令的行为相同。但是，从脚本调用时，\ir会相对于脚本所在目录而不是当前工作目录来解释文件名。

\l[+] | \list[+] [*pattern*]

列出服务器中的数据库，并显示其名称，所有者，字符集编码和访问特权。如果指定了模式，则仅列出名称与该模式匹配的数据库。如果在命令名称后附加+，则还会显示数据库大小，默认表空间和描述。（大小信息仅适用于当前用户可以连接到的数据库。）

\lo_export *loid filename*

从数据库读取OID *loid*的大对象，并将其写入*filename*。请注意，这与服务器函数lo_export稍有不同，后者以数据库服务器运行的用户权限和服务器文件系统上的用户权限运行。使用\lo_list找出大对象的OID。

Note: Greenplum数据库不支持PostgreSQL大对象工具 来流存储在大对象结构中的用户数据。

\lo_import *large_object_filename [comment]*

将文件存储到大对象中。（可选）它将给定的注释与对象相关联。例：

```
mydb=> \lo_import '/home/gpadmin/pictures/photo.xcf'
'a
picture of me'
lo_import 152801
```

该响应表明大对象接收到对象ID 152801，如果要再次访问该对象，应该记住该对象。因此，建议始终将可读的注释与每个对象相关联。然后可以使用\lo_list命令查看这些内容。请注意，此命令与服务器端lo_import略有不同，因为它充当本地文件系统上的本地用户，而不是服务器的用户和文件系统。

Note: Greenplum数据库不支持PostgreSQL大对象工具 来流存储

在大对象结构中的用户数据。

\lo_list

显示当前存储在数据库中的所有大型对象的列表，以及为其提供的任何注释。

Note: Greenplum数据库不支持PostgreSQL大对象工具 [来流存储](#) 在大对象结构中的用户数据。

\lo_unlink *largeobject_oid*

从数据库中删除指定OID的大对象。使用\lo_list找出大对象的OID。

Note: Greenplum数据库不支持PostgreSQL大对象工具 [来流存储](#) 在大对象结构中的用户数据。

\o [\out [*filename*]

\o [\out [| *command*]

将将来的查询结果保存到*filename*，或将将来的结果通过管道传递给shell命令*command*。如果未指定任何参数，则查询输出将重置为标准输出。查询结果包括从数据库服务器获得的所有表，命令响应和通知，以及查询数据库的各种反斜杠命令的输出（例如\d），但不包括错误消息。要在查询结果之间插入文本输出，请使用\qecho。

\p

将当前查询缓冲区打印到标准输出。

\password [*username*]

更改指定用户（默认情况下为当前用户）的密码。该命令提示输入新密码，对其进行加密，然后将其作为ALTER ROLE命令发送到服务器。这样可以确保新密码不会在命令历史记录，服务器日志或其他地方以明文形式出现。

\prompt [*text*] *name*

提示用户提供文本，该文本已分配给变量*name*。可以指定一个可选的提示字符串，*text*。（对于多字提示，请用单引号将文本引起来。）

默认情况下，\prompt使用终端进行输入和输出。但是，如果使用-f命令行开关，\prompt将使用标准输入和标准输出。

\pset [*print_option* [*value*]]

此命令设置影响查询结果表输出的选项。*print_option*描述了要设置的选项。*value*的语义取决于所选选项。对于某些选项，省略*value*会导致该选项被切换或取消设置，如特定选项所述。如果未提及此类行为，则省略*value*只会导致显示当前设置。没有任何参数的\pset将显示所有打印选项的当前状态。

可调打印选项包括：

- **border – value**必须是一个数字。通常，数字越大，表就越宽的边界和越多的线，但是这个参数取决于实际的格式。在HTML格式中，这个参数会直接翻译成border=...属性，在其它的格式中，只有值0(无边界)、1(内部分隔线)、2(表框架)有意义。`latex`和`latex-longtable`也支持border值为3，在每行之间添加一个分隔线。

- **columns** – 设置wrapped格式的目标宽度，以及用于确定输出是否足够宽以要求分页或在扩展自动模式下切换到垂直显示的宽度限制。默认值为零。零导致目标宽度由环境变量COLUMNS控制，如果未设置COLUMNS，则检测到的屏幕宽度。另外，如果columns为零，则wrapped格式仅影响屏幕输出。如果columns不为零，则文件和管道输出也将被包装为该宽度。
- 设置目标宽度后，请使用\pset format wrapped命令启用包装格式。
- **expanded | x** – 如果指定了value，则必须是on或off，这将启用或禁用扩展模式或auto。如果省略value，该命令将在on和off设置之间切换。启用扩展模式后，查询结果将显示在两列中，列名在左侧，数据在右侧。如果在正常的“水平”模式下数据无法显示在屏幕上，则此模式很有用。在auto设置中，只要查询输出比屏幕宽，就使用扩展模式，否则使用常规模式。auto设置仅在对齐和包装的格式中有效。在其他格式下，它始终表现为关闭扩展模式。
 - **fieldsep** – 指定在未对齐输出模式下使用的字段分隔符。这样，就可以创建制表符或逗号分隔的输出，而其他程序可能更喜欢这种输出。要将选项卡设置为字段分隔符，请键入\pset fieldsep '\t'。默认字段分隔符为'|'（竖线）。
 - **fieldsep_zero** - 设置非对齐输出格式中使用的域分隔符为0字节。
 - **footer** – 如果指定了value，则必须将其打开或关闭，这将启用或禁用表脚的显示（(n行)计数）。如果省略value，则该命令将打开或关闭页脚显示。
 - **format** – 将输出格式设置为unaligned, aligned, html, latex (使用tabular), latex-longtable, troff-ms或wrapped之一。允许使用唯一的缩写，包括一个字母。**unaligned**格式将一行的所有列写在一行上，由当前活动的字段分隔符分隔。这对于创建可能打算由其他程序读取的输出（例如，制表符分隔或逗号分隔的格式）很有用。**aligned**格式是标准的，人类可读的，格式良好的文本输出；这是默认值。
html, **latex**, **latex-longtable**和**troff-ms**模式把表输出为可用于文档里的对应标记语言。它们还不是完整的文档！（可能对于HTML变化还不是太大，但是在LaTeX中，您必须具有完整的文档包装器。**latex-longtable**也需要LaTeX longtable和booktabs软件包。）
wrapped格式类似于aligned，但是在行之间包装宽数据值

以使输出适合目标列宽。 目标宽度是根据columns选项下所述确定的。 请注意，psql不会尝试包装列标题。 如果列标题所需的总宽度超过目标，则wrapped格式与aligned的行为相同。

- **linestyle [unicode | ascii | old-ascii]** – 将边界线绘制样式设置为unicode, ascii或old-ascii中的一种。 三种样式允许使用唯一的缩写，包括一个字母。 默认设置为ascii。 此选项仅影响aligned和wrapped的输出格式。
 - ascii** – 使用纯ASCII字符。 数据中的换行符在右边距中使用+符号显示。 当换行格式将数据从一行换行到下一行而没有换行符时，在第一行的右边距中显示一个点(.)，在下一行的左边距中再次显示一个点(.)。
 - old-ascii** – 样式使用纯ASCII字符，并使用PostgreSQL 8.4及更早版本中使用的格式样式。 数据中的换行符用:符号代替左侧的列分隔符。 当数据从一行到另一行的换行而没有换行符时，;符号用于代替左侧的列分隔符。
 - unicode** – 样式使用Unicode框画字符。 数据中的换行符在右边缘使用回车符显示。 当数据从一行换行到下一行而没有换行符时，在第一行的右边距中显示省略号，并在下一行的左边距中显示省略号。
 当border设置大于零时，此选项还确定绘制边框线所使用的字符。 纯ASCII字符在任何地方都可以使用，但是Unicode字符在可以识别它们的显示器上看起来更好。
- **null 'string'** – 第二个参数是一列为空时要打印的字符串。 默认设置为不打印任何内容，这很容易被误认为是空字符串。 例如，可能更喜欢\pset null '(null)'。
- **numericlocale** – 如果指定了value，则必须将其打开或关闭，这将启用或禁用显示特定于语言的字符以将小数点标记左侧的数字组分开。 如果省略value，则该命令在常规和特定于区域的数字输出之间切换。
- **pager** – 控制用于查询和psql帮助输出的分页器程序的使用。 如果设置了环境变量PAGER，则将输出通过管道传递到指定的程序。 否则，将使用依赖于平台的默认值（例如more）。 禁用时，不使用分页器程序。 启用时，仅在适当的时候使用分页器，即当输出到终端且不匹配屏幕时。 也可以将分页器设置为always，这将使该分页器用于所有终端输出，而不管其是否适合屏幕显示。 \pset pager (不带value) 可打开和关闭分页器的使用。
- **recordsep** – 声明在非对齐模式时的记录分隔符。 缺省是换行符。
- **recordsep_zero** - 设置在非对齐输出格式中使用的记录分隔符为0字节。

- **tableattr** | **T** [*text*] – 在HTML格式下，这指定了要放置在table标签中的属性。例如cellpadding或bgcolor。请注意，你可能不需要在这里声明border，因为已经在\pset border里用过了。如果没有给出*value*，那么表的属性是未设置的。
在latex-longtable格式中，这控制每个包含左对齐数据类型的字段的比例宽度。它声明为空白分隔的值列表，比如'0.2 0.2 0.6'。未指定的输出列使用最后指定的值。
- **title** [*text*] – 为任何随后打印的表设置标题。这个参数可以用于给你的输出一个描述性标记。如果没有给出*value*，则标题未设置。
- **tuples_only** | **t** [*novalue* | **on** | **off**] – 如果指定了*value*，那么必须是**on**或**off**，这将启用或禁用仅元组模式。如果省略了*value*，那么该命令在普通和仅元组输出间切换。普通输出包括额外的信息，比如列头、标题、各种脚注等。在仅元组模式下，只显示实际的表数据。**\t**命令等效于\pset tuples_only，并且为方便起见而提供。

Tip:

有很多用于\pset的快速命令。参阅\a, \C, \f, \H, \t, \T和\x。

\q | **\quit**

退出psql程序。在脚本文件中，仅终止该脚本的执行。

\qecho *text* [...]

此命令与\echo相同，除了输出将被写入\o设置的查询输出通道外。

\r | **\reset**

重置（清除）查询缓冲区。

\s [*filename*]

将psql的命令行历史记录打印到*filename*。如果省略*filename*，则将历史记录写入标准输出（如果合适，使用分页器）。如果psql是在没有Readline支持的情况下构建的，则此命令不可用。

\set [*name* [*value* [...]]]

将psql变量*name*设置为*value*，或者如果给定多个值，则将其全部串联。如果仅给出一个参数，则该变量将设置为空值。要取消设置变量，请使用\unset命令。

没有任何参数的\set将显示所有当前设置的psql变量的名称和值。

有效的变量名称可以包含字符，数字和下划线。请参阅[高级特性](#)中的“变量”。变量名称区分大小写。

尽管你可以设置任何变量为任意值，psql对一些变量特殊对待。它们在关于变量的小节里面有文档。
该命令与SQL命令[SET](#)无关。

\setenv *name* [*value*]

将环境变量*name*设置为*value*, 或者如果未提供该*value*, 则取消设置环境变量。例:

```
testdb=> \setenv PAGER less
testdb=> \setenv LESS -imx4F
```

\sf[+] *function_description*

该命令以CREATE OR REPLACE FUNCTION命令的形式获取并显示命名函数的定义。 定义将打印到\o设置的当前查询输出通道。

目标函数可以仅通过名称指定, 也可以通过名称和参数指定, 例如foo(integer, text)。如果有多个相同名称的函数, 则必须提供参数类型。

如果在命令名称后附加+, 则对输出行进行编号, 函数主体的第一行为行¹。

\t [novalue | on | off]

\t命令本身可切换输出列名称标题和行计数页脚的显示。无论当前设置如何, 打开和关闭的值都会设置元组显示。此命令等效于\pset tuples_only, 并且为方便起见而提供。

\T *table_options*

声明HTML输出格式中放在table标记里的属性。此命令等效于\pset tableattr *table_options*

\timing [novalue | on | off]

如果没有参数, 则以毫秒为单位切换显示每个SQL语句花费的时间。无论当前设置如何, 打开和关闭的值都会设置时间显示。

\unset *name*

取消设置(删除)psql变量*name*。

\w | \write *filename*\w | \write | *command*

将当前查询缓冲区输出到*filename*或将其通过管道传递给shell命令*command*。

\watch [*seconds*]

重复执行当前查询缓冲区(如\g), 直到被中断或查询失败。

在两次执行之间等待指定的秒数(默认为2)。

\x [on | off | auto]

设置或切换扩展表格式设置模式。因此, 它等效于\pset expanded。

\z [*pattern*]

列出具有相关访问权限的表, 视图和序列。如果指定了模式, 则仅列出名称与该模式匹配的表, 视图和序列。这是\dp的别名。

\! [*command*]

转义到单独的shell或执行shell命令*command*。参数不会被进一步解释, shell将看到全部参数。特别是, 变量替换规则和反斜杠转义不适用。

\?

显示有关psql反斜杠命令的帮助信息。

模式

各种\d命令都接受一个pattern参数，声明要显示的对象名字。最简单的情况下pattern正好等于对象的名字。pattern中的字符通常会被自动转换成小写，就像SQL名字一样。例如\dt Foo将显示名为foo的表。与在SQL名字中相同的是双引号界定的pattern将保持原样(不做大小写转换)。如果需要在双引号界定的pattern中使用双引号字符，你可以写两个并列的双引号，这与SQL的引号规则相同。例如，\dt "FOO" "BAR" 将会显示名为FOO"BAR的表，但是不会显示foo"bar。与一般的SQL名字规则不同的是，你可以仅用双引号界定名字的一部分，例如\dt FOO"FOO"BAR将显示名为fooFOObar的表。

在模式中，*匹配任何字符序列（包括无字符）和?匹配任何单个字符。（此表示法可与UNIX Shell文件名模式相提并论。）例如，\dt int*显示名称以int开头的所有表。但是在双引号内，*和?失去了这些特殊含义，只是字面上的匹配。

包含点(.) 的模式被解释为模式名称模式，后跟对象名称模式。例如，\dt foo*.bar*将显示其表名以bar开头的所有表，这些表的模式名以foo开头。如果没有点出现，则该模式仅匹配在当前模式搜索路径中可见的对象。同样，双引号中的点失去其特殊含义，并且在字面上匹配。

高级用户可以使用正则表达式。除了.如上所述被当做分隔符，*将被理解成.*，?将被理解成.，所有正则表达式特殊字符均按[PostgreSQL正则表达式文档](#)的规定工作。这样一来，你就可以用?代替.、用(R+|)代替R*、用(R|)代替R?。请记住，模式必须匹配全名，这与对正则表达式的通常解释不同；如果您不希望锚定模式，请在开头和/或结尾处输入*。请注意，在双引号中，所有正则表达式特殊字符都会失去其特殊含义，并且会在字面上进行匹配。同样，正则表达式特殊字符在运算符名称模式（例如\do的参数）中按字面值进行匹配。

每当完全省略pattern参数时，\d命令都会显示在当前模式搜索路径中可见的所有对象 – 这等效于使用模式*。要查看数据库中的所有对象，请使用模式*.*。

高级特性

变量

psql提供类似通常Unix命令shell那样的变量替换特性。 变量只是简单的名称/值对，这里的值可以是任何长度的任何值。 名字必须由字母（包括非拉丁字母）、数字和下划线组成。

要设置一个变量，使用psql元命令\set。例如：

```
testdb=> \set foo bar
```

把变量`foo`的值设置为`bar`。 要检索变量的内容，在变量名前面放上冒号，例如：

```
testdb=> \echo :foo
bar
```

这在SQL命令和元命令中都能运行；更详细信息在下面的[SQL代换](#)中给出。

如果你不带第二个参数调用\set，那么设置这个变量，带有一个空字符串作为`value`。要重置(也就是删除)一个变量，使用\unset命令。要显示所有变量的值，不带有任何参数调用\set。

Note: \set的参数服从和其它命令一样的替换规则。因此你可以构造有趣的引用，像\set :foo 'something'这样，获得分别像Perl或PHP那样有名的"软连接"或"变量变量"。不幸的是，用这些构造不能做任何有用的事情。另一方面，\set bar :foo是一个非常有效的拷贝变量的方法。

有一些常用变量被psql特殊对待。它们代表特定的选项设置，这些选项在运行时可以通过改变变量的值而改变，或者在某些情况下代表psql的可变状态。尽管你可以把这些变量用于其它用途，但是不鼓励这么做，因为程序的行为可能会变得非常奇怪。通常，所有特殊对待的变量名都是由大写ASCII字母组成(可能还有数字和下划线)。为了保证和未来的最大限度的兼容性，请避免使用这样的变量。下面是一个所有特殊对待的变量列表。

AUTOCOMMIT

启用时（默认），成功完成后将自动提交每个SQL命令。要以这种方式推迟提交，必须输入BEGIN或START TRANSACTION SQL命令。禁用或未设置时，除非明确发出COMMIT或END，否则不会提交SQL命令。自动提交模式的工作方式是：在任何尚未在事务块中且本身不是BEGIN或其他事务控制命令的命令或无法在事务块内部执行的命令之前（例如VACUUM），为您发出隐式BEGIN。

在自动提交模式下，必须通过输入ABORT或ROLLBACK明确放弃任何失败的事务。另外请记住，如果您退出会话而不提交，则您的工作将会丢失。

自动提交模式是PostgreSQL的传统行为，但是自动提交模式更接近SQL规范。如果您喜欢关闭自动提交，在`~/.psqlrc`文件中进行设置。

`COMP_KEYWORD_CASE`

在完成SQL关键字时决定哪个字母使用大小写。如果设置为lower或upper，则完成的单词将分别为小写或大写。如果设置为preserve-lower或preserve-upper（缺省），那么完成的单词将是输入时的情况，但是单词在没有任何输入的情况下完成，将分别是小写或大写的情况。

`DBNAME`

正在连接着的数据库名称。每次与一个数据库联结都会设置这个值（包括程序启动），但是可以删除。

`ECHO`

如果设为all，那么所有非空输入行在读取时都回显到标准输出。（这不适用于交互式读取的行。）使用-a选项声明在程序启动时就默认这样做。如果设置为queries，那么psql只是在查询发送给服务器时打印到标准输出。这个开关是-e。

`ECHO_HIDDEN`

当此变量设置为on并且反斜杠命令查询数据库时，将首先显示该查询。此特性可帮助您研究Greenplum数据库的内部结构，并在您自己的程序中提供类似的功能。（要在程序启动时选择此行为，请使用开关-E。）如果将变量设置为值noexec，则仅显示查询，但实际上并未将其发送到服务器并执行查询。

`ENCODING`

当前的客户端字符集编码。

`FETCH_COUNT`

如果将此变量设置为大于0的整数值，则将提取SELECT查询的结果并将其显示在这么多行的组中，而不是在显示之前收集整个结果集的默认行为。因此，无论结果集的大小如何，仅使用有限的内存量。启用此功能时，通常使用100到1000的设置。请记住，使用此功能时，在显示了一些行后查询可能会失败。

尽管可以使用此特性使用任何输出格式，但是默认的对齐格式看起来很糟糕，因为每组`FETCH_COUNT`行将分别设置格式，从而导致各行组的列宽变化。不过对于其他格式这个特性工作的很好。

`HISTCONTROL`

如果将这个变量设为ignorespace，那么以空格开始的行将不会进入历史列表。如果设置为ignoredups，那么与以前历史记录里匹配的行也不会进入历史记录。值ignoreboth是上面两个的结合。如果删除此变量或者其值为任何与上面的值不同的东西，所有交互模式读入的行都被保存入历史列表。

`HISTFILE`

此文件将用于存储历史列表。 默认值是`~/.pgsql_history`。

例如，在`~/.pgsqlrc`里使用：

```
\set HISTFILE ~/.pgsql_history- :DBNAME
```

将使得psql为每个数据库维护一个独立的历史。

HISTSIZE

保存在命令历史里的命令的个数。缺省值是500。

HOST

当前你正连接的数据库服务器主机。这是在每次你与数据库连接时(包括程序启动)设置的，但是可以删除。

IGNOREEOF

如果删除此变量，向一个交互的psql会话发送一个EOF(通常 是CTRL+D)将终止应用。如果设置为一个数字值，那么在应用终止前该数值的EOF字符将被忽略。如果设置了此变量但是没有数值，缺省是10。

LASTOID

最后影响的OID值，即为从一条INSERT或lo_import命令返回的值。此变量只保证在下一条SQL命令的结果显示之前有效。

ON_ERROR_ROLLBACK

当设置为on时，如果一个事务块里的语句产生错误，这个错误将被忽略而事务将继续。当设置为interactive时，这样的错误只是在交互的会话里忽略，而不是在从读取脚本文件的时候。如果未设置或者设置为off，事务块里一个语句生成的错误将会中止整个事务。错误回滚的模式是通过在一个事务块的每个命令前为你隐含地发出一个SAVEPOINT的方式工作的，在命令错误的时候回滚到该保存点。

ON_ERROR_STOP

缺省时，遇到错误后命令处理继续进行。当这个变量设置为on，处理会立即停止。在交互模式下，psql将返回到命令提示符；否则，psql将退出，并返回错误代码3，以示这个情况与致命错误条件的区别，致命错误条件的错误代码为1。不管在哪种情况下，任何当前运行的脚本（顶级脚本，如果有，和任何它调用的其他脚本）都将立即终止。如果顶级命令字符串包含多个SQL命令，处理将在当前命令停止。

PORT

当前你正在连接的数据库服务器的端口。这是在每次你与数据库连接时(包括程序启动)设置的，但是可以取消设置。

PROMPT1

PROMPT2

PROMPT3

这些指明psql显示的提示符如何显示。参阅下面的提示符。

QUIET

设置这个变量为on，等效于命令行选项-q。可能在交互模式下没有什么用。

SINGLELINE

此变量等效于命令行选项-S。

SINGLESTEP

将此变量设置为on等效于命令行选项-s。

USER

当前连接的数据库用户。每次您连接到数据库（包括程序启动）时都会进行设置，但是可以取消设置。

VERBOSITY

可以将此变量设置为default, verbose或terse值，以控制错误报告的详细程度。

SQL代换

psql变量的关键特性是你可以把它们替换成正规的SQL语句，也可以是元命令的参数。另外，psql提供工具确保变量值用作SQL文本并且正确的引用标识符。不用引用替换一个值的语法是在变量名前面加一个冒号(:)。例如：

```
testdb=> \set foo 'my_table'
testdb=> SELECT * FROM :foo;
```

将会查询my_table表。请注意，这可能是不安全的：变量的值是逐字拷贝的，所以它甚至可以包含不对称的引号或反斜杠命令。你必须保证你输入的东西是有意义的。

当一个值被用作SQL文本或标识符时，将其引用是最安全的。要引用一个变量的值作为SQL文本，在单引号中的变量名后面写一个冒号。要引用值作为SQL标识符，在双引号中的变量名后面写一个冒号。这些构造正确的处理引号和嵌入在变量值中的其他特殊字符。上面的例子这样写更安全：

```
testdb=> \set foo 'my_table'
testdb=> SELECT * FROM :"foo";
```

变量替换将不在引用的SQL文本和标识符中执行。因此，一个构造比如':foo'并不从变量的值中产生一个引用的文本（如果它确实发生了，那么它是不安全的，因为它不会正确的处理嵌入在值中的引号。）

使用这个机制的一个例子是拷贝一个文件的内容到一个表字段中。首先加载文件到一个变量，然后替换变量的值为引用的字符串：

```
testdb=> \set content `cat my_file.txt`
testdb=> INSERT INTO my_table VALUES (:'content');
```

(注意，如果my_file.txt包含NUL字节，则仍然无法使用。
psql在变量值中不支持嵌入的NUL字节。)

由于冒号可以合法地出现在SQL命令中，因此除非当前设置了命名变量，否则不会替换明显的插值尝试
(即: name, ': name' 或: "name")。无论如何，您都可以使用反斜杠转义冒号，以防止其被替换。

变量的冒号语法符合诸如ECPG之类的嵌入式查询语言的SQL标准。数组切片和类型转换的冒号语法是Greenplum数据库扩展，有时可能与标准用法冲突。将变量值转义为SQL文字或标识符的冒号引用语法是psql扩展。

提示符

可以根据您的喜好自定义psql提示符。三个变量PROMPT1, PROMPT2和PROMPT3包含字符串和特殊的转义序列，用于描述提示符的外观。PROMPT1是psql请求新命令时显示的普通提示。当在命令输入期间需要更多输入时(例如，因为命令未使用分号终止或未关闭引号)，将显示PROMPT2。当您运行SQL COPY FROM STDIN命令并且您需要在终端上输入行值时，将显示PROMPT3。

所选提示变量的值按字面意义打印，除非遇到百分号(%)。根据下一个字符，某些其他文本将被替换。定义的替换是：

%M

数据库服务器的完整主机名(带有域名)，如果连接是通过UNIX域套接字的，则为[local]；如果UNIX域套接字的默认设置为未编译的默认位置，则为[local]:/dir/name]。

%m

数据库服务器的主机名，在第一个点处被截断，如果连接是通过UNIX域套接字的，则为[local]。

%>

数据库服务器正在侦听的端口号。

%n

数据库会话用户名。(此值的扩展可能在数据库会话期间由于命令SET SESSION AUTHORIZATION的结果而改变。)

%/

当前数据库的名称。

%~

与%/类似，但是如果数据库是您的默认数据库，则输出为~(波浪号)。

%#

如果会话用户是数据库超级用户，则为#，否则为>。(此值的扩展可能在数据库会话期间由于命令SET SESSION AUTHORIZATION的结果而改变。)

%R

在PROMPT1中通常为=，但在单行模式下为^或!如果会话与数据库断开连接（如果\connect失败，则可能发生）。

在PROMPT2中，%R被一个字符替换，该字符取决于psql期望更多输入的原因： - 如果命令尚未终止； *如果有未完成的/*

... */注释； '，如果有一个未完成的带引号的字符串； "，如果有一个未完成的带引号的标识符； \$，如果有一个未完成的带美元引号的字符串； 或者(，如果有一个不匹配的左括号。

在PROMPT3中，%R不产生任何东西。

%x

事务状态：不在事务块中时为空字符串；在事务块中时为*；或!当事务失败时；或?当事务状态不确定时（例如，因为没有连接）。

%digits

指示的八进制代码的字符被替换。

%:name:

psql变量name的值。参阅[高级特性](#)中变量小节获取更多信息。

%`command`

command的输出，类似于通常的反引号替换。

%[... %]

提示可能包含终端控制字符，例如，这些字符会更改提示文本的颜色，背景或样式，或更改终端窗口的标题。为了使行编辑正常工作，必须将这些非打印控制字符用%[和%]括起来以指定为不可见。提示中可能会出现多个这种配对。例如，

```
testdb=> \set PROMPT1 '%[%033[1;33;40m%]%n@%/%R%[%033[0m%]%'#'
```

会在兼容VT100的有颜色的终端上显示黑底黄字（33;40）的粗体（1;）。要将百分号插入提示中，请输入%%。对于提示1和2，默认提示为'%/ %R%# '，对于提示3，默认提示为'>>'。

命令行编辑

psql支持NetBSD libedit库，用于方便的行编辑和检索。当psql退出时，命令历史记录将自动保存，而在psql启动时，将重新加载命令历史记录。尽管补全逻辑不声称是SQL解析器，但也支持制表符补全。制表符补全生成的查询也可能会干扰其他SQL命令，例如SET TRANSACTION ISOLATION LEVEL。如果由于某种原因您不喜欢制表符补全，可以通过将其放在主目录中名为.inputrc的文件中来将其关闭：

```
$if psql
set disable-completion on
$endif
```

环境变量

COLUMNS

如果\pset columns为零，则控制换行格式的宽度和以确定宽度输出是需要分页器还是应在扩展自动模式下切换为垂直格式的宽度。

PAGER

如果查询结果无法显示在屏幕上，则通过此命令通过管道传递查询结果。典型值为more或less。默认值取决于平台。可以通过将**PAGER**设置为空或使用\pset命令中与分页器相关的选项来禁用该分页器的使用。

PGDATABASE

PGHOST

PGPORT

PGUSER

默认连接参数。

PSQL_EDITOR

EDITOR

VISUAL

\e和\ef命令使用的编辑器。按照列出的顺序检查变量；使用设置的第一个。

内置的默认编辑器在Unix系统上为vi，在Windows系统上为notepad.exe。

PSQL_EDITOR_LINENUMBER_ARG

当\e或\ef与行号参数一起使用时，此变量指定用于将起始行号传递给用户的编辑器的命令行参数。对于Emacs或vi等编辑器，这是一个加号。如果选项名称和行号之间需要有空格，请在变量的值中包含尾随空格。例子：

```
PSQL_EDITOR_LINENUMBER_ARG='+'  
PSQL_EDITOR_LINENUMBER_ARG='--line '
```

在Unix系统上，默认值为+（与默认编辑器vi对应，对许多其他常用编辑器很有用）；但Windows系统上没有默认设置。

PSQL_HISTORY

命令历史记录文件的备用位置。进行波浪 (~) 扩展。

PSQLRC

用户的.sqlrc文件的备用位置。进行波浪 (~) 扩展。

SHELL

命令由\!命令执行。

TMPDIR

用于存储临时文件的目录。默认值为`/tmp`。

文件

psqlrc和`~/.psqlrc`

除非传递了`-x`或`-c`选项，否则`psql`在连接数据库之后接受正常的命令之前，尝试从系统范围的启动文件(`psqlrc`)，然后用户的启动文件(`~/.psqlrc`)读取并执行命令。

系统范围的启动文件名为`psqlrc`，可在安装的“系统配置”目录中查找，该目录可以通过运行`pg_config --sysconfdir`来准确获取。默认情况下，相对于包含Greenplum数据库可执行文件的目录，该目录为`../etc/`。可以通过`PGSYSCONFDIR`环境变量显式设置此目录的名称。

用户的个人启动文件名为`.psqlrc`，在用户的主目录中查找。

在缺少这种概念的Windows上，个人启动文件名为`%APPDATA%\postgresql\psqlrc.conf`。可以通过`PSQLRC`环境变量显式设置用户启动文件的位置。

通过在文件名后附加破折号和基础的PostgreSQL主要或次要发行版号(例如`~/.psqlrc-9.4`)，可以使系统范围内的启动文件和用户的个人启动文件都特定于`psql`版本。与非版本特定的文件相比，将优先读取最特定的版本匹配文件。

`.psql_history`

命令行历史存储在`~/.psql_history`文件中，或者在Windows里是`%APPDATA%\postgresql\psql_history`文件。

历史文件的位置可以通过`PSQL_HISTORY`环境变量来明确设置。

注解

`psql`与相同或较旧的主要版本的服务器一起使用时效果最佳。如果服务器的版本比`psql`本身新，则反斜杠命令很可能会失败。但是，`\d`系列的反斜杠命令可以与较旧的服务器版本一起使用，尽管不一定与`psql`本身更新的服务器一起使用。运行SQL命令和显示查询结果的常规功能也应与较新的主要版本的服务器一起使用，但这不能在所有情况下都得到保证。

如果要使用`psql`连接到多个具有不同主要版本的服务器，则建议使用最新版本的`psql`。另外，您可以保留每个主要版本的`psql`副本，并确保使用与各自服务器匹配的版本。但是在实践中，这种额外的复杂性应该没有必要。

Windows用户注解

psql被构建为控制台应用程序。由于Windows控制台窗口使用与系统其余部分不同的编码，因此在psql中使用8位字符时必须格外小心。如果psql检测到有问题的控制台代码页，它将在启动时警告您。要更改控制台代码页，需要做两件事：

通过输入以下内容来设置代码页：

```
cmd.exe /c chcp 1252
```

1252是拉丁字母的字符编码，Microsoft Windows将其用于英语和某些其他西方语言。如果使用的是Cygwin，则可以将此命令放在/etc/profile中。

将控制台字体设置为Lucida Console，因为栅格字体不适用于ANSI代码页。

示例

以交互方式启动psql：

```
psql -p 54321 -U sally mydatabase
```

在psql交互方式下，将命令分布在多行输入上。注意更改提示：

```
testdb=> CREATE TABLE my_table (
testdb(>   first integer not null default 0,
testdb(>   second text)
testdb-> ;
CREATE TABLE
```

查看表定义：

```
testdb=> \d my_table
          Table "my_table"
 Attribute | Type      |      Modifier
-----+-----+-----
 first    | integer   | not null default 0
 second   | text      |
```

通过传入包含SQL命令的文件，以非交互方式运行psql：

```
psql -f /home/gpadmin/test/myscript.sql
```

Greenplum数据库® 6.0文档

[工具指南](#)

[管理工具参考](#)

[客户端工具参考](#)

[客户端工具摘要](#)

clusterdb

createdb

createlang

createuser

dropdb

droplang

dropuser

pg_config

pg_dump

pg_dumpall

pg_restore

psql

reindexdb

vacuumdb

[附加程序](#)

Greenplum database 5管理员
指南

重建数据库中的索引。

概要

```
reindexdb [connection-option ...] [--table | -t table ]
          [--index | -i index ] [dbname]
```

```
reindexdb [connection-option ...] --all | -a
```

```
reindexdb [connection-option ...] --system | -s [dbname]
```

```
reindexdb -? | --help
```

```
reindexdb -v | --version
```

描述

reindexdb是用于在Greenplum数据库中重建索引的工具。

reindexdb是SQL命令REINDEX的包装。通过此工具和通过其他访问服务器的方法为数据库重新索引之间没有明显的区别。

选项

-a | --all

重新索引所有数据库。

[-d] dbname | [--dbname=] dbname

指定要重新建立索引的数据库的名称。如果未指定并且未使用-all，则从环境变量PGDATABASE读取数据库名称。如果未设置，则使用为连接指定的用户名。

-e | --echo

回显reindexdb生成并发送到服务器的命令。

-i index | --index= index

仅重新创建索引。

-q | --quiet

不显示响应。

-s | --system



重新索引系统catalog。

-t *table* | --table= *table*
仅重新索引表。可以通过编写多个-t开关为多个表重新索引。

-V | --version
打印reindexdb版本并退出。

-? | --help
显示有关reindexdb命令行参数的帮助，然后退出。

连接选项

-h *host* | --host= *host*
运行Greenplum master数据库服务器的计算机的主机名。如果未指定，则从环境变量PGHOST读取或默认为localhost。

-p *port* | --port= *port*
Greenplum master数据库服务器正在侦听连接的TCP端口。如果未指定，则从环境变量PGPORT读取或默认为5432。

-U *username* | --username= *username*
要用作连接的数据库角色名称。如果未指定，则从环境变量PGUSER读取或默认为当前系统角色名称。

-w | --no-password
不发出密码提示。如果服务器要求密码验证，而其他方式（例如.pgpass文件）无法使用密码，则连接尝试将失败。此选项在没有用户输入密码的批处理作业和脚本中很有用。

-W | --password
强制输入密码提示。

--maintenance-db= *dbname*
指定要连接以发现应重新索引哪些其他数据库的数据库的名称。如果未指定，将使用postgres数据库，如果不存在，将使用template1。

注解

reindexdb可能需要连接几次master服务器，每次都要求输入密码。在这种情况下，使用~/.pgpass文件很方便。

示例

要重新索引数据库mydb：

```
reindexdb mydb
```

要在名为abcd的数据库中为表foo和索引bar重建索引：

```
reindexdb --table foo --index bar abcd
```

另见

[REINDEX](#)

Greenplum数据库® 6.0文档

□ 工具指南

□ 管理工具参考

□ 客户端工具参考

□ 客户端工具摘要

clusterdb

createdb

createlang

createuser

dropdb

droplang

dropuser

pg_config

pg_dump

pg_dumpall

pg_restore

psql

reindexdb

vacuumdb

□ 附加程序

Greenplum database 5管理员
指南

垃圾收集并分析数据库。

概要

```
vacuumdb [connection-option...] [--full | -f] [--freeze | -F]
[--verbose | -v]
[--analyze | -z] [--analyze-only | -Z] [--table | -t
table [( column [, ...] )]] [dbname]
```

```
vacuumdb [connection-option...] [--all | -a] [--full | -f]
[-F]
```

```
[--verbose | -v] [--analyze | -z]
[--analyze-only | -Z]
```

```
vacuumdb -? | --help
```

```
vacuumdb -V | --version
```

描述

vacuumdb是用于清理Greenplum数据库的工具。 vacuumdb还将生成Greenplum数据库查询优化器使用的内部统计信息。

vacuumdb是SQL命令VACUUM的包装。 通过此工具和通过其他访问服务器的方法来清理数据库之间没有明显的区别。

选项

-a | --all

清理所有数据库。

[-d] dbname | [--dbname=] dbname

要清理的数据库的名称。 如果未指定此选项，并且未使用-a (或--all)，则从环境变量PGDATABASE读取数据库名称。 如果未设置，则使用为连接指定的用户名。

-e | --echo

回显vacuumdb生成并发送到服务器的命令。

-f | --full



选择完全清理，这可以回收更多空间，但是需要更长的时间并排他锁定表。

Warning: 在Greenplum数据库中不建议使用VACUUM FULL。

-F | --freeze

冻结行事务信息。

-q | --quiet

不显示响应。

-t *table* [(*column*)] | --table= *table* [(*column*)]

仅清理或分析该表。只能与--analyze或--analyze-all选项一起指定列名。可以通过写入多个-t开关来清理多个表。如果指定列，则可能必须从shell转义括号。

-v | --verbose

在处理过程中打印详细信息。

-z | --analyze

收集统计信息以供查询优化器使用。

-Z | --analyze-only

仅计算统计信息以供查询优化器使用（不清理）。

-V | --version

打印vacuumdb版本并退出。

-? | --help

显示有关vacuumdb命令行参数的帮助，然后退出。

连接选项

-h *host* | --host= *host*

运行Greenplum master数据库服务器的计算机的主机名。如果未指定，则从环境变量PGHOST读取或默认为localhost。

-p *port* | --port= *port*

Greenplum master数据库服务器正在侦听连接的TCP端口。如果未指定，则从环境变量PGPORT读取或默认为5432。

-U *username* | --username= *username*

要用作连接的数据库角色名称。如果未指定，则从环境变量PGUSER读取或默认为当前系统角色名称。

-w | --no-password

不发出密码提示。如果服务器要求密码验证，而其他方式（例如.pgpass文件）无法使用密码，则连接尝试将失败。此选项在没有用户输入密码的批处理作业和脚本中很有用。

-W | --password

强制输入密码提示。

--maintenance-db= *dbname*

指定要连接以发现应清理哪些其他数据库的数据库名称。如果未指定，将使用postgres数据库，如果不存在，将使用template1。

注解

vacuumdb可能需要连接几次master服务器，每次都要求输入密码。在这种情况下，使用~/.pgpass文件很方便。

示例

清理数据库test：

```
vacuumdb test
```

清理分析数据库bigdb：

```
vacuumdb --analyze bigdb
```

要清理名为mydb的数据库中的单个表foo，并分析表的单个列bar。请注意表名和列名两边的引号，从shell转义括号：

```
vacuumdb --analyze --verbose --table 'foo(bar)' mydb
```

另见

[VACUUM, ANALYZE](#)

Greenplum数据库® 6.0文档

 工具指南 管理工具参考 客户端工具参考 客户端工具摘要 附加程序

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

clusterdb

重新群集之前用CLUSTER群集过的表。

```
clusterdb [connection-option ...] [--verbose | -v] [--table
| -t table] [[--dbname | -d] dbname]
```

```
clusterdb [connection-option ...] [--all | -a] [--verbose |
-v]
```

```
clusterdb -? | --help
```

```
clusterdb -v | --version
```

更多信息请参考[clusterdb](#)。

createdb

创建一个新的数据库。

```
createdb [connection-option ...] [option ...] [dbname
['description']]
```

```
createdb -? | --help
```

```
createdb -v | --version
```

更多信息请参考[createdb](#)。

createlang

为数据库定义新的过程语言。

```
createlang [connection_option ...] [-e] langname [dbname]
```

```
createlang [connection-option ...] -l dbname
```

```
createlang -? | --help  
createlang -v | --version
```

更多信息请参考[createlang](#)。

createuser

创建一个新的数据库角色。

```
createuser [connection-option ...] [role_attribute ...] [-e] role_name  
createuser -? | --help  
createuser -v | --version
```

更多信息请参考[createuser](#)。

dropdb

删除数据库。

```
dropdb [connection-option ...] [-e] [-i] dbname  
dropdb -? | --help  
dropdb -v | --version
```

更多信息请参考[dropdb](#)。

droplang

删除过程语言。

```
droplang [connection-option ...] [-e] langname [[-d] dbname]  
droplang [connection-option ...] [-e] -l dbname  
droplang -? | --help
```

```
droplang -V | --version
```

更多信息请参考[droplang](#)。

dropuser

删除数据库角色。

```
dropuser [connection-option ...] [-e] [-i] role_name  
dropuser -? | --help  
dropuser -V | --version
```

更多信息请参考[dropuser](#)。

pg_config

检索有关Greenplum数据库已安装版本的信息。

```
pg_config [option ...]  
pg_config -? | --help  
pg_config --version
```

更多信息请参考[pg_config](#)。

pg_dump

Extracts a database into a single script file or other archive file.

```
pg_dump [connection-option ...] [dump_option ...] [dbname]  
pg_dump -? | --help  
pg_dump -V | --version
```

更多信息请参考[pg_dump](#)。

pg_dumpall

将Greenplum数据库系统中的所有数据库提取到单个脚本文件或其他归档文件中。

```
pg_dumpall [connection-option ...] [dump_option ...]  
pg_dumpall -? | --help  
pg_dumpall -v | --version
```

更多信息请参考[pg_dumpall](#)。

pg_restore

从pg_dump创建的存档文件中恢复数据库。

```
pg_restore [connection-option ...] [restore_option ...]  
filename  
pg_restore -? | --help  
pg_restore -v | --version
```

更多信息请参考[pg_restore](#)。

psql

Greenplum数据库的交互式命令行接口

```
psql [option ...] [dbname [username]]
```

更多信息请参考[psql](#)。

reindexdb

重建数据库中的索引。

```
reindexdb [connection-option ...] [--table | -t table ]  
          [--index | -i index ] [dbname]  
  
reindexdb [connection-option ...] --all | -a  
  
reindexdb [connection-option ...] --system | -s [dbname]  
  
reindexdb -? | --help  
  
reindexdb -v | --version
```

更多信息请参考[reindexdb](#)。

vacuumdb

垃圾收集并分析数据库。

```
vacuumdb [connection-option...] [--full | -f] [--freeze | -F]  
          [--verbose | -v]  
          [--analyze | -z] [--analyze-only | -Z] [--table | -t  
table [( column [,....] )] ] [dbname]  
  
vacuumdb [connection-option...] [--all | -a] [--full | -f]  
          [-F]  
          [--verbose | -v] [--analyze | -z]  
          [--analyze-only | -Z]  
  
vacuumdb -? | --help  
  
vacuumdb -v | --version
```

更多信息请参考[vacuumdb](#)。

Greenplum数据库® 6.0 文档

□ 参考指南

□ SQL Command
Reference**SQL Syntax Summary**

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

ABORT

放弃当前事务。

ABORT [WORK | TRANSACTION]

See [ABORT](#) for more information.

ALTER AGGREGATE

修改聚集函数的定义。

ALTER AGGREGATE *name* (*type* [, ...]) RENAME TO *new_name*ALTER AGGREGATE *name* (*type* [, ...]) OWNER TO *new_owner*ALTER AGGREGATE *name* (*type* [, ...]) SET SCHEMA *new_schema*See [ALTER AGGREGATE](#) for more information.

ALTER COLLATION

修改字符集的定义。

ALTER COLLATION *name* RENAME TO *new_name*ALTER COLLATION *name* OWNER TO *new_owner*ALTER COLLATION *name* SET SCHEMA *new_schema*See [ALTER COLLATION](#) for more information.

ALTER CONVERSION

更改一个转换的定义。



```
ALTER CONVERSION name RENAME TO newname
```

```
ALTER CONVERSION name OWNER TO newowner
```

```
ALTER CONVERSION name SET SCHEMA new_schema
```

See [ALTER CONVERSION](#) for more information.

ALTER DATABASE

修改数据库的属性

```
ALTER DATABASE name [ WITH CONNECTION LIMIT connlimit ]
```

```
ALTER DATABASE name RENAME TO newname
```

```
ALTER DATABASE name OWNER TO new_owner
```

```
ALTER DATABASE name SET TABLESPACE new_tablespace
```

```
ALTER DATABASE name SET parameter { TO | = } { value | DEFAULT }
```

```
ALTER DATABASE name SET parameter FROM CURRENT
```

```
ALTER DATABASE name RESET parameter
```

```
ALTER DATABASE name RESET ALL
```

See [ALTER DATABASE](#) for more information.

ALTER DEFAULT PRIVILEGES

修改默认的访问权限。

```
ALTER DEFAULT PRIVILEGES
```

```
[ FOR { ROLE | USER } target_role [, ...] ]
```

```
[ IN SCHEMA schema_name [, ...] ]
```

```
abbreviated_grant_or_revoke
```

where *abbreviated_grant_or_revoke* is one of:

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES |  
TRIGGER }
```

```
[, ...] | ALL [ PRIVILEGES ] }
```

```
ON TABLES
```

```
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```
GRANT { { USAGE | SELECT | UPDATE }
```

```
[, ...] | ALL [ PRIVILEGES ] }
```

```

ON SEQUENCES
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
  ON FUNCTIONS
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
  ON TYPES
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

REVOKE [ GRANT OPTION FOR ]
  { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES |
    TRIGGER }
  [, ...] | ALL [ PRIVILEGES ] }
  ON TABLES
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
  { { USAGE | SELECT | UPDATE }
  [, ...] | ALL [ PRIVILEGES ] }
  ON SEQUENCES
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
  { EXECUTE | ALL [ PRIVILEGES ] }
  ON FUNCTIONS
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
  { USAGE | ALL [ PRIVILEGES ] }
  ON TYPES
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ]

```

See [ALTER DEFAULT PRIVILEGES](#) for more information.

ALTER DOMAIN

更改现有域的定义。

```

ALTER DOMAIN name { SET DEFAULT expression | DROP DEFAULT }

ALTER DOMAIN name { SET | DROP } NOT NULL

ALTER DOMAIN name ADD domain_constraint [ NOT VALID ]

ALTER DOMAIN name DROP CONSTRAINT [ IF EXISTS ] constraint_name
[ RESTRICT | CASCADE]

```

```

ALTER DOMAIN name RENAME CONSTRAINT constraint_name TO
new_constraint_name

ALTER DOMAIN name VALIDATE CONSTRAINT constraint_name

ALTER DOMAIN name OWNER TO new_owner

ALTER DOMAIN name RENAME TO new_name

ALTER DOMAIN name SET SCHEMA new_schema

```

See [ALTER DOMAIN](#) for more information.

ALTER EXTENSION

更改在Greenplum数据库中注册的扩展的定义。

```

ALTER EXTENSION name UPDATE [ TO new_version ]
ALTER EXTENSION name SET SCHEMA new_schema
ALTER EXTENSION name ADD member_object
ALTER EXTENSION name DROP member_object

```

where *member_object* is:

```

ACCESS METHOD object_name |
AGGREGATE aggregate_name ( aggregate_signature ) |
CAST (source_type AS target_type) |
COLLATION object_name |
CONVERSION object_name |
DOMAIN object_name |
EVENT TRIGGER object_name |
FOREIGN DATA WRAPPER object_name |
FOREIGN TABLE object_name |
FUNCTION function_name ( [ [ argmode ] [ argname ] argtype [, ...] ] ) |
MATERIALIZED VIEW object_name |
OPERATOR operator_name (left_type, right_type) |
OPERATOR CLASS object_name USING index_method |
OPERATOR FAMILY object_name USING index_method |
[ PROCEDURAL ] LANGUAGE object_name |
SCHEMA object_name |
SEQUENCE object_name |
SERVER object_name |
TABLE object_name |
TEXT SEARCH CONFIGURATION object_name |
TEXT SEARCH DICTIONARY object_name |
TEXT SEARCH PARSER object_name |
TEXT SEARCH TEMPLATE object_name |
TRANSFORM FOR type_name LANGUAGE lang_name |
TYPE object_name |
VIEW object_name

```

and *aggregate_signature* is:

```
* | [ argmode ] [ argname ] argtype [, ... ] |
[ [ argmode ] [ argname ] argtype [, ... ] ]
    ORDER BY [ argmode ] [ argname ] argtype [, ... ]
```

See [ALTER EXTENSION](#) for more information.

ALTER EXTERNAL TABLE

更改一个外部表的定义。

```
ALTER EXTERNAL TABLE name action [, ... ]
```

where *action* is one of:

```
ADD [COLUMN] new_column type
DROP [COLUMN] column [RESTRICT|CASCADE]
ALTER [COLUMN] column TYPE type
OWNER TO new_owner
```

See [ALTER EXTERNAL TABLE](#) for more information.

ALTER FOREIGN DATA WRAPPER

修改一个外部数据包装的定义。

```
ALTER FOREIGN DATA WRAPPER name
[ HANDLER handler_function | NO HANDLER ]
[ VALIDATOR validator_function | NO VALIDATOR ]
[ OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] ) ]
```

```
ALTER FOREIGN DATA WRAPPER name OWNER TO new_owner
ALTER FOREIGN DATA WRAPPER name RENAME TO new_name
```

See [ALTER FOREIGN DATA WRAPPER](#) for more information.

ALTER FOREIGN TABLE

修改外表的定义。

```

ALTER FOREIGN TABLE [ IF EXISTS ] name
    action [, ... ]
ALTER FOREIGN TABLE [ IF EXISTS ] name
    RENAME [ COLUMN ] column_name TO new_column_name
ALTER FOREIGN TABLE [ IF EXISTS ] name
    RENAME TO new_name
ALTER FOREIGN TABLE [ IF EXISTS ] name
    SET SCHEMA new_schema

```

See [ALTER FOREIGN TABLE](#) for more information.

ALTER FUNCTION

修改函数的定义。

```

ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
    action [, ... ] [RESTRICT]

ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
    RENAME TO new_name

ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
    OWNER TO new_owner

ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
    SET SCHEMA new_schema

```

See [ALTER FUNCTION](#) for more information.

ALTER GROUP

更改角色名称或成员关系。

```

ALTER GROUP groupname ADD USER username [, ... ]

ALTER GROUP groupname DROP USER username [, ... ]

ALTER GROUP groupname RENAME TO newname

```

See [ALTER GROUP](#) for more information.

ALTER INDEX

更改一个索引的定义。

```
ALTER INDEX [ IF EXISTS ] name RENAME TO new_name  

ALTER INDEX [ IF EXISTS ] name SET TABLESPACE tablespace_name  

ALTER INDEX [ IF EXISTS ] name SET (storage_parameter = value [, ...])  

ALTER INDEX [ IF EXISTS ] name RESET (storage_parameter [, ...])  

ALTER INDEX ALL IN TABLESPACE name [ OWNED BY role_name [, ...] ]  

SET TABLESPACE new_tablespace [ NOWAIT ]
```

See [ALTER INDEX](#) for more information.

ALTER LANGUAGE

更改过程语言的名称。

```
ALTER LANGUAGE name RENAME TO newname  

ALTER LANGUAGE name OWNER TO new_owner
```

See [ALTER LANGUAGE](#) for more information.

ALTER OPERATOR

更改操作符的定义。

```
ALTER OPERATOR name ( {left_type | NONE} , {right_type | NONE} )  

OWNER TO new_owner  

ALTER OPERATOR name ( {left_type | NONE} , {right_type | NONE} )  

SET SCHEMA new_schema
```

See [ALTER OPERATOR](#) for more information.

ALTER OPERATOR CLASS

更改一个操作符类的定义。

```
ALTER OPERATOR CLASS name USING index_method RENAME TO
```

new_name

```
ALTER OPERATOR CLASS name USING index_method OWNER TO
new_owner
```

```
ALTER OPERATOR CLASS name USING index_method SET SCHEMA
new_schema
```

See [ALTER OPERATOR CLASS](#) for more information.

ALTER OPERATOR FAMILY

更改操作符族的定义。

```
ALTER OPERATOR FAMILY name USING index_method ADD
{ OPERATOR strategy_number operator_name (op_type, op_type) [
FOR SEARCH | FOR ORDER BY sort_family_name ]
| FUNCTION support_number [ (op_type [, op_type ]) ] funcname (
argument_type [, ...] )
} [, ...]
```

```
ALTER OPERATOR FAMILY name USING index_method DROP
{ OPERATOR strategy_number (op_type, op_type)
| FUNCTION support_number [ (op_type [, op_type ]) ]
} [, ...]
```

```
ALTER OPERATOR FAMILY name USING index_method RENAME TO
new_name
```

```
ALTER OPERATOR FAMILY name USING index_method OWNER TO
new_owner
```

```
ALTER OPERATOR FAMILY name USING index_method SET SCHEMA
new_schema
```

See [ALTER OPERATOR FAMILY](#) for more information.

ALTER PROTOCOL

更改一个协议的定义。

```
ALTER PROTOCOL name RENAME TO newname
```

```
ALTER PROTOCOL name OWNER TO newowner
```

See [ALTER PROTOCOL](#) for more information.

ALTER RESOURCE GROUP

更改一个资源组的限制项。

```
ALTER RESOURCE GROUP name SET group_attribute value
```

See [ALTER RESOURCE GROUP](#) for more information.

ALTER RESOURCE QUEUE

更改资源队列的限制。

```
ALTER RESOURCE QUEUE name WITH ( queue_attribute=value [, ...] )
```

See [ALTER RESOURCE QUEUE](#) for more information.

ALTER ROLE

更改一个数据库角色（用户或组）。

```
ALTER ROLE name [ [ WITH ] option [ ... ] ]
```

其中 *option* 可以是：

- SUPERUSER | NOSUPERUSER
- | CREATEDB | NOCREATEDB
- | CREATEROLE | NOCREATEROLE
- | CREATEEXTTABLE | NOCREATEEXTTABLE [(*attribute='value'* [, ...])
where attributes and values are:
type='readable'|'writable'
protocol='gpfdist'|'http'
- | INHERIT | NOINHERIT
- | LOGIN | NOLOGIN
- | REPLICATION | NOREPLICATION
- | CONNECTION LIMIT *connlimit*
- | [ENCRYPTED | UNENCRYPTED] PASSWORD '*password*'
- | VALID UNTIL '*timestamp*'

```
ALTER ROLE name RENAME TO new_name
```

```
ALTER ROLE { name | ALL } [ IN DATABASE database_name ] SET  
configuration_parameter { TO | = } { value | DEFAULT }
```

```
ALTER ROLE { name | ALL } [ IN DATABASE database_name ] SET
```

```
configuration_parameter FROM CURRENT
ALTER ROLE { name | ALL } [ IN DATABASE database_name ] RESET
configuration_parameter
ALTER ROLE { name | ALL } [ IN DATABASE database_name ] RESET ALL
ALTER ROLE name RESOURCE QUEUE {queue_name | NONE}
ALTER ROLE name RESOURCE GROUP {group_name | NONE}
```

See [ALTER ROLE](#) for more information.

ALTER RULE

修改一个规则的定义。

```
ALTER RULE name ON table_name RENAME TO new_name
```

See [ALTER RULE](#) for more information.

ALTER SCHEMA

更改一个模式定义。

```
ALTER SCHEMA name RENAME TO newname
```

```
ALTER SCHEMA name OWNER TO newowner
```

See [ALTER SCHEMA](#) for more information.

ALTER SEQUENCE

更改一个序列发生器的定义。

```
ALTER SEQUENCE [ IF EXISTS ] name [INCREMENT [ BY ] increment]
[MINVALUE minvalue | NO MINVALUE]
[MAXVALUE maxvalue | NO MAXVALUE]
[START [ WITH ] start ]
[RESTART [ [ WITH ] restart] ]
[CACHE cache] [[ NO ] CYCLE]
[OWNED BY {table.column | NONE}]
```

```
ALTER SEQUENCE [ IF EXISTS ] name OWNER TO new_owner
```

```
ALTER SEQUENCE [ IF EXISTS ] name RENAME TO new_name
```

```
ALTER SEQUENCE [ IF EXISTS ] name SET SCHEMA new_schema
```

See [ALTER SEQUENCE](#) for more information.

ALTER SERVER

修改一个外部服务器的定义。

```
ALTER SERVER server_name [ VERSION 'new_version' ]
[ OPTIONS ( [ ADD | SET | DROP ] option['value'] [, ... ] ) ]
```

```
ALTER SERVER server_name OWNER TO new_owner
```

```
ALTER SERVER server_name RENAME TO new_name
```

See [ALTER SERVER](#) for more information.

ALTER TABLE

更改一个表的定义。

```
ALTER TABLE [IF EXISTS] [ONLY] name
action [, ... ]
```

```
ALTER TABLE [IF EXISTS] [ONLY] name
RENAME [COLUMN] column_name TO new_column_name
```

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name
RENAME CONSTRAINT constraint_name TO new_constraint_name
```

```
ALTER TABLE [IF EXISTS] name
RENAME TO new_name
```

```
ALTER TABLE [IF EXISTS] name
SET SCHEMA new_schema
```

```
ALTER TABLE ALL IN TABLESPACE name [ OWNED BY role_name [, ... ] ]
SET TABLESPACE new_tablespace [ NOWAIT ]
```

```
ALTER TABLE [IF EXISTS] [ONLY] name SET
WITH (REORGANIZE=true|false)
| DISTRIBUTED BY ({column_name [opc class]}) [, ... ] )
| DISTRIBUTED RANDOMLY
| DISTRIBUTED REPLICATED
```

```
ALTER TABLE name
[ ALTER PARTITION { partition_name | FOR (RANK(number)) }
```

```
| FOR (value) } partition_action[...]
partition_action
```

其中 *action* 是下列之一：

```
ADD [COLUMN] column_name data_type [ DEFAULT default_expr ]
[column_constraint [ ... ] ]
[ COLLATE collation ]
[ ENCODING (storage_directive [,...]) ]
DROP [COLUMN] [IF EXISTS] column_name [RESTRICT | CASCADE]
ALTER [COLUMN] column_name [ SET DATA ] TYPE type [COLLATE
collation] [USING expression]
ALTER [COLUMN] column_name SET DEFAULT expression
ALTER [COLUMN] column_name DROP DEFAULT
ALTER [COLUMN] column_name { SET | DROP } NOT NULL
ALTER [COLUMN] column_name SET STATISTICS integer
ALTER [COLUMN] column SET ( attribute_option = value [, ... ] )
ALTER [COLUMN] column RESET ( attribute_option [, ... ] )
ADD table_constraint [NOT VALID]
ADD table_constraint_using_index
VALIDATE CONSTRAINT constraint_name
DROP CONSTRAINT [IF EXISTS] constraint_name [RESTRICT | CASCADE]
DISABLE TRIGGER [trigger_name | ALL | USER]
ENABLE TRIGGER [trigger_name | ALL | USER]
CLUSTER ON index_name
SET WITHOUT CLUSTER
SET WITH OIDS
SET WITHOUT OIDS
SET (storage_parameter = value)
RESET (storage_parameter [, ... ])
INHERIT parent_table
NO INHERIT parent_table
OF type_name
NOT OF
OWNER TO new_owner
SET TABLESPACE new_tablespace
```

See [ALTER TABLE](#) for more information.

ALTER TABLESPACE

更改表空间的定义。

```
ALTER TABLESPACE name RENAME TO new_name
```

```
ALTER TABLESPACE name OWNER TO new_owner
```

```
ALTER TABLESPACE name SET ( tablespace_option = value [, ... ] )
```

```
ALTER TABLESPACE name RESET ( tablespace_option [, ... ] )
```

See [ALTER TABLESPACE](#) for more information.

ALTER TEXT SEARCH CONFIGURATION

更改文本搜索配置的定义。

```
ALTER TEXT SEARCH CONFIGURATION name
    ALTER MAPPING FOR token_type [, ...] WITH dictionary_name [, ...]
ALTER TEXT SEARCH CONFIGURATION name
    ALTER MAPPING REPLACE old_dictionary WITH new_dictionary
ALTER TEXT SEARCH CONFIGURATION name
    ALTER MAPPING FOR token_type [, ...] REPLACE old_dictionary WITH
new_dictionary
ALTER TEXT SEARCH CONFIGURATION name
    DROP MAPPING [ IF EXISTS ] FOR token_type [, ...]
ALTER TEXT SEARCH CONFIGURATION name RENAME TO new_name
ALTER TEXT SEARCH CONFIGURATION name OWNER TO new_owner
ALTER TEXT SEARCH CONFIGURATION name SET SCHEMA new_schema
```

See [ALTER TEXT SEARCH CONFIGURATION](#) for more information.

ALTER TEXT SEARCH DICTIONARY

更改文本搜索词典的定义。

```
ALTER TEXT SEARCH DICTIONARY name (
    option [= value] [, ...]
)
ALTER TEXT SEARCH DICTIONARY name RENAME TO new_name
ALTER TEXT SEARCH DICTIONARY name OWNER TO new_owner
ALTER TEXT SEARCH DICTIONARY name SET SCHEMA new_schema
```

See [ALTER TEXT SEARCH DICTIONARY](#) for more information.

ALTER TEXT SEARCH PARSER

更改文本搜索解析器的定义。

```
ALTER TEXT SEARCH PARSE name RENAME TO new_name
```

```
ALTER TEXT SEARCH PARSER name SET SCHEMA new_schema
```

See [ALTER TEXT SEARCH PARSER](#) for more information.

ALTER TEXT SEARCH TEMPLATE

更改文本搜索模板的定义。

```
ALTER TEXT SEARCH TEMPLATE name RENAME TO new_name  
ALTER TEXT SEARCH TEMPLATE name SET SCHEMA new_schema
```

See [ALTER TEXT SEARCH TEMPLATE](#) for more information.

ALTER TYPE

更改一个数据类型的定义。

```
ALTER TYPE name action [, ... ]  
ALTER TYPE name OWNER TO new_owner  
ALTER TYPE name RENAME ATTRIBUTE attribute_name TO  
new_attribute_name [ CASCADE | RESTRICT ]  
ALTER TYPE name RENAME TO new_name  
ALTER TYPE name SET SCHEMA new_schema  
ALTER TYPE name ADD VALUE [ IF NOT EXISTS ] new_enum_value [ {  
BEFORE | AFTER } existing_enum_value ]  
ALTER TYPE name SET DEFAULT ENCODING (storage_directive)
```

其中 *action*是下列选项之一：

```
ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [  
CASCADE | RESTRICT ]  
DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRICT ]  
ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [  
COLLATE collation] [ CASCADE | RESTRICT ]
```

See [ALTER TYPE](#) for more information.

ALTER USER

更改数据库用户（角色）的定义。

```

ALTER USER name RENAME TO newname

ALTER USER name SET config_parameter {TO|=} {value|DEFAULT}

ALTER USER name RESET config_parameter

ALTER USER name RESOURCE QUEUE {queue_name|NONE}

ALTER USER name RESOURCE GROUP {group_name|NONE}

ALTER USER name [ [WITH] option [ ... ] ]

```

See [ALTER USER](#) for more information.

ALTER USER MAPPING

更改外部服务器的用户映射的定义。

```

ALTER USER MAPPING FOR { username | USER | CURRENT_USER | PUBLIC }
    SERVER servername
    OPTIONS ( [ ADD | SET | DROP ] option [ 'value' ] [, ... ] )

```

See [ALTER USER MAPPING](#) for more information.

ALTER VIEW

更改一个视图的定义。

```

ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name SET
    DEFAULT expression

ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name DROP
    DEFAULT

ALTER VIEW [ IF EXISTS ] name OWNER TO new_owner

ALTER VIEW [ IF EXISTS ] name RENAME TO new_name

ALTER VIEW [ IF EXISTS ] name SET SCHEMA new_schema

ALTER VIEW [ IF EXISTS ] name SET (view_option_name [=]
    view_option_value] [, ... ] )

ALTER VIEW [ IF EXISTS ] name RESET (view_option_name [, ... ] )

```

See [ALTER VIEW](#) for more information.

ANALYZE

收集有关一个数据库的统计信息。

```
ANALYZE [VERBOSE] [table [(column [, ...])]]
```

```
ANALYZE [VERBOSE] {root_partition|leaf_partition} [(column [, ...])]
```

```
ANALYZE [VERBOSE] ROOTPARTITION {ALL | root_partition [(column [, ...])]}
```

See [ANALYZE](#) for more information.

BEGIN

启动事务块。

```
BEGIN [WORK | TRANSACTION] [transaction_mode]
```

See [BEGIN](#) for more information.

CHECKPOINT

强制执行事务日志检查点。

```
CHECKPOINT
```

See [CHECKPOINT](#) for more information.

CLOSE

关闭游标。

```
CLOSE cursor_name
```

See [CLOSE](#) for more information.

CLUSTER

根据索引对磁盘上的堆存储表进行物理重新排序。在Greenplum数据库中不建议执行此操作。

```
CLUSTER indexname ON tablename
CLUSTER [VERBOSE] tablename
CLUSTER [VERBOSE]
```

See [CLUSTER](#) for more information.

COMMENT

定义或更改对象的注释。

```
COMMENT ON
{ TABLE object_name |
COLUMN relation_name.column_name |
AGGREGATE agg_name (agg_type [, ...]) |
CAST (source_type AS target_type) |
COLLATION object_name
CONSTRAINT constraint_name ON table_name |
CONVERSION object_name |
DATABASE object_name |
DOMAIN object_name |
EXTENSION object_name |
FOREIGN DATA WRAPPER object_name |
FOREIGN TABLE object_name |
FUNCTION func_name ([[argmode] [argname] argtype [, ...]]) |
INDEX object_name |
LARGE OBJECT large_object_oid |
OPERATOR operator_name (left_type, right_type) |
OPERATOR CLASS object_name USING index_method |
[PROCEDURAL] LANGUAGE object_name |
RESOURCE GROUP object_name |
RESOURCE QUEUE object_name |
ROLE object_name |
RULE rule_name ON table_name |
SCHEMA object_name |
SEQUENCE object_name |
SERVER object_name |
TABLESPACE object_name |
TRIGGER trigger_name ON table_name |
TYPE object_name |
VIEW object_name }
IS 'text'
```

See [COMMENT](#) for more information.

COMMIT

提交当前事务。

```
COMMIT [WORK | TRANSACTION]
```

See [COMMIT](#) for more information.

COPY

在文件和表之间复制数据。

```
COPY table_name [(column_name [, ...])]  
FROM {'filename' | PROGRAM 'command' | STDIN}  
[ [ WITH ] ( option [, ...] ) ]  
[ ON SEGMENT ]  
  
COPY { table_name [(column_name [, ...])] | (query) }  
TO {'filename' | PROGRAM 'command' | STDOUT}  
[ [ WITH ] ( option [, ...] ) ]  
[ ON SEGMENT ]
```

See [COPY](#) for more information.

CREATE AGGREGATE

定义一个新的聚集函数

```
CREATE AGGREGATE name ( [ argmode ] [ ] arg_data_type [ , ... ] )(  
    SFUNC = sfunc,  
    STYPE = state_data_type  
    [ , SSPACE = state_data_size ]  
    [ , FINALFUNC = ffunc ]  
    [ , FINALFUNC_EXTRA ]  
    [ , COMBINEFUNC = combinefunc ]  
    [ , SERIALFUNC = serialfunc ]  
    [ , DESERIALFUNC = deserialfunc ]  
    [ , INITCOND = initial_condition ]  
    [ , MSFUNC = msfunc ]  
    [ , MINVFUNC = minvfunc ]  
    [ , MSTYPE = mstate_data_type ]
```

```
[ , MSSPACE = mstate_data_size ]
[ , MFINALFUNC = mffunc ]
[ , MFINALFUNC_EXTRA ]
[ , MINITCOND = minitial_condition ]
[ , SORTOP = sort_operator ]
)

CREATE AGGREGATE name ( [ [ argmode ] [ argname ] arg_data_type [ , ... ]
] ]
    ORDER BY [ argmode ] [ argname ] arg_data_type [ , ... ] ) (
    SFUNC = sfunc,
    STYPE = state_data_type
    [ , SSPACE = state_data_size ]
    [ , FINALFUNC = ffunc ]
    [ , FINALFUNC_EXTRA ]
    [ , COMBINEFUNC = combinefunc ]
    [ , SERIALFUNC = serialfunc ]
    [ , DESERIALFUNC = deserialfunc ]
    [ , INITCOND = initial_condition ]
    [ , HYPOTHETICAL ]
)
)
```

or the old syntax

```
CREATE AGGREGATE name (
    BASETYPE = base_type,
    SFUNC = sfunc,
    STYPE = state_data_type
    [ , SSPACE = state_data_size ]
    [ , FINALFUNC = ffunc ]
    [ , FINALFUNC_EXTRA ]
    [ , COMBINEFUNC = combinefunc ]
    [ , SERIALFUNC = serialfunc ]
    [ , DESERIALFUNC = deserialfunc ]
    [ , INITCOND = initial_condition ]
    [ , MSFUNC = msfunc ]
    [ , MINVFUNC = minvfunc ]
    [ , MSTYPE = mstate_data_type ]
    [ , MSSPACE = mstate_data_size ]
    [ , MFINALFUNC = mffunc ]
    [ , MFINALFUNC_EXTRA ]
    [ , MINITCOND = minitial_condition ]
    [ , SORTOP = sort_operator ]
)
)
```

See [CREATE AGGREGATE](#) for more information.

CREATE CAST

定义一种新的造型。

```
CREATE CAST (sourcetype AS targettype)
    WITH FUNCTION funcname (argtype [ , ... ])
```

[AS ASSIGNMENT | AS IMPLICIT]

```
CREATE CAST (sourcetype AS targettype)
WITHOUT FUNCTION
[AS ASSIGNMENT | AS IMPLICIT]
```

```
CREATE CAST (sourcetype AS targettype)
WITH INOUT
[AS ASSIGNMENT | AS IMPLICIT]
```

See [CREATE CAST](#) for more information.

CREATE COLLATION

使用指定的操作系统语言环境或通过复制现有的排序规则来定义新的排序规则。

```
CREATE COLLATION name (
[ LOCALE = locale, ]
[ LC_COLLATE = lc_collate, ]
[ LC_CTYPE = lc_ctype ])
```

```
CREATE COLLATION name FROM existing_collation
```

See [CREATE COLLATION](#) for more information.

CREATE CONVERSION

定义新的编码转换。

```
CREATE [DEFAULT] CONVERSION name FOR source_encoding TO
dest_encoding FROM funcname
```

See [CREATE CONVERSION](#) for more information.

CREATE DATABASE

创建一个新的数据库。

```
CREATE DATABASE name [ [WITH] [OWNER [=] user_name]
[TEMPLATE [=] template]
[ENCODING [=] encoding]
```

```
[LC_COLLATE [=] lc_collate]
[LC_CTYPE [=] lc_ctype]
[TABLESPACE [=] tablespace]
[CONNECTION LIMIT [=] connlimit ] ]
```

See [CREATE DATABASE](#) for more information.

CREATE DOMAIN

定义一个新域。

```
CREATE DOMAIN name [AS] data_type [DEFAULT expression]
[ COLLATE collation ]
[ CONSTRAINT constraint_name
| NOT NULL | NULL
| CHECK (expression) [...] ]
```

See [CREATE DOMAIN](#) for more information.

CREATE EXTENSION

Registers an extension in a Greenplum database.

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
[ WITH ] [ SCHEMA schema_name ]
[ VERSION version ]
[ FROM old_version ]
[ CASCADE ]
```

See [CREATE EXTENSION](#) for more information.

CREATE EXTERNAL TABLE

定义一个新的外部表。

```
CREATE [READABLE] EXTERNAL [TEMPORARY | TEMP] TABLE

(column_name data_type [, ...] | LIKE other_table )
LOCATION ('file://segghost[:port]/path/file' [, ...])
| ('gpfdist://filehost[:port]/file_pattern[#transform=trans_name]'
[, ...]
| ('gpfdists://filehost[:port]/file_pattern[#transform=trans_name]'
```

```

        [, ...])
        | ('pxf://path-to-data?
PROFILE=profile_name[&SERVER=server_name][&custom-
option=value[...]]')
        | ('s3://S3_endpoint[:port]/bucket_name/[S3_prefix] [region=S3-
region] [config=config_file]')
        [ON MASTER]
FORMAT 'TEXT'
[([ [HEADER]
    [DELIMITER [AS] 'delimiter' | 'OFF']
    [NULL [AS] 'null string']
    [ESCAPE [AS] 'escape' | 'OFF']
    [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
    [FILL MISSING FIELDS] )]

| 'CSV'
[([ [HEADER]
    [QUOTE [AS] 'quote']
    [DELIMITER [AS] 'delimiter']
    [NULL [AS] 'null string']
    [FORCE NOT NULL column [, ...]]
    [ESCAPE [AS] 'escape']
    [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
    [FILL MISSING FIELDS] )

| 'CUSTOM' (Formatter=<formatter_specifications>
[ ENCODING 'encoding' ]
[ [LOG ERRORS] SEGMENT REJECT LIMIT count
[ROWS | PERCENT] ]


CREATE [READABLE] EXTERNAL WEB [TEMPORARY | TEMP] TABLE

( column_name data_type [, ...] | LIKE other_table )
LOCATION ('http://webhost[:port]/path/file' [, ...])
| EXECUTE 'command' [ON ALL
    | MASTER
    | number_of_segments
    | HOST ['segment_hostname']
    | SEGMENT segment_id ]

FORMAT 'TEXT'
[([ [HEADER]
    [DELIMITER [AS] 'delimiter' | 'OFF']
    [NULL [AS] 'null string']
    [ESCAPE [AS] 'escape' | 'OFF']
    [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
    [FILL MISSING FIELDS] )

| 'CSV'
[([ [HEADER]
    [QUOTE [AS] 'quote']
    [DELIMITER [AS] 'delimiter']
    [NULL [AS] 'null string']
    [FORCE NOT NULL column [, ...]]
    [ESCAPE [AS] 'escape']
    [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
    [FILL MISSING FIELDS] )

| 'CUSTOM' (Formatter=<formatter specifications>
[ ENCODING 'encoding' ]
[ [LOG ERRORS] SEGMENT REJECT LIMIT count
[ROWS | PERCENT] ]

```

```

CREATE WRITABLE EXTERNAL [TEMPORARY | TEMP] TABLE table_name
( column_name data_type [, ...] | LIKE other_table )
LOCATION('gpfdist://outputhost[:port]/filename[#transform=trans_name]'
[, ...])
| ('gpfdists://outputhost[:port]/file_pattern[#transform=trans_name]'
[, ...])
FORMAT 'TEXT'
[ ( [DELIMITER [AS] 'delimiter']
[NULL [AS] 'null string']
[ESCAPE [AS] 'escape' | 'OFF' ] ) ]
| 'CSV'
[ ([QUOTE [AS] 'quote']
[DELIMITER [AS] 'delimiter']
[NULL [AS] 'null string']
[FORCE QUOTE column [, ...]] | * ]
[ESCAPE [AS] 'escape' ] )]

| 'CUSTOM' (Formatter=<formatter specifications>)
[ ENCODING 'write_encoding' ]
[ DISTRIBUTED BY ({column opclass}, [ ... ]) | DISTRIBUTED
RANDOMLY ]

```



```

CREATE WRITABLE EXTERNAL [TEMPORARY | TEMP] TABLE table_name
( column_name data_type [, ...] | LIKE other_table )
LOCATION('s3://S3_endpoint[:port]/bucket_name/[S3_prefix]
[region=S3-region] [config=config_file])
[ON MASTER]
FORMAT 'TEXT'
[ ( [DELIMITER [AS] 'delimiter']
[NULL [AS] 'null string']
[ESCAPE [AS] 'escape' | 'OFF' ] ) ]
| 'CSV'
[ ([QUOTE [AS] 'quote']
[DELIMITER [AS] 'delimiter']
[NULL [AS] 'null string']
[FORCE QUOTE column [, ...]] | * ]
[ESCAPE [AS] 'escape' ] )]

```



```

CREATE WRITABLE EXTERNAL WEB [TEMPORARY | TEMP] TABLE
table_name
( column_name data_type [, ...] | LIKE other_table )
EXECUTE 'command' [ON ALL]
FORMAT 'TEXT'
[ ( [DELIMITER [AS] 'delimiter']
[NULL [AS] 'null string']
[ESCAPE [AS] 'escape' | 'OFF' ] ) ]
| 'CSV'
[ ([QUOTE [AS] 'quote']
[DELIMITER [AS] 'delimiter']
[NULL [AS] 'null string']
[FORCE QUOTE column [, ...]] | * ]
[ESCAPE [AS] 'escape' ] )]
| 'CUSTOM' (Formatter=<formatter specifications>)
[ ENCODING 'write_encoding' ]
[ DISTRIBUTED BY ({column opclass}, [ ... ]) | DISTRIBUTED
RANDOMLY ]

```

See [CREATE EXTERNAL TABLE](#) for more information.

CREATE FOREIGN DATA WRAPPER

定义一个新的外部数据包装器。

```
CREATE FOREIGN DATA WRAPPER name
  [ HANDLER handler_function | NO HANDLER ]
  [ VALIDATOR validator_function | NO VALIDATOR ]
  [ OPTIONS ( [ mpp_execute { 'master' | 'any' | 'all segments' } [,] ] option 'value' [,]
    ... ) ]
```

See [CREATE FOREIGN DATA WRAPPER](#) for more information.

CREATE FOREIGN TABLE

定义一个新的外部表。

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name ( [
  column_name data_type [ OPTIONS ( option 'value' [, ...] ) ] [ COLLATE
collation ] [ column_constraint [ ... ] ]
  [, ...]
])
  SERVER server_name
  [ OPTIONS ( [ mpp_execute { 'master' | 'any' | 'all segments' } [,] ] option 'value' [,]
    ... ) ]
```

See [CREATE FOREIGN TABLE](#) for more information.

CREATE FUNCTION

定义一个新函数。

```
CREATE [OR REPLACE] FUNCTION name
( [ [argmode] [argname] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
  [ RETURNS rettype
  | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE langname
  | WINDOW
  | IMMUTABLE | STABLE | VOLATILE | [NOT] LEAKPROOF
```

```

| CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
| [EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER
| EXECUTE ON { ANY | MASTER | ALL SEGMENTS }
| COST execution_cost
| SET configuration_parameter { TO value | = value | FROM CURRENT
}
| AS 'definition'
| AS 'obj_file', 'link_symbol' } ...
[ WITH ({ DESCRIBE = describe_function
} [, ...]) ]

```

See [CREATE FUNCTION](#) for more information.

CREATE GROUP

定义一个新的数据库角色。

```
CREATE GROUP name [[WITH] option [ ... ]]
```

See [CREATE GROUP](#) for more information.

CREATE INDEX

定义一个新索引。

```

CREATE [UNIQUE] INDEX [name] ON table_name [USING method]
( {column_name | (expression)} [COLLATE parameter] [opclass] [
ASC | DESC ] [NULLS { FIRST | LAST } ] [, ...] )
[ WITH ( storage_parameter = value [, ...] ) ]
[ TABLESPACE tablespace ]
[ WHERE predicate ]

```

See [CREATE INDEX](#) for more information.

CREATE LANGUAGE

定义一种新的过程语言。

```
CREATE [ OR REPLACE ] [ PROCEDURAL ] LANGUAGE name
```

```

CREATE [ OR REPLACE ] [ TRUSTED ] [ PROCEDURAL ] LANGUAGE name
HANDLER call_handler [ INLINE inline_handler ]

```

```
[ VALIDATOR valfunction ]
```

See [CREATE LANGUAGE](#) for more information.

CREATE OPERATOR

定义一个新的运算符。

```
CREATE OPERATOR name (
    PROCEDURE = funcname
    [, LEFTARG = lefttype] [, RIGHTARG = righttype]
    [, COMMUTATOR = com_op] [, NEGATOR = neg_op]
    [, RESTRICT = res_proc] [, JOIN = join_proc]
    [, HASHES] [, MERGES] )
```

See [CREATE OPERATOR](#) for more information.

CREATE OPERATOR CLASS

定义一个新的运算符类。

```
CREATE OPERATOR CLASS name [DEFAULT] FOR TYPE data_type
    USING index_method [ FAMILY family_name ] AS
        { OPERATOR strategy_number operator_name [ ( op_type, op_type ) ] [
            FOR SEARCH | FOR ORDER BY sort_family_name ]
            | FUNCTION support_number funcname (argument_type [, ...])
            | STORAGE storage_type
        } [, ...]
```

See [CREATE OPERATOR CLASS](#) for more information.

CREATE OPERATOR FAMILY

定义一个新的运算符族。

```
CREATE OPERATOR FAMILY name USING index_method
```

See [CREATE OPERATOR FAMILY](#) for more information.

CREATE PROTOCOL

注册自定义数据访问协议，该协议可以在定义Greenplum数据库外部表时指定。

```
CREATE [TRUSTED] PROTOCOL name (  
    [readfunc='read_call_handler'] [, writefunc='write_call_handler']  
    [, validatorfunc='validate_handler'])
```

See [CREATE PROTOCOL](#) for more information.

CREATE RESOURCE GROUP

定义一个新的资源组。

```
CREATE RESOURCE GROUP name WITH (group_attribute=value [, ...])
```

See [CREATE RESOURCE GROUP](#) for more information.

CREATE RESOURCE QUEUE

定义一个新的资源队列。

```
CREATE RESOURCE QUEUE name WITH (queue_attribute=value [, ...])
```

See [CREATE RESOURCE QUEUE](#) for more information.

CREATE ROLE

定义一个新的数据库角色（用户或组）。

```
CREATE ROLE name [[WITH] option [ ... ]]
```

See [CREATE ROLE](#) for more information.

CREATE RULE

定义新的重写规则。

```
CREATE [OR REPLACE] RULE name AS ON event
    TO table_name [WHERE condition]
    DO [ALSO | INSTEAD] { NOTHING | command | (command; command
        ...) }
```

See [CREATE RULE](#) for more information.

CREATE SCHEMA

定义一个新的模式。

```
CREATE SCHEMA schema_name [AUTHORIZATION username]
    [schema_element [ ... ]]

CREATE SCHEMA AUTHORIZATION rolename [schema_element [ ... ]]

CREATE SCHEMA IF NOT EXISTS schema_name [ AUTHORIZATION
    user_name ]

CREATE SCHEMA IF NOT EXISTS AUTHORIZATION user_name
```

See [CREATE SCHEMA](#) for more information.

CREATE SEQUENCE

定义一个新的序列生成器。

```
CREATE [TEMPORARY | TEMP] SEQUENCE name
    [INCREMENT [BY] value]
    [MINVALUE minvalue | NO MINVALUE]
    [MAXVALUE maxvalue | NO MAXVALUE]
    [START [ WITH ] start]
    [CACHE cache]
    [[NO] CYCLE]
    [OWNED BY { table.column | NONE }]
```

See [CREATE SEQUENCE](#) for more information.

CREATE SERVER

定义一个新的外部服务器。

```
CREATE SERVER server_name [ TYPE 'server_type' ] [ VERSION 'server_version' ]
    FOREIGN DATA WRAPPER fdw_name
    [ OPTIONS ( [ mpp_execute { 'master' | 'any' | 'all segments' } [,] ] option_value [,]
    ... ) ]
```

See [CREATE SERVER](#) for more information.

CREATE TABLE

定义一个新表。

```
CREATE [ [GLOBAL | LOCAL] {TEMPORARY | TEMP} | UNLOGGED] TABLE [IF NOT EXISTS]
    table_name (
        [ { column_name data_type [ COLLATE collation ]
        [column_constraint ...] }
        [ ENCODING ( storage_directive [, ...] ) ]
        | table_constraint
        | LIKE source_table [ like_option ... ] }
        || column_reference_storage_directive [, ...]
        [, ...]
    ])
    [ INHERITS ( parent_table [, ...] ) ]
    [ WITH ( storage_parameter [=value] [, ...] )
        | WITH OIDS | WITHOUT OIDS ]
    [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
    [ TABLESPACE tablespace_name ]
    [ DISTRIBUTED BY (column [opclass], [, ...] )
        | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
    [ PARTITION BY partition_type (column)
        [ SUBPARTITION BY partition_type (column) ]
        [ SUBPARTITION TEMPLATE ( template_spec ) ]
        [...]
        (partition_spec)
        || SUBPARTITION BY partition_type (column) ]
        [...]
        (partition_spec
        [ ( subpartition_spec
            [...] )
        ]
        )
    ]
)

CREATE [ [GLOBAL | LOCAL] {TEMPORARY | TEMP} | UNLOGGED ] TABLE [IF NOT EXISTS]
    table_name
    OF type_name [ (
```

```
CREATE [ [GLOBAL | LOCAL] {TEMPORARY | TEMP} | UNLOGGED ] TABLE [IF NOT EXISTS]
    table_name
    OF type_name [ (
```

```

{ column_name WITH OPTIONS [ column_constraint [ ... ] ]
| table_constraint }
[, ...]
)
[ WITH ( storage_parameter [=value] [, ... ] ) | WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace_name ]

```

See [CREATE TABLE](#) for more information.

CREATE TABLE AS

根据查询结果定义一个新表。

```

CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE
table_name
[ (column_name [, ...]) ]
[ WITH ( storage_parameter [=value] [, ... ] ) | WITH OIDS | WITHOUT
OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace_name ]
AS query
[ WITH [ NO ] DATA ]
[ DISTRIBUTED BY (column [, ... ]) | DISTRIBUTED RANDOMLY |
DISTRIBUTED REPLICATED ]

```

See [CREATE TABLE AS](#) for more information.

CREATE TABLESPACE

定义一个新的表空间。

```

CREATE TABLESPACE tablespace_name [OWNER username] LOCATION
'/path/to/dir'
[WITH (contentID_1='/path/to/dir1'[, contentID_2='/path/to/dir2' ... ])]

```

See [CREATE TABLESPACE](#) for more information.

CREATE TEXT SEARCH CONFIGURATION

定义新的文本搜索配置。

```
CREATE TEXT SEARCH CONFIGURATION name (
    PARSER = parser_name |
    COPY = source_config
)
```

See [CREATE TEXT SEARCH CONFIGURATION](#) for more information.

CREATE TEXT SEARCH DICTIONARY

定义一个新的文本搜索字典。

```
CREATE TEXT SEARCH DICTIONARY name (
    TEMPLATE = template
    [, option=value [, ... ]]
)
```

See [CREATE TEXT SEARCH DICTIONARY](#) for more information.

CREATE TEXT SEARCH PARSER

定义一个新的文本搜索解析器。

```
CREATE TEXT SEARCH PARSER name (
    START = start_function ,
    GETTOKEN = gettken_function ,
    END = end_function ,
    LEXTYPES = lextypes_function
    [, HEADLINE = headline_function ]
)
```

See [CREATE TEXT SEARCH PARSER](#) for more information.

CREATE TEXT SEARCH TEMPLATE

定义一个新的文本搜索模板。

```
CREATE TEXT SEARCH TEMPLATE name (
    [ INIT = init_function , ]
    LEXIZE = lexize_function
)
```

See [CREATE TEXT SEARCH TEMPLATE](#) for more information.

CREATE TYPE

定义新的数据类型。

```
CREATE TYPE name AS
(attribute_name data_type [ COLLATE collation ] [, ... ]))

CREATE TYPE name AS ENUM
(['label' [, ... ]])

CREATE TYPE name AS RANGE (
    SUBTYPE = subtype
    [, SUBTYPE_OPCLASS = subtype_operator_class ]
    [, COLLATION = collation ]
    [, CANONICAL = canonical_function ]
    [, SUBTYPE_DIFF = subtype_diff_function ]
)

CREATE TYPE name (
    INPUT = input_function,
    OUTPUT = output_function
    [, RECEIVE = receive_function]
    [, SEND = send_function]
    [, TYPMOD_IN = type_modifier_input_function ]
    [, TYPMOD_OUT = type_modifier_output_function ]
    [, INTERNALLENGTH = {internallength | VARIABLE}]
    [, PASSEDBYVALUE]
    [, ALIGNMENT = alignment]
    [, STORAGE = storage]
    [, LIKE = like_type]
    [, CATEGORY = category]
    [, PREFERRED = preferred]
    [, DEFAULT = default]
    [, ELEMENT = element]
    [, DELIMITER = delimiter]
    [, COLLATABLE = collatable]
    [, COMPRESSTYPE = compression_type]
    [, COMPRESSLEVEL = compression_level]
    [, BLOCKSIZE = blocksize] )
```

CREATE TYPE *name*

See [CREATE TYPE](#) for more information.

CREATE USER

默认情况下，使用LOGIN特权定义一个新的数据库角色。

```
CREATE USER name [[WITH] option [ ... ]]
```

See [CREATE USER](#) for more information.

CREATE USER MAPPING

定义用户到外部服务器的新映射。

```
CREATE USER MAPPING FOR { username | USER | CURRENT_USER | PUBLIC }
    SERVER servername
    [ OPTIONS ( option'value'[, ...] ) ]
```

See [CREATE USER MAPPING](#) for more information.

CREATE VIEW

定义一个新的视图。

```
CREATE [OR REPLACE] [TEMP | TEMPORARY] [RECURSIVE] VIEW name
    [ ( column_name [, ...] ) ]
    [ WITH ( view_option_name [= view_option_value] [, ...] ) ]
    AS query
    [ WITH [ CASCDED | LOCAL ] CHECK OPTION ]
```

See [CREATE VIEW](#) for more information.

DEALLOCATE

取消分配预编译语句。

```
DEALLOCATE [PREPARE] name
```

See [DEALLOCATE](#) for more information.

DECLARE

定义一个游标。

```
DECLARE name [BINARY] [INSENSITIVE] [NO SCROLL] CURSOR  
[ {WITH | WITHOUT} HOLD ]  
FOR query [FOR READ ONLY]
```

See [DECLARE](#) for more information.

DELETE

从表中删除行。

```
[ WITH [ RECURSIVE ] with_query [, ...] ]  
DELETE FROM [ONLY] table [[AS] alias]  
[USING usinglist]  
[WHERE condition | WHERE CURRENT OF cursor_name]  
[RETURNING * | output_expression [[AS] output_name] [, ...]]
```

See [DELETE](#) for more information.

DISCARD

放弃会话状态。

```
DISCARD { ALL | PLANS | TEMPORARY | TEMP }
```

See [DISCARD](#) for more information.

DROP AGGREGATE

删除聚合函数。

```
DROP AGGREGATE [IF EXISTS] name ( type [, ...] ) [CASCADE | RESTRICT]
```

See [DROP AGGREGATE](#) for more information.

DO

执行匿名代码块作为临时匿名函数。

```
DO [ LANGUAGE lang_name ] code
```

See [DO](#) for more information.

DROP CAST

删除一个造型。

```
DROP CAST [IF EXISTS] (sourcetype AS targettype) [CASCADE | RESTRICT]
```

See [DROP CAST](#) for more information.

DROP COLLATION

删除以前定义的排序规则。

```
DROP COLLATION [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

See [DROP COLLATION](#) for more information.

DROP CONVERSION

删除转换。

```
DROP CONVERSION [IF EXISTS] name [CASCADE | RESTRICT]
```

See [DROP CONVERSION](#) for more information.

DROP DATABASE

删除数据库。

```
DROP DATABASE [IF EXISTS] name
```

See [DROP DATABASE](#) for more information.

DROP DOMAIN

删除域。

```
DROP DOMAIN [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See [DROP DOMAIN](#) for more information.

DROP EXTENSION

从Greenplum数据库中删除扩展。

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

See [DROP EXTENSION](#) for more information.

DROP EXTERNAL TABLE

删除外部表定义。

```
DROP EXTERNAL [WEB] TABLE [IF EXISTS] name [CASCADE | RESTRICT]
```

See [DROP EXTERNAL TABLE](#) for more information.

DROP FOREIGN DATA WRAPPER

删除外部数据包装器。

```
DROP FOREIGN DATA WRAPPER [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

See [DROP FOREIGN DATA WRAPPER](#) for more information.

DROP FOREIGN TABLE

删除外部表。

```
DROP FOREIGN TABLE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

See [DROP FOREIGN TABLE](#) for more information.

DROP FUNCTION

删除函数。

```
DROP FUNCTION [IF EXISTS] name ( [ [argmode] [argname] argtype  
[, ...] ] ) [CASCADE | RESTRICT]
```

See [DROP FUNCTION](#) for more information.

DROP GROUP

删除数据库角色。

```
DROP GROUP [IF EXISTS] name [, ...]
```

See [DROP GROUP](#) for more information.

DROP INDEX

删除索引。

```
DROP INDEX [ CONCURRENTLY ] [ IF EXISTS ] name [, ...] [ CASCADE |  
RESTRICT ]
```

See [DROP INDEX](#) for more information.

DROP LANGUAGE

删除过程语言。

```
DROP [PROCEDURAL] LANGUAGE [IF EXISTS] name [CASCADE | RESTRICT]
```

See [DROP LANGUAGE](#) for more information.

DROP OPERATOR

删除运算符。

```
DROP OPERATOR [IF EXISTS] name ( {lefttype | NONE} ,  
{righttype | NONE} ) [CASCADE | RESTRICT]
```

See [DROP OPERATOR](#) for more information.

DROP OPERATOR CLASS

删除运算符类。

```
DROP OPERATOR CLASS [IF EXISTS] name USING index_method [CASCADE |  
RESTRICT]
```

See [DROP OPERATOR CLASS](#) for more information.

DROP OPERATOR FAMILY

删除一个运算符族

```
DROP OPERATOR FAMILY [IF EXISTS] name USING index_method  
[CASCADE | RESTRICT]
```

See [DROP OPERATOR FAMILY](#) for more information.

DROP OWNED

删除数据库角色拥有的数据库对象。

```
DROP OWNED BY name [, ...] [CASCADE | RESTRICT]
```

See [DROP OWNED](#) for more information.

DROP PROTOCOL

从数据库中删除外部表数据访问协议。

```
DROP PROTOCOL [IF EXISTS] name
```

See [DROP PROTOCOL](#) for more information.

DROP RESOURCE GROUP

删除资源组。

```
DROP RESOURCE GROUP group_name
```

See [DROP RESOURCE GROUP](#) for more information.

DROP RESOURCE QUEUE

删除资源队列。

```
DROP RESOURCE QUEUE queue_name
```

See [DROP RESOURCE QUEUE](#) for more information.

DROP ROLE

删除数据库角色。

```
DROP ROLE [IF EXISTS] name [, ...]
```

See [DROP ROLE](#) for more information.

DROP RULE

删除重写规则。

```
DROP RULE [IF EXISTS] name ON table_name [CASCADE | RESTRICT]
```

See [DROP RULE](#) for more information.

DROP SCHEMA

删除模式。

```
DROP SCHEMA [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See [DROP SCHEMA](#) for more information.

DROP SEQUENCE

删除序列。

```
DROP SEQUENCE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See [DROP SEQUENCE](#) for more information.

DROP SERVER

删除外部服务器描述符。

```
DROP SERVER [ IF EXISTS ] servername [ CASCADE | RESTRICT ]
```

See [DROP SERVER](#) for more information.

DROP TABLE

删除表。

```
DROP TABLE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See [DROP TABLE](#) for more information.

DROP TABLESPACE

删除表空间。

```
DROP TABLESPACE [IF EXISTS] tablespacename
```

See [DROP TABLESPACE](#) for more information.

DROP TEXT SEARCH CONFIGURATION

删除文本搜索配置。

```
DROP TEXT SEARCH CONFIGURATION [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

See [DROP TEXT SEARCH CONFIGURATION](#) for more information.

DROP TEXT SEARCH DICTIONARY

删除文本搜索字典。

```
DROP TEXT SEARCH DICTIONARY [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

See [DROP TEXT SEARCH DICTIONARY](#) for more information.

DROP TEXT SEARCH PARSER

删除文本搜索解析器。

```
DROP TEXT SEARCH PARSER [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

See [DROP TEXT SEARCH PARSER](#) for more information.

DROP TEXT SEARCH TEMPLATE

删除文本搜索模板。

```
DROP TEXT SEARCH TEMPLATE [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

See [DROP TEXT SEARCH TEMPLATE](#) for more information.

DROP TYPE

删除数据类型。

```
DROP TYPE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See [DROP TYPE](#) for more information.

DROP USER

删除数据库角色。

```
DROP USER [IF EXISTS] name [, ...]
```

See [DROP USER](#) for more information.

DROP USER MAPPING

删除外部服务器的用户映射。

```
DROP USER MAPPING [ IF EXISTS ] { username | USER | CURRENT_USER |  
PUBLIC }  
    SERVER servername
```

See [DROP USER MAPPING](#) for more information.

DROP VIEW

删除视图。

```
DROP VIEW [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See [DROP VIEW](#) for more information.

END

提交当前事务。

```
END [WORK | TRANSACTION]
```

See [END](#) for more information.

EXECUTE

执行准备好的SQL语句。

```
EXECUTE name [ (parameter [, ...]) ]
```

See [EXECUTE](#) for more information.

EXPLAIN

显示语句的查询计划。

```
EXPLAIN [ ( option [, ...] ) ] statement
EXPLAIN [ANALYZE] [VERBOSE] statement
```

See [EXPLAIN](#) for more information.

FETCH

使用游标从查询中检索行。

```
FETCH [ forward_direction { FROM | IN } ]
      cursor_name
```

See [FETCH](#) for more information.

GRANT

定义访问权限。

```
GRANT { {SELECT | INSERT | UPDATE | DELETE | REFERENCES |
TRIGGER | TRUNCATE} [, ...] | ALL [PRIVILEGES] }
      ON { [TABLE] table_name [, ...]
           | ALL TABLES IN SCHEMA schema_name [, ...] }
      TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name [, ...] )
      [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
      ON [ TABLE ] table_name [, ...]
      TO { role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { {USAGE | SELECT | UPDATE} [, ...] | ALL [PRIVILEGES] }
      ON { SEQUENCE sequence_name [, ...]
           | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
      TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { {CREATE | CONNECT | TEMPORARY | TEMP} [, ...] | ALL
[PRIVILEGES] }
      ON DATABASE database_name [, ...]
      TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
      ON DOMAIN domain_name [, ...]
      TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN DATA WRAPPER fdw_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN SERVER server_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { EXECUTE | ALL [PRIVILEGES] }
    ON { FUNCTION function_name ( [ [ argmode ] [ argname ] argtype [, ...]
        ] ) [, ...]
        | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { USAGE | ALL [PRIVILEGES] }
    ON LANGUAGE lang_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { { CREATE | USAGE } [, ...] | ALL [PRIVILEGES] }
    ON SCHEMA schema_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC} [, ...] [ WITH GRANT OPTION ]

GRANT { CREATE | ALL [PRIVILEGES] }
    ON TABLESPACE tablespace_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON TYPE type_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT parent_role [, ...]
    TO member_role [, ...] [WITH ADMIN OPTION]

GRANT { SELECT | INSERT | ALL [PRIVILEGES] }
    ON PROTOCOL protocolname
    TO username

```

See [GRANT](#) for more information.

INSERT

在表中创建新行。

```

[ WITH [ RECURSIVE ] with_query [, ...] ]
INSERT INTO table [(column [, ... ])]
    {DEFAULT VALUES | VALUES ( {expression|DEFAULT} [, ...] ) [, ...] |
query}
    [RETURNING *|output_expression [[AS] output_name] [, ...]]

```

See [INSERT](#) for more information.

LOAD

加载或重新加载共享库文件。

```
LOAD 'filename'
```

See [LOAD](#) for more information.

LOCK

锁表。

```
LOCK [TABLE] [ONLY] name [ * ] [, ...] [IN lockmode MODE] [NOWAIT]
```

See [LOCK](#) for more information.

MOVE

定位游标。

```
MOVE [ forward_direction [ FROM | IN ] ]  
      cursor_name
```

See [MOVE](#) for more information.

PREPARE

准备要执行的语句。

```
PREPARE name [ (datatype [, ...] ) ] AS statement
```

See [PREPARE](#) for more information.

REASSIGN OWNED

更改数据库角色拥有的数据库对象的所有权。

```
REASSIGN OWNED BY old_role [, ...] TO new_role
```

See [REASSIGN OWNED](#) for more information.

REINDEX

重建索引。

```
REINDEX {INDEX | TABLE | DATABASE | SYSTEM} name
```

See [REINDEX](#) for more information.

RELEASE SAVEPOINT

销毁先前定义的保存点。

```
RELEASE [SAVEPOINT] savepoint_name
```

See [RELEASE SAVEPOINT](#) for more information.

RESET

将系统配置参数的值恢复为默认值。

```
RESET configuration_parameter
```

```
RESET ALL
```

See [RESET](#) for more information.

REVOKE

删除访问权限。

```
REVOKE [GRANT OPTION FOR] { {SELECT | INSERT | UPDATE | DELETE
| REFERENCES | TRIGGER | TRUNCATE} [, ...] | ALL [PRIVILEGES] }
```

```
ON { [TABLE] table_name [, ...]
| ALL TABLES IN SCHEMA schema_name [, ...] }
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT]
```

```
REVOKE [ GRANT OPTION FOR ] { { SELECT | INSERT | UPDATE
| REFERENCES } ( column_name [, ...] )
[, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
ON [ TABLE ] table_name [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]
```

```
REVOKE [GRANT OPTION FOR] { {USAGE | SELECT | UPDATE} [, ...]
| ALL [PRIVILEGES] }
ON { SEQUENCE sequence_name [, ...]
| ALL SEQUENCES IN SCHEMA schema_name [, ...] }
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT]
```

```
REVOKE [GRANT OPTION FOR] { {CREATE | CONNECT
| TEMPORARY | TEMP} [, ...] | ALL [PRIVILEGES] }
ON DATABASE database_name [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT]
```

```
REVOKE [ GRANT OPTION FOR ]
{ USAGE | ALL [ PRIVILEGES ] }
ON DOMAIN domain_name [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]
```

```
REVOKE [ GRANT OPTION FOR ]
{ USAGE | ALL [ PRIVILEGES ] }
ON FOREIGN DATA WRAPPER fdw_name [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]
```

```
REVOKE [ GRANT OPTION FOR ]
{ USAGE | ALL [ PRIVILEGES ] }
ON FOREIGN SERVER server_name [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]
```

```
REVOKE [GRANT OPTION FOR] {EXECUTE | ALL [PRIVILEGES]}
ON { FUNCTION funcname ( [[argmode] [argname] argtype
[, ...]] ) [, ...]
| ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT]
```

```
REVOKE [GRANT OPTION FOR] {USAGE | ALL [PRIVILEGES]}
ON LANGUAGE langname [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]
```

```
REVOKE [GRANT OPTION FOR] { {CREATE | USAGE} [, ...]
| ALL [PRIVILEGES] }
ON SCHEMA schema_name [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { CREATE | ALL [PRIVILEGES] }
ON TABLESPACE tablespacename [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT]

REVOKE [ GRANT OPTION FOR ]
{ USAGE | ALL [ PRIVILEGES ] }
ON TYPE type_name [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]

REVOKE [ADMIN OPTION FOR] parent_role [, ...]
FROM [ GROUP ] member_role [, ...]
[CASCADE | RESTRICT]
```

See [REVOKE](#) for more information.

ROLLBACK

中止当前事务。

```
ROLLBACK [WORK | TRANSACTION]
```

See [ROLLBACK](#) for more information.

ROLLBACK TO SAVEPOINT

将当前事务回滚到保存点。

```
ROLLBACK [WORK | TRANSACTION] TO [SAVEPOINT]
savepoint_name
```

See [ROLLBACK TO SAVEPOINT](#) for more information.

SAVEPOINT

在当前事务中定义一个新的保存点。

```
SAVEPOINT savepoint_name
```

See [SAVEPOINT](#) for more information.

SELECT

从表或视图中检索行。

```
[ WITH [ RECURSIVE1] with_query [, ...] ]
SELECT [ALL | DISTINCT [ON (expression [, ...])]]
      * | expression[[AS] output_name] [, ...]
      [FROM from_item [, ...]]
      [WHERE condition]
      [GROUP BY grouping_element [, ...]]
      [HAVING condition [, ...]]
      [WINDOW window_name AS (window_definition) [, ...]]
      [{UNION | INTERSECT | EXCEPT} [ALL | DISTINCT] select]
      [ORDER BY expression [ASC | DESC | USING operator] [NULLS {FIRST | LAST}] [, ...]]
      [LIMIT {count | ALL}]
      [OFFSET start [ ROW | ROWS ]]
      [FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY]
      [FOR {UPDATE | NO KEY UPDATE | SHARE | KEY SHARE} [OF table_name [, ...]] [NOWAIT] [...]]
TABLE { [ ONLY ] table_name [*] | with_query_name }
```

See [SELECT](#) for more information.

SELECT INTO

根据查询结果定义一个新表。

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [ALL | DISTINCT [ON ( expression [, ...] )]]
      * | expression [AS output_name] [, ...]
      INTO [TEMPORARY | TEMP | UNLOGGED ] [TABLE] new_table
      [FROM from_item [, ...]]
      [WHERE condition]
      [GROUP BY expression [, ...]]
      [HAVING condition [, ...]]
      [{UNION | INTERSECT | EXCEPT} [ALL | DISTINCT] select]
      [ORDER BY expression [ASC | DESC | USING operator] [NULLS {FIRST |
```

```

LAST} [, ...]
[LIMIT {count | ALL}]
[OFFSET start [ ROW | ROWS ] ]
[FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
[FOR {UPDATE | SHARE} [OF table_name [, ...]] [NOWAIT]
[...]]

```

See [SELECT INTO](#) for more information.

SET

更改Greenplum数据库配置参数的值。

```

SET [SESSION | LOCAL] configuration_parameter {TO | =} value | 'value'
| DEFAULT}
SET [SESSION | LOCAL] TIME ZONE {timezone | LOCAL | DEFAULT}

```

See [SET](#) for more information.

SET CONSTRAINTS

设置当前事务的约束检查时间。

```
SET CONSTRAINTS { ALL | name [, ...] } { DEFERRED | IMMEDIATE }
```

See [SET CONSTRAINTS](#) for more information.

SET ROLE

设置当前会话的当前角色标识符。

```

SET [SESSION | LOCAL] ROLE rolename
SET [SESSION | LOCAL] ROLE NONE
RESET ROLE

```

See [SET ROLE](#) for more information.

SET SESSION AUTHORIZATION

设置会话角色标识符和当前会话的当前角色标识符。

```
SET [SESSION | LOCAL] SESSION AUTHORIZATION rolename
```

```
SET [SESSION | LOCAL] SESSION AUTHORIZATION DEFAULT
```

```
RESET SESSION AUTHORIZATION
```

See [SET SESSION AUTHORIZATION](#) for more information.

SET TRANSACTION

设置当前事务的特性。

```
SET TRANSACTION [transaction_mode] [READ ONLY | READ WRITE]
```

```
SET TRANSACTION SNAPSHOT snapshot_id
```

```
SET SESSION CHARACTERISTICS AS TRANSACTION transaction_mode
[READ ONLY | READ WRITE]
[NOT] DEFERRABLE
```

See [SET TRANSACTION](#) for more information.

SHOW

显示系统配置参数的值。

```
SHOW configuration_parameter
```

```
SHOW ALL
```

See [SHOW](#) for more information.

START TRANSACTION

启动事务块。

```
START TRANSACTION [transaction_mode] [READ WRITE | READ ONLY]
```

See [START TRANSACTION](#) for more information.

TRUNCATE

清空表的所有行。

```
TRUNCATE [TABLE] [ONLY] name [ * ] [, ...]  
[ RESTART IDENTITY | CONTINUE IDENTITY ] [CASCADE | RESTRICT]
```

See [TRUNCATE](#) for more information.

UPDATE

更新表的行。

```
[ WITH [ RECURSIVE ] with_query [, ...] ]  
UPDATE [ONLY] table [[AS] alias]  
SET {column=expression|DEFAULT} |  
(column [, ...])=({expression|DEFAULT} [, ...])} [, ...]  
[FROM fromlist]  
[WHERE condition| WHERE CURRENT OF cursor_name ]
```

See [UPDATE](#) for more information.

VACUUM

垃圾收集并可选地分析数据库。

```
VACUUM [({ FULL | FREEZE | VERBOSE | ANALYZE } [, ...])] [table [(column [, ...])]]  
VACUUM [FULL] [FREEZE] [VERBOSE] [table]  
VACUUM [FULL] [FREEZE] [VERBOSE] ANALYZE  
[table [(column [, ...])]]
```

See [VACUUM](#) for more information.

VALUES

计算一组行。

```
VALUES ( expression [, ...] ) [, ...]
    [ORDER BY sort_expression [ ASC | DESC | USING operator ] [, ...]
    ]
    [LIMIT { count | ALL } ]
    [OFFSET start [ ROW | ROWS ] ]
    [FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
```

See [VALUES](#) for more information.

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

 参考指南 SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

放弃当前事务.

概要

ABORT [WORK | TRANSACTION]

Description

ABORT 回滚当前事务并使这个事务触发的所有更新操作都被丢弃掉。这个命令与SQL标准命令ROLLBACK表现相同，[ROLLBACK](#)它是由于历史原因而出现的。

Parameters

WORK

TRANSACTION

可选关键词，对Abort操作没有影响

注解

使用 COMMIT 来成功终止一个事务。

在一个事务之外发送 ABORT 命令会没有效果，但是会产生一个警告消息。

兼容性

由于历史原因，这是Greenplum数据库的扩展命令，与标准SQL命令 ROLLBACK等效。



另见

[BEGIN, COMMIT, ROLLBACK](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

修改聚集函数的定义。

概要

```
ALTER AGGREGATE name ( type [ , ... ] ) RENAME TO new_name
```

```
ALTER AGGREGATE name ( type [ , ... ] ) OWNER TO new_owner
```

```
ALTER AGGREGATE name ( type [ , ... ] ) SET SCHEMA  
new_schema
```

Description

ALTER AGGREGATE 用来修改一个聚集函数的定义

你必须是该聚集函数的所有者才能使用ALTER AGGREGATE。要更改聚集函数的模式, 你还必须拥有新模式的 CREATE 权限。要更改聚集函数的所有者, 你还必须是新角色的直接或者成员, 而且新角色必须在聚集函数的模式上面拥有 CREATE 权限。 (这些限制强制要求拥有者不能通过丢弃并重建该聚集函数来做任何不能做的事情。然而, 超级用户可以改变任何聚合函数的所有权。)

参数

name

现有聚集函数的名称 (可以是限定模式)

type

聚集函数能运行的输入数据类型。要引用零参数聚合函数, 写入*代替输入数据类型的列表。

new_name

聚集函数的新名称。

new_owner

聚集函数的新所有者。

new_schema

聚集函数的新模式名



示例

将 integer 类型的聚集函数 myavg 重命名为 my_average:

```
ALTER AGGREGATE myavg(integer) RENAME TO my_average;
```

将 integer 类型的聚集函数 myavg 的所有者改为 joe:

```
ALTER AGGREGATE myavg(integer) OWNER TO joe;
```

将 integer 类型的聚集函数 myavg 模式修改为 myschema:

```
ALTER AGGREGATE myavg(integer) SET SCHEMA myschema;
```

兼容性

SQL 标准中没有 ALTER AGGREGATE 语句。

另见

[CREATE AGGREGATE](#), [DROP AGGREGATE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

修改字符集的定义。

概要

```
ALTER COLLATION name RENAME TO new_name
```

```
ALTER COLLATION name OWNER TO new_owner
```

```
ALTER COLLATION name SET SCHEMA new_schema
```

参数

name

现有字符集名称。 (可以是限定模式)

new_name

新的字符集名称

new_owner

新的字符集所有者。

new_schema

新的字符集模式。

描述

你必须是字符集的所有者才能使用 `ALTER COLLATION`。要更改字符集的所有者，你还必须是新角色的直接或者成员，而且新角色必须在字符集的模式上面拥有 `CREATE` 的权限。 (这些限制强制要求拥有者不能通过丢弃并重建该字符集来做任何不能做的事情。然而，超级用户可以改变任何字符集的所有权。)

示例

将字符集名从 `de_DE` 修改为 `german`:

```
ALTER COLLATION "de_DE" RENAME TO german;
```



将字符集en_US的所有者修改为 joe:

```
ALTER COLLATION "en_US" OWNER TO joe;
```

兼容性

SQL标准中没有 ALTER COLLATION 语法。

另见

[CREATE COLLATION](#), [DROP COLLATION](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改一个转换的定义。

概要

```
ALTER CONVERSION name RENAME TO newname
```

```
ALTER CONVERSION name OWNER TO newowner
```

```
ALTER CONVERSION name SET SCHEMA new_schema
```

描述

ALTER CONVERSION 修改一个转换的定义

你必须是转换的所有者才能使用ALTER CONVERSION。要更改转换的所有者，你还必须是新角色的直接或者成员，而且新角色必须在转换的模式上面拥有CREATE 的权限。（这些限制强制要求拥有者不能通过丢弃并重建该转换来做任何不能做的事情。然而，超级用户可以改变任何转换的所有权。）

参数

name

现有转换的名称。（可以是限定模式）

newname

新的转换名称。

newowner

转换的新所有者。

new_schema

转换的新模式

示例

将转换iso_8859_1_to_utf8重命名为latin1_to_unicode:



```
ALTER CONVERSION iso_8859_1_to_utf8 RENAME TO  
latin1_to_unicode;
```

将转换iso_8859_1_to_utf8的所有者修改为 joe:

```
ALTER CONVERSION iso_8859_1_to_utf8 OWNER TO joe;
```

兼容性

SQL标准中没有 ALTER CONVERSION 语句。

另见

[CREATE CONVERSION](#), [DROP CONVERSION](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

修改数据库的属性

概要

```
ALTER DATABASE name [ WITH CONNECTION LIMIT connlimit ]
ALTER DATABASE name RENAME TO newname
ALTER DATABASE name OWNER TO new_owner
ALTER DATABASE name SET TABLESPACE new_tablespace
ALTER DATABASE name SET parameter { TO | = } { value | DEFAULT }
ALTER DATABASE name SET parameter FROM CURRENT
ALTER DATABASE name RESET parameter
ALTER DATABASE name RESET ALL
```

描述

ALTER DATABASE修改一个数据库的属性。

第一个语句修改数据库的连接数。只有数据的所有者或者超级用户可以更改此设置。

第二个语句修改数据库的名字。只有数据库的所有者或者超级用户可以重命名数据库;非超级用户必须拥有 CREATEDB 权限。不能修改当前数据库的名字, 连接到其他的数据库先。

第三个聚集修改数据库的所有者。要修改所有者, 你必须是新角色的直接或者间接的成员, 而且必须有 CREATEDB 的权限。 (超级用户可以自动拥有所有的权限。)

第四个语句修改数据库的默认表空间。只有数据库的所有者或者超级用户可以修改默认表空, 你还需要在新的表空间上有创建的权限。这个命令会将这个数据库所有在默认表空间的表和索引都移到新的表空间。注意, 不在默认表空间的表和索引不受影响。

剩下的语句是用来修改Greenplum数据库会话级别的参数默认值。当一个会话在数据库开启后, 这个会话提供的参数值将成为默认参数值。



数据库在配置文件（`postgresql.conf`）配置的默认值将被覆盖。只有数据库所有者或者超级用户才能修改会话默认值。某些参数不能用这种方式设置，或者只能被超级用户设置。

Parameters

name

将要被修改属性的数据库名称。

connlimit

最大并发连接数。缺省值-1表示没有限制

parameter value

将指定配置参数的数据库的会话默认值设置为给定值。如果这个值是 `DEFAULT`，或者是等效的 `RESET`，数据库的特定设置会被删除，因此系统范围的默认设置将在新会话中继承。用 `RESET ALL` 来清除所有的数据库设置，参见 [服务器配置参数](#) 来获取更多关于用户级别的参数设置。

newname

新的数据库名。

new_owner

新的数据库所有者。

new_tablespace

新的数据库默认表空间。

注意

还可以为特定角色（用户）而不是数据库设置配置参数会话默认值。如果存在冲突，角色特定的设置将覆盖数据库特定的设置。参见 `ALTER ROLE`.

示例

为数据库`mydatabase` 设置默认的模式搜索路径：

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

兼容性

ALTER DATABASE 语句是Greenplum的扩展语句。

另见

[CREATE DATABASE](#), [DROP DATABASE](#), [SET](#), [CREATE TABLESPACE](#)

Parent topic: [SQL Command Reference](#)

PRIVILEGES

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

**ALTER DEFAULT
PRIVILEGES**

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLE

ALTER FOREIGN
DATA WRAPPER

ALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

修改默认的访问权限。

概要

```

ALTER DEFAULT PRIVILEGES
    [ FOR { ROLE | USER } target_role [, ...] ]
    [ IN SCHEMA schema_name [, ...] ]
    abbreviated_grant_or_revoke

where abbreviated_grant_or_revoke is one of:

GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE |
  REFERENCES | TRIGGER }
  [, ...] | ALL [ PRIVILEGES ] }
  ON TABLES
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH
  GRANT OPTION ]

GRANT { { USAGE | SELECT | UPDATE }
  [, ...] | ALL [ PRIVILEGES ] }
  ON SEQUENCES
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH
  GRANT OPTION ]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
  ON FUNCTIONS
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH
  GRANT OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
  ON TYPES
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH
  GRANT OPTION ]

REVOKE [ GRANT OPTION FOR ]
  { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE |
  REFERENCES | TRIGGER }
  [, ...] | ALL [ PRIVILEGES ] }
  ON TABLES
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
  { { USAGE | SELECT | UPDATE }
  [, ...] | ALL [ PRIVILEGES ] }
  ON SEQUENCES
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]

```



```
[ CASCADE | RESTRICT ]  
  
REVOKE [ GRANT OPTION FOR ]  
  { EXECUTE | ALL [ PRIVILEGES ] }  
  ON FUNCTIONS  
  FROM { [ GROUP ] role_name | PUBLIC } [ , ... ]  
  [ CASCADE | RESTRICT ]  
  
REVOKE [ GRANT OPTION FOR ]  
  { USAGE | ALL [ PRIVILEGES ] }  
  ON TYPES  
  FROM { [ GROUP ] role_name | PUBLIC } [ , ... ]  
  [ CASCADE | RESTRICT ]
```

描述

`ALTER DEFAULT PRIVILEGES`允许你为将来会创建的对象赋予权限。（不会影响那些已经赋予存在对象的权限。）目前只有表（包括视图和外键表）、序列、函数、类型的权限可以被修改。

你只可以修改那些你自己或者你所在角色会创建的对象的权限。这些权限既可以在全局范围内设置（例如：在当前数据库创建的所有对象。），也可以为指定模式下的对象设置。每个模式指定的默认权限将添加到特定对象类型的全局默认权限。

如GRANT中介绍的，任何对象的默认权限通常都会被授予所有可授予的权限，也可能被授予一些PUBLIC权限。不管怎样，这些都可以通过`ALTER DEFAULT PRIVILEGES`来修改全局默认权限来设置。

参数

target_role

当前的角色是一个成员时，那么*target_role*为要修改的角色名，如果FOR ROLE被省略，那么会将当前的角色当为目标角色。

schema_name

模式名，如果指定了模式名，那么之后在这个模式下面创建的所有对象默认的权限都会被修改。如果 IN SCHEMA被省略，那么全局权限会被修改。

role_name

授予或撤消权限的现有角色的名称。这个参数和所有在abbreviated_grant_or_revoke中的参数，按照GRANT或REVOKE描述中进行操作，除非这个是为整个对象类而不是特

定的命名对象设置权限。

注意

用 [psql](#) 的 `\ddp` 命令来获取有关默认权限的现有分配信息，权限值的含义与在 [GRANT](#) 中 `\dp` 命令的解释相同。

如果你想删除一个被修改了默认权限的角色，必须撤消其默认权限的更改，或使用 `DROP OWNED BY` 删除该角色的默认权限条目。

示例

为随后在 `myschema` 中创建的所有表（和视图）的每个人授予 `SELECT` 权限，并允许角色 `webuser` 也插入到它们中：

```
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema GRANT SELECT ON
TABLES TO PUBLIC;
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema GRANT INSERT ON
TABLES TO webuser;
```

撤销上面的操作，那样随后创建的表不会拥有比正常权限更多的权限：

```
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema REVOKE SELECT
ON TABLES FROM PUBLIC;
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema REVOKE INSERT
ON TABLES FROM webuser;
```

将所有在 `admin` 角色上随后创建的函数上删除正常在函数上赋予的所有的执行权限：

```
ALTER DEFAULT PRIVILEGES FOR ROLE admin REVOKE EXECUTE ON
FUNCTIONS FROM PUBLIC;
```

兼容性

在标准SQL语句中没有 `ALTER DEFAULT PRIVILEGES` 语句。

另见

[GRANT, REVOKE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改现有域的定义。

概要

```
ALTER DOMAIN name { SET DEFAULT expression | DROP DEFAULT }
```

```
ALTER DOMAIN name { SET | DROP } NOT NULL
```

```
ALTER DOMAIN name ADD domain_constraint [ NOT VALID ]
```

```
ALTER DOMAIN name DROP CONSTRAINT [ IF EXISTS ]  
constraint_name [ RESTRICT | CASCADE ]
```

```
ALTER DOMAIN name RENAME CONSTRAINT constraint_name TO  
new_constraint_name
```

```
ALTER DOMAIN name VALIDATE CONSTRAINT constraint_name
```

```
ALTER DOMAIN name OWNER TO new_owner
```

```
ALTER DOMAIN name RENAME TO new_name
```

```
ALTER DOMAIN name SET SCHEMA new_schema
```

描述

ALTER DOMAIN更改一个现有域的定义。有几种形式：

- **SET/DROP DEFAULT** — 这些形式设置或删除域的默认值。默认值仅适用于后续的INSERT命令。它们不影响使用域的表中已经存在的行。
- **SET/DROP NOT NULL** — 这些形式会改变域是否被标记为允许NULL值或者拒绝NULL值。用户只能SET NOT NULL当使用域的列不包含空值时。
- **ADD *domain_constraint* [NOT VALID]** — 这种形式使用和CREATE DOMAIN相同的语法为域增加一个新的约束。如果一个新的约束被添加到域中，所有在这个域中的列都会根据新增加的约束重新检查，这些检查可以用NOT VALID选项来限制； 约束可以被随后的ALTER DOMAIN ... VALIDATE CONSTRAINT来使之可用，新插入或这更新的行会永远根据所有的约束最检查，即使那

些被标记为NOT VALID的约束，NOT VALID只能定义在CHECK的约束上。

- **DROP CONSTRAINT [IF EXISTS]**—此形式删除域的约束。如果提供了IF EXISTS选项而要删除的约束不存在，语句只会抛出一个提醒而不会抛出错误。
- **RENAME CONSTRAINT**—此形式改稿一个现有域的约束的名字
- **VALIDATE CONSTRAINT**—此形式会验证之前被标记为NOT VALID的约束，这样会验证在这个约束中的列的所有数据
- **OWNER**—This form changes the owner of the domain to the specified user.
- **RENAME**—此形式将域的所有者更改为指定的用户。
- **SET SCHEMA**—此形式更改域的模式。与域相关联的任何约束也被移动到新的模式中。

用户必须拥有域才能ALTER DOMAIN。要更改域的模式，用户还必须对新模式具有CREATE特权。要更改所有者，用户还必须是新拥有角色的直接或间接成员，并且该角色必须对该域的模式具有CREATE特权。这些限制强制修改拥有者不能做一些通过删除和重建域做不到的事情。不过，一个超级用户怎么都能更改任何域的所有权。）。

参数

name

要更改的现有域的名称（可选方案限定。

domain_constraint

域的新域约束。

constraint_name

要修改护着删除的约束名

NOT VALID

不检查在约束中定义的已存在的数据。

CASCADE

自动删除依赖于此约束的所有对象。

RESTRICT

如果有任何依赖对象，拒绝删除约束。这是默认行为。

new_name

新的域名。

new_constraint_name

新的约束名。

new_owner

域的新所有者的用户名。

new_schema

域的新模式。

示例

添加NOT NULL 约束到一个域：

```
ALTER DOMAIN zipcode SET NOT NULL;
```

从一个域中移除NOT NULL约束：

```
ALTER DOMAIN zipcode DROP NOT NULL;
```

向域添加检查约束：

```
ALTER DOMAIN zipcode ADD CONSTRAINT zipchk CHECK  
(char_length(VALUE) = 5);
```

从域中删除检查约束：

```
ALTER DOMAIN zipcode DROP CONSTRAINT zipchk;
```

在一个域中从命名一个约束

```
ALTER DOMAIN zipcode RENAME CONSTRAINT zipchk TO zip_check;
```

将域移动到不同的模式：

```
ALTER DOMAIN zipcode SET SCHEMA customers;
```

兼容性

ALTER DOMAIN 符合SQL标准，除了 OWNER, RENAME, SET SCHEMA, and VALIDATE CONSTRAINT 变形，这些是Greenplum Database的扩展功能。 ADD CONSTRAINT中的NOT VALID变形也是Greenplum Database的扩展功能。

另见

[CREATE DOMAIN, DROP DOMAIN](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改在Greenplum数据库中注册的扩展的定义。

概要

```
ALTER EXTENSION name UPDATE [ TO new_version ]
ALTER EXTENSION name SET SCHEMA new_schema
ALTER EXTENSION name ADD member_object
ALTER EXTENSION name DROP member_object
```

where *member_object* is:

```
ACCESS METHOD object_name |
AGGREGATE aggregate_name ( aggregate_signature ) |
CAST (source_type AS target_type) |
COLLATION object_name |
CONVERSION object_name |
DOMAIN object_name |
EVENT TRIGGER object_name |
FOREIGN DATA WRAPPER object_name |
FOREIGN TABLE object_name |
FUNCTION function_name ( [ [ argmode ] [ argname ] argtype [, ...] ] ) |
MATERIALIZED VIEW object_name |
OPERATOR operator_name (left_type, right_type) |
OPERATOR CLASS object_name USING index_method |
OPERATOR FAMILY object_name USING index_method |
[ PROCEDURAL ] LANGUAGE object_name |
SCHEMA object_name |
SEQUENCE object_name |
SERVER object_name |
TABLE object_name |
TEXT SEARCH CONFIGURATION object_name |
TEXT SEARCH DICTIONARY object_name |
TEXT SEARCH PARSER object_name |
TEXT SEARCH TEMPLATE object_name |
TRANSFORM FOR type_name LANGUAGE lang_name |
TYPE object_name |
VIEW object_name
```

and *aggregate_signature* is:

```
* | [ argmode ] [ argname ] argtype [ , ... ] |
[ [ argmode ] [ argname ] argtype [ , ... ] ]
ORDER BY [ argmode ] [ argname ] argtype [ , ... ]
```



描述

`ALTER EXTENSION` 更改已安装扩展的定义。有几种子形式：

UPDATE

此形式将扩展更新到一个较新版本。该扩展必须提供一个合适的更新脚本（或一系列脚本），可以将当前安装的版本修改为所要求的版本。

SET SCHEMA

这种形式将扩展对象移动到另一个模式中。扩展名必须是可重定位。

ADD *member_object*

这种形式将一个现有对象添加到该扩展中。这在扩展更新脚本中很有用。该对象后续将被当作该扩展的一个成员。尤其是该对象只有通过删除扩展才能删除。

DROP *member_object*

这种形式从扩展中删除一个成员对象。这主要对扩展更新脚本有用。只有撤销该对象与其扩展之间的关联后才能删除该对象。

参阅 [将相关对象打包到扩展中](#) 获取更多关于这些操作的信息。

用户必须对扩展有所有权才可以使用`ALTER EXTENSION`。`ADD` 和 `DROP` 形式也要求对需要增加或者删除的对象有所有权。

参数

name

需要安装的扩展名

new_version

新版本的扩展。*new_version* 以是标识符或字符串文字。如果未指定，该命令将尝试更新到扩展控制文件中的默认版本。

new_schema

扩展的新模式。

object_name
aggregate_name
function_name
operator_name

要从该扩展增加或者移除的对象的名称。表、聚集、域、外部表、函数、操作符、操作符类、操作符族、序列、文本搜索对象、类型和视图的名称可以被方案限定。

source_type

需要转换的源数据类型的名称。

target_type

需要转换的目标数据类型的名称。

argmode

函数或聚集参数的模型：IN, OUT, INOUT, 或者 VARIADIC。默认值是IN。

这个命令会忽略掉OUT参数。只需要输入参数才决定标识一个函数。这足够列出IN, INOUT和VARIADIC参数。

argname

函数或聚集参数的名字。

这个命令会忽略掉参数的名字，因为参数的数据类型已经决定标识一个函数了。

argtype

函数或者聚集参数的数据类型。

left_type
right_type

操作符参数的数据类型（可以是方案限定）。指定NONE对于一个前缀或后缀操作符的缺失的参数。

PROCEDURAL

Z这是一个噪声词。

type_name

该转换的数据类型的名称。

lang_name

该转换的语言的名称。

示例

将hstore更新到2.0版本：

```
ALTER EXTENSION hstore UPDATE TO '2.0';
```

将hstore扩展的模式更改为 utils：

```
ALTER EXTENSION hstore SET SCHEMA utils;
```

将一个已存在的函数添加到hstore 扩展：

```
ALTER EXTENSION hstore ADD FUNCTION
populate_record(anyelement, hstore);
```

兼容性

ALTER EXTENSION是一个Greenplum数据库的扩展

另见

[CREATE EXTENSION, DROP EXTENSION](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改一个外部表的定义。

概要

```
ALTER EXTERNAL TABLE name action [ , ... ]
```

where *action* is one of:

```
ADD [COLUMN] new_column type
DROP [COLUMN] column [RESTRICT | CASCADE]
ALTER [COLUMN] column TYPE type
OWNER TO new_owner
```

描述

ALTER EXTERNAL TABLE 改变了一个已存在的外部表的定义。下面这些是ALTER EXTERNAL TABLE支持的动作。

- **ADD COLUMN** — 添加一个新列到外部表的定义。
- **DROP COLUMN** — 从一个外部表定义中删除一列。如果你删掉只读外部表的列，这只会修改Greenplum的表定义中修改。CASCADE关键字是用来当一些其他表依赖于这些列时使用，比如说一个视图依赖于这列。
- **ALTER COLUMN TYPE** — 改变一个表列的数据类型。
- **OWNER** — 将一个外部表的所有者改为指定用户。

用 [ALTER TABLE](#) 来在一个外部表中实现这些操作。

- 设置（修改）表模式
- 重命名表
- 重命名表中的列

你必须时外部表的所有者才能使用ALTER EXTERNAL TABLE或者ALTER TABLE。要改变一个外部表的模式，你还必须在模式上有CREATE的权限。要更改所有者，用户还必须是新拥有者的直接或间接成员，该角色必须对外部表的模式具有 CREATE特权。超级用户自动拥有这些权限。

用ALTER EXTERNAL TABLE或者 ALTER TABLE修改外部表的定义不会影响外部的数据。

ALTER EXTERNAL TABLE 和 ALTER TABLE命令 不会修改外部表的类型（读、写、网页），表的FORMAT信息，或者 外部数据的位置。要修改这些信息，你必须删除然后重建外部表的定义。

参数

name

要修改的现有外部表定义的名称（可以是方案限定）。

column

现有列的名称。

new_column

新列的名称

type

新列的数据类型，或者现有列的新数据类型。

new_owner

外部表的新所有者角色名。

CASCADE

自动删除那些依赖于要删除列的对象，比如说依赖于要删除列的视图。

RESTRICT

拒绝删除那些有依赖对象的列，这是默认的设置。

示例

往外部表的定义中添加新列：

```
ALTER EXTERNAL TABLE ext_expenses ADD COLUMN manager text;
```

改变外部表的所有者：

```
ALTER EXTERNAL TABLE ext_data OWNER TO jojo;
```

改变外部表的数据类型：

```
ALTER EXTERNAL TABLE ext_leads ALTER COLUMN acct_code TYPE integer
```

兼容性

ALTER EXTERNAL TABLE是Greenplum数据的扩展。标准SQL语句或者PostgreSQL中没有ALTER EXTERNAL TABLE语句。

另见

[CREATE EXTERNAL TABLE](#), [DROP EXTERNAL TABLE](#), [ALTER TABLE](#)

Parent topic: [SQL Command Reference](#)

WRAPPER

Greenplum数据库® 6.0文档

修改一个外部数据包装的定义。

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

概要

```
ALTER FOREIGN DATA WRAPPER name
    [ HANDLER handler_function | NO HANDLER ]
    [ VALIDATOR validator_function | NO VALIDATOR ]
    [ OPTIONS ( [ ADD | SET | DROP ] option ['value') [, ...
    ] ) ]
```

```
ALTER FOREIGN DATA WRAPPER name OWNER TO new_owner
ALTER FOREIGN DATA WRAPPER name RENAME TO new_name
```

描述

`ALTER FOREIGN DATA WRAPPER`修改一个外部数据包装的定义。第一个语句修改外部数据包装的支持函数或者一个一般性的选项。Greenplum 数据库要求只要有一个选项。第二和第三的语句修改外部数据包装的所有者或者名字。

只有超级用户才可以修改外部数据包装。另外，只有超级用户才可以拥有外部数据包装。

参数

name

外部数据包装的名字。

HANDLER *handler_function*

为外部数据包装提供一个新的处理函数。

NO HANDLER

指定外部数据包装不再拥有处理函数。



Note: 你不能访问没有处理的外部数据包装的外部表。
VALIDATOR *validator_function*

为外部数据包装提供一个新的验证函数。

当你修改验证函数时，外部数据包装、服务器、用户映射可能会变成不可用。在修改外部数据包装前你必须要确保这些选项时正确的。注意使用新验证者时，Greenplum数据库会检查在ALTER FOREIGN DATA WRAPPER 命令中用到的所有选项。

NO VALIDATOR

指定外部数据包装不再拥有验证函数。

OPTIONS ([ADD | SET | DROP] *option* ['*value*'] [, ...])

修改外部数据包装的选项。ADD, SET和DROP 指定了操作的动作。如果没有明确指出操作，默认的操作是ADD。选项的名字必须是唯一的。如果使用了验证函数，Greenplum会验证提供的选项名字和值。

OWNER TO *new_owner*

指定外部数据包装的新所有者，只有超级用户可以拥有外部数据包装。

RENAME TO *new_name*

指定外部数据包装的新名字。

示例

修改外部数据包装名为dbi的定义，增加一个新选项foo，然后移除名为bar 的选项：

```
ALTER FOREIGN DATA WRAPPER dbi OPTIONS (ADD foo '1', DROP 'bar');
```

修改外部数据包装名为dbi的验证函数为 bob.myvalidator：

```
ALTER FOREIGN DATA WRAPPER dbi VALIDATOR bob.myvalidator;
```

兼容性

ALTER FOREIGN DATA WRAPPER 适配ISO/IEC 9075-9 (SQL/MED)，但是HANDLER、VALIDATOR、OWNER TO和RENAME TO 选项是Greenplum数据库的扩展。

另见

[CREATE FOREIGN DATA WRAPPER](#), [DROP FOREIGN DATA WRAPPER](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

修改外表的定义。

概要

```
ALTER FOREIGN TABLE [ IF EXISTS ] name
    action [ , ... ]
ALTER FOREIGN TABLE [ IF EXISTS ] name
    RENAME [ COLUMN ] column_name TO new_column_name
ALTER FOREIGN TABLE [ IF EXISTS ] name
    RENAME TO new_name
ALTER FOREIGN TABLE [ IF EXISTS ] name
    SET SCHEMA new_schema
```

*action*是下列中的一个：

```
ADD [ COLUMN ] column_name column_type [ COLLATE
collation ] [ column_constraint [ ... ] ]
DROP [ COLUMN ] [ IF EXISTS ] column_name [ RESTRICT |
CASCADE ]
ALTER [ COLUMN ] column_name [ SET DATA ] TYPE
data_type
ALTER [ COLUMN ] column_name SET DEFAULT expression
ALTER [ COLUMN ] column_name DROP DEFAULT
ALTER [ COLUMN ] column_name { SET | DROP } NOT NULL
ALTER [ COLUMN ] column_name SET STATISTICS integer
ALTER [ COLUMN ] column_name SET ( attribute_option =
value [ , ... ] )
ALTER [ COLUMN ] column_name RESET ( attribute_option
[ , ... ] )
ALTER [ COLUMN ] column_name OPTIONS ( [ ADD | SET |
DROP ] option ['value'] [ , ... ] )
DISABLE TRIGGER [ trigger_name | ALL | USER ]
ENABLE TRIGGER [ trigger_name | ALL | USER ]
ENABLE REPLICA TRIGGER trigger_name
ENABLE ALWAYS TRIGGER trigger_name
OWNER TO new_owner
OPTIONS ( [ ADD | SET | DROP ] option ['value'] [ , ...
] )
```

描述

ALTER FOREIGN TABLE 修改一个已存在的外表的定义，命令有以下的几种形式：



ADD COLUMN

这种方式往外表中新增一列，和[CREATE FOREIGN TABLE](#)使用相同的语法。这个不像往通常的表中新加一列，它其实在存储层面不做任何操作，仅仅只是往现在访问的外表中定义了一些新列。

DROP COLUMN [IF EXISTS]

这种形式从外表中删除一列，如果有表之外的其他对象，比如说视图依赖于此列，必须指定 CASCADE关键字。如果指定了IF EXISTS关键字而这个列不存在，Greenplum数据库只会产生一个提醒而不是抛出错误。

IF EXISTS

如果指定了IF EXISTS关键字而外表不存在，Greenplum数据库只会产生一个提醒而不是抛出错误。

SET DATA TYPE

这种形式修改外表中一列的类型。

SET/DROP DEFAULT

这种形式设置或者删除了列的默认值，默认值只会应用在随后的INSERT或者 UPDATE命令；不会触发那些表中已经存在的行做修改。

SET/DROP NOT NULL

标志一个列允许或者不允许空值。

SET STATISTICS

这种形式为随后的ANALYZE操作设置每列的统计信息收集目标。更多细节参考[ALTER TABLE](#)中类似的形式。

`SET (attribute_option = value [, ...])`

`RESET (attribute_option [, ...])`

这种形式设置或者重置了每个属性的选项。更多细节参考[ALTER TABLE](#)中类似的形式。

DISABLE/ENABLE [REPLICA | ALWAYS] TRIGGER

这些形式配置了触发那些属于外表的触发器。更多细节参考[ALTER TABLE](#)中类似的形式。

OWNER

这种形式修改外表的所有者为指定的用户。

RENAME

RENAME修改一个外表的名字或者外表中个别列的名字。

SET SCHEMA

这种形式将外表移到其他的模式下。

OPTIONS ([ADD | SET | DROP] *option* ['*value*'] [, ...])

为外表修改选项。ADD, SET和DROP指定了要执行的操作。如果没有明确指出操作，默认的操作是ADD。选项名必须是唯一的。Greenplum数据库用外部数据包装器来验证名字和值。

你可以将除You can combine all of the actions except RENAME 和 SET SCHEMA之外的所有操作写在一个列表里让Greemplum数据库来并行的应用这些修改。比如，可以在一个命令中增加多列或者修改多列的属性。

你必须是拥有这个表才能使用ALTER FOREIGN TABLE操作。为了修改外表的模式，你还必须在新的模式下拥有CREATE的权限。为了修改所有者，你必须是新的角色的直接或者间接成员，而且那个角色必须在表的模式下拥有CREATE权限。（这些限制强制要求通过删除和重新创建表来更改所有者不会执行任何操作。但是，超级用户无论如何都可以更改任何表的所有权。）为了新增或者修改一个列的类型，你必须在数据类型上拥有USAGE权限。

参数

name

要更改的现有外表的名称（可能是模式限定的）。

column_name

新的或者已存在的列的名称。

new_column_name

已存在的列的新名称

new_name

外表的新的名称。

data_type

新的列的数据类型，或者已存在的列的新数据类型。

CASCADE

自动删除依赖于要删除列的对象（比如说，关联到列的视图）。

RESTRICT

拒绝删除有任何对象依赖的列。这是默认的表现。

trigger_name

要启用或者禁用的单个触发器名称。

ALL

禁用或启用属于外表的所有触发器。（如果任何触发器是内部生成的触发器，则需要超级用户权限。核心系统不会将此类触发器添加到外表，但附加代码可以这样做。）

USER

除内部生成的触发器外，禁用或启用属于外表的所有触发器。

new_owner

外表的新所有者的用户名。

new_schema

外表将移动到的模式的名称

Notes

关键字COLUMN是一个噪声词可以被忽略。

使用Consistency with the foreign server is not checked when a column is added or removed with ADD COLUMN或DROP COLUMN添加或删除列，添加NOT NULL约束，或者使用SET DATA TYPE修改列类型时，不会检查与外部服务器的一致性·您有责任确保表定义与远程端匹配。

参考[CREATE FOREIGN TABLE](#)来获取未来更多的验证参数的描述。

示例

标志一个列为not-null:

```
ALTER FOREIGN TABLE distributors ALTER COLUMN street SET  
NOT NULL;
```

修改一个外表的选项:

```
ALTER FOREIGN TABLE myschema.distributors  
OPTIONS (ADD opt1 'value', SET opt2 'value2', DROP opt3  
'value3');
```

兼容性

这些ADD, DROP和SET DATA TYPE的形式于SQL标准相符。其他形式为Greenplum数据库在SQL标准的扩展。在单个ALTER FOREIGN TABLE命令中指定多个操作的功能也是Greenplum数据库的扩展。

你可以使用ALTER FOREIGN TABLE ... DROP COLUMN来删除外表中唯一的列，留下一个零列的表。这也是SQL的扩展，允许存在零列的外表。

另见

[ALTER TABLE](#), [CREATE FOREIGN TABLE](#), [DROP FOREIGN TABLE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

修改函数的定义。

概要

```

ALTER FUNCTION name ( [ [ argmode] [argname] argtype [, ...]
] )
      action [, ...] [RESTRICT]

ALTER FUNCTION name ( [ [ argmode] [argname] argtype [, ...]
] )
      RENAME TO new_name

ALTER FUNCTION name ( [ [ argmode] [argname] argtype [, ...]
] )
      OWNER TO new_owner

ALTER FUNCTION name ( [ [ argmode] [argname] argtype [, ...]
] )
      SET SCHEMA new_schema

```

where *action* is one of:

```

{CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT |
STRICT}
{IMMUTABLE | STABLE | VOLATILE | [ NOT ] LEAKPROOF}
{[EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER}
EXECUTE ON { ANY | MASTER | ALL SEGMENTS }
COST execution_cost
SET configuration_parameter { TO | = } { value | DEFAULT }
SET configuration_parameter FROM CURRENT
RESET configuration_parameter
RESET ALL

```

描述

ALTER FUNCTION修改函数的定义。

你必须是函数的所有者才能使用ALTER FUNCTION。为了修改函数的模式，你还必须在新模式下拥有CREATE权限。为了修改所有者，你还必须是新角色的直接或间接成员，而且那个角色必须在函数模式上有用CREATE的权限。（这些限制强制执行。改变所有者不会通过删除和重新创建函数做任何你不能做的事情，但是，超级用户无论如

何都可以改变任何函数的所有权。)

参数

name

一个存在的函数的名称（可能是模式限定的）。

argmode

参数的模式：IN, OUT, INOUT或者VARIADIC。如果省略，默认值为IN。请注意ALTER FUNCTION实际上并不关注OUT参数，因为只需要输入参数来确定函数的身份。因此已经足够了对于只列出IN, INOUT 和 VARIADIC参数

argname

参数的名称。请注意，ALTER FUNCTION实际上并不关心参数名称，因为只需要参数数据类型来确定函数的身份。

argtype

函数参数（如果有）的数据类型（可以是方案限定）。

new_name

函数的新名称。

new_owner

函数的新拥有者。请注意，如果函数被标记为SECURITY DEFINER，随后它将作为新的所有者执行。

new_schema

该函数的新模式。

CALLED ON NULL INPUT

RETURNS NULL ON NULL INPUT

STRICT

CALLED ON NULL INPUT将该函数改为在某些或者全部参数为空值时可以被调用。RETURNS NULL ON NULL INPUT或者STRICT更改函数，以便如果其任何参数为空，则不会调用该函数；而是自动假设一个空的结果。参阅[CREATE FUNCTION](#)获取更多信息。

IMMUTABLE

STABLE

VOLATILE

将函数的波动性改为指定的设置。参阅[CREATE FUNCTION](#)获取更多信息

[EXTERNAL] SECURITY INVOKER

[EXTERNAL] SECURITY DEFINER

更改该函数是否为一个安全性定义者。关键词EXTERNAL为了SQL的一致性而被忽略。参阅[CREATE FUNCTION](#)获取更多有关此功能的信息。

LEAKPROOF

[CREATE FUNCTION](#)

更改函数是否被视为防漏的。参阅
功能的更多信息。

关于此

EXECUTE ON ANY

EXECUTE ON MASTER

EXECUTE ON ALL SEGMENTS

EXECUTE ON属性指定函数在查询执行过程中调用时执行的位置（主实例或段实例）。

EXECUTE ON ANY（默认值）表示该函数可以在主服务器或任何段实例上执行，并且无论执行的位置如何，它都会返回相同的结果。Greenplum数据库确定函数执行的位置。

EXECUTE ON MASTER表示该函数必须只在主实例上执行。

EXECUTE ON ALL SEGMENTS表示对于每次调用，该函数必须在所有主段实例上执行，而不是在主节点上执行。该函数的总体结果是来自所有段实例的UNION ALL结果。

EXECUTE ON属性的更多信息，请参阅[CREATE FUNCTION](#)。

COST *execution_cost*

更改该函数的估计执行代价。参阅[CREATE FUNCTION](#)获取更多信息。

configuration_parameter

value

当该函数被调用时，要对一个配置参数做出增加或者更改的赋值。如果*value*是DEFAULT或者，等价的RESET被使用，该函数本地的设置将会被移除，这样该函数会使用其环境中存在的值执行。使用RESET ALL可以清除所有函数本地的设置。SET FROM CURRENT应用会话的当前值当函数被输入时。

RESTRICT

忽略SQL标准。

注意

Greenplum数据库对某些定义的函数有STABLE或者VOLATILE这样的限制。参阅[CREATE FUNCTION](#)来获取更多信息。

示例

将integer类型的函数sqrt重命名为square_root：

```
ALTER FUNCTION sqrt(integer) RENAME TO square_root;
```

更改integer类型的sqrt函数的所有者为joe

```
ALTER FUNCTION sqrt(integer) OWNER TO joe;
```

更改integer类型的函数sqrt的模式为math:

```
ALTER FUNCTION sqrt(integer) SET SCHEMA math;
```

要调整一个函数的自动搜索路径:

```
ALTER FUNCTION check_password(text) RESET search_path;
```

兼容性

这个语句部分兼容SQL标准中的ALTER FUNCTION语句。该标准允许修改一个函数的更多属性，但是不提供重命名一个函数、标记一个函数为安全性定义者、为一个函数附加配置参数值或者更改一个函数的拥有者、模式或者稳定性等功能。该标准还需要RESTRICT关键字，它在Greenplum数据库中是可选的。

另见

[CREATE FUNCTION, DROP FUNCTION](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改角色名称或成员关系。

概要

```
ALTER GROUP groupname ADD USER username [ , ... ]
```

```
ALTER GROUP groupname DROP USER username [ , ... ]
```

```
ALTER GROUP groupname RENAME TO newname
```

描述

ALTER GROUP 更改了用户组的属性，这是一个被废弃的命令，不过为了向后兼容 还是被接受的。组和用户被更一般的概念角色所替代，参阅 [ALTER ROLE](#) 获取更多信息

前面两个语句往组里面添加或者删除一个用户，任何角色都可以当作是用户名或者组名。完成这种 任务的首选方式是使用[GRANT](#) 和 [REVOKE](#)。

参数

groupname

要修改的组（角色）的名称

username

要添加到组或从组中删除的用户（角色）。用户（角色）必须已经存在。

newname

组（角色）的新名称。

示例

要将用户添加到组中：



```
ALTER GROUP staff ADD USER karl, john;
```

从组中删除用户:

```
ALTER GROUP workers DROP USER beth;
```

兼容性

在SQL标准中没有ALTER GROUP语句

See Also

[ALTER ROLE](#), [GRANT](#), [REVOKE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改一个索引的定义。

概要

```
ALTER INDEX [ IF EXISTS ] name RENAME TO new_name
```

```
ALTER INDEX [ IF EXISTS ] name SET TABLESPACE
tablespace_name
```

```
ALTER INDEX [ IF EXISTS ] name SET ( storage_parameter =
value [ , ... ] )
```

```
ALTER INDEX [ IF EXISTS ] name RESET ( storage_parameter
[ , ... ] )
```

```
ALTER INDEX ALL IN TABLESPACE name [ OWNED BY role_name [ ,
... ] ]
SET TABLESPACE new_tablespace [ NOWAIT ]
```

描述

ALTER INDEX更改存在的一个索引的定义。有下列的一些子形式：

- **RENAME** — 修改索引的名字，对存储的数据无影响。
- **SET TABLESPACE** — 修改索引的表空间到指定的表空间，然后将分配给这个索引的数据文件也移到新的表空间。要修改索引的表空间，你必须在新的表空间有CREATE的权限。在当前数据库中一个表空间下所有的索引可以用ALL IN TABLESPACE形式移动，这会锁住所有的要移动的索引，然后移动每一个索引。这种方式也在OWNED BY形式中支持，这只会移动指定角色所拥有的索引。指定了NOWAIT选项的话，如果不能立即获取到需要的锁，命令将会失败。注意系统目录不会被这个命令移动，如果需要，请使用ALTER DATABASE或者显式调用调用ALTER INDEX。另见CREATE TABLESPACE。
- **IF EXISTS** — 如果索引不存在不会抛出错误，而会触发一个提醒。
- **SET** — 更改索引的特定于索引方法的存储参数。内方法都接受一个参数：*fillfactor*。索引的*fillfactor*是一个百分比，用于确定索引方法尝试打包索引页的程度。此命令不会立即修改索引内容。使用REINDEX重建索引以获得所需的效果。

- **RESET** — 将索引的存储参数重置为其默认值。内置索引方法都接受一个参数：*fillfactor*。与SET一样，可能需要 REINDEX 来完全更新索引。

参数

name

要修改的现有索引的名称（可选方案限定）。

new_name

索引的新名称。

tablespace_name

要移动索引的表空间。

storage_parameter

*index-method-specific*指定的存储参数名称。

value

*index-method-specific*指定的存储参数值，这可能是基于参数的数字或者词

注意

这些操作也可能用ALTER TABLE实现。

不允许更改系统目录索引的任何部分。

示例

修改一个索引的名字：

```
ALTER INDEX distributors RENAME TO suppliers;
```

移动索引到另外一个表空间：

```
ALTER INDEX distributors SET TABLESPACE fasttablespace;
```

要更改索引的填充因子（假设索引方法支持它）：

```
ALTER INDEX distributors SET (fillfactor = 75);
REINDEX INDEX distributors;
```

兼容性

ALTER INDEX 是Greenplum数据库的扩展。

另见

[CREATE INDEX](#), [REINDEX](#), [ALTER TABLE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)[ABORT](#)[ALTER AGGREGATE](#)[ALTER COLLATION](#)[ALTER CONVERSION](#)[ALTER DATABASE](#)[ALTER DEFAULT PRIVILEGES](#)[ALTER DOMAIN](#)[ALTER EXTENSION](#)[ALTER EXTERNAL TABLE](#)[ALTER FOREIGN DATA WRAPPER](#)[ALTER FOREIGN TABLE](#)[ALTER FUNCTION](#)[ALTER GROUP](#)[ALTER INDEX](#)[ALTER LANGUAGE](#)[ALTER OPERATOR](#)

更改过程语言的名称。

概要

```
ALTER LANGUAGE name RENAME TO newname
ALTER LANGUAGE name OWNER TO new_owner
```

描述

ALTER LANGUAGE修改指定数据库的过程语言定义。定义的修改支持包括重命名语言或者指定一个新的所有者。你必须是超级用户或者是这个语言的所有者才能使用ALTER LANGUAGE.

参数

name

语言的名称。

newname

语言的新名称

new_owner

语言的新的所有者。

兼容性

在SQL标准中没有ALTER LANGUAGE语句。

另见

[CREATE LANGUAGE, DROP LANGUAGE](#)[Parent topic: SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改操作符的定义。

概要

```
ALTER OPERATOR name ( {left_type | NONE} , {right_type | NONE} )
    OWNER TO new_owner
```

```
ALTER OPERATOR name ( {left_type | NONE} , {right_type | NONE} )
    SET SCHEMA new_schema
```

描述

ALTER OPERATOR 更改一个操作符的定义。 目前唯一可用的功能是修改操作符的所有者。

用户必须是操作符的所有者才能使用ALTER OPERATOR。 要更改所有者，用户必须是新角色的直接或间接成员，而且该角色必须在操作符的模式上有CREATE 权限。 (这种限制强制要求即使更改所有者也不能做那些通过删除或重建操作符所不能做到的事情。然而，超级用户可以任意修改操作符的所有权。)

参数

name

现有操作符的名称（可选方案限定）

left_type

操作符左操作数的数据类型；记为NONE 如果没有左操作数。

right_type

NONE操作符右操作数的数据类型；记为 NONE 如果操作符没有右操作数。

new_owner

操作符新的所有者。

new_schema

操作符新的模式。



示例

更改一个text类型的自定义操作符a @@ b:

```
ALTER OPERATOR @@ (text, text) OWNER TO joe;
```

兼容性

在SQL标准中没有 ALTEROPERATOR语句。 in the SQL standard.

另见

[CREATE OPERATOR](#), [DROP OPERATOR](#)

Parent topic: [SQL Command Reference](#)

CLASS

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改一个操作符类的定义。

概要

```
ALTER OPERATOR CLASS name USING index_method RENAME TO new_name
```

```
ALTER OPERATOR CLASS name USING index_method OWNER TO new_owner
```

```
ALTER OPERATOR CLASS name USING index_method SET SCHEMA new_schema
```

描述

`ALTER OPERATOR CLASS` 更改操作符类的定义。

用户必须是操作符类的所有者才能使用`ALTER OPERATOR CLASS`。要更改所有者，用户必须是新角色的直接或间接成员，而且该角色必须在操作符的模式上有`CREATE` 权限。（这种限制强制要求即使更改所有者也不能做那些通过删除或重建操作符所不能做到的事情。然而，超级用户可以任意修改操作符类的所有权。）

参数

name

现有操作符类的名称（可选方案限定）

index_method

操作符类索引方法的名称。

new_name

操作符类的新名称。

new_owner

操作符类的新的拥有者。

new_schema

操作符类的新模式。



兼容性

在SQL标准中没有ALTER OPERATOR CLASS语句

另见

[CREATE OPERATOR CLASS](#), [DROP OPERATOR CLASS](#)

Parent topic: [SQL Command Reference](#)

FAMILY

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLE

ALTER FOREIGN
DATA WRAPPER

ALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改操作符族的定义。

概要

```

ALTER OPERATOR FAMILY name USING index_method ADD
    { OPERATOR strategy_number operator_name ( op_type,
op_type ) [ FOR SEARCH | FOR ORDER BY sort_family_name ]
      | FUNCTION support_number [ ( op_type [ , op_type ] ) ]
funcname ( argument_type [ , ... ] )
    } [ , ... ]
}

ALTER OPERATOR FAMILY name USING index_method DROP
    { OPERATOR strategy_number ( op_type, op_type )
      | FUNCTION support_number [ ( op_type [ , op_type ] ) ]
    } [ , ... ]

ALTER OPERATOR FAMILY name USING index_method RENAME TO
new_name

ALTER OPERATOR FAMILY name USING index_method OWNER TO
new_owner

ALTER OPERATOR FAMILY name USING index_method SET SCHEMA
new_schema

```

描述

ALTER OPERATOR FAMILY更改操作符族的定义。 用户能够向族中添加操作符并提供函数支持、从族中删除它们，或者更改族的名称或所有者。

当通过ALTER OPERATOR FAMILY将操作符和函数支持添加到族中，则在族中它们不是任何指定操作符类的一部分，而仅是“松散”的存在于族内。这表示这些操作符和函数与该族的语义兼容，但是没有被任何特定索引的正确功能所要求。（要求操作符和函数作为操作符类的部分声明；参见 [CREATE OPERATOR CLASS](#)。）用户可以在任何时候将一个族中松散的成员从族中删除，但是除非将 类和所有依赖于这个类的索引都删除，否则一个操作符类的成员不能被删除。具有代表性的是，单一数据类型的操作符和函数是操作类的一部分，

因为在指定数据类型上的索引需要他们支持，而多数据类型操作符和函数则被作为该族的松散成员。

用户必须是超级用户才能使用ALTER OPERATOR FAMILY。（做这样的限制是因为一个错误的操作符族定义可能会迷惑服务器甚至让它崩溃）

ALTER OPERATOR FAMILY 目前不检测操作符族 定义是否包括该索引方法所要求的所有操作符和函数，也不检查操作符和函数是否形成了一个有理的集合。定义合法的操作符族是用户的责任。

OPERATOR 和 FUNCTION子句可以以任何顺序出现。

参数

name

现有操作符族的名称（可选限定模式）。

index_method

操作符族所应用的索引方法的名称。

strategy_number

与操作符族相关联的一个操作符的索引方法策略号。

operator_name

与操作符族相关联的一个操作符的名称（可选限定模式）。

op_type

在一个OPERATOR子句中，操作符的操作数的数据类型，或者用NONE来表示一个左一元或者右一元操作符。不同于CREATE OPERATOR CLASS中类似的语法，操作数数据类型总是必须被指定。在一个ADD FUNCTION子句中，指定该函数意图支持的操作数 数据类型（如果不同于该函数的输入数据类型）。对于B-树比较函数和哈希 函数，有必要指定*op_type*，因为该函数的输入数据类型 总是正确的。对于 B-树排序支持函数和GIN和GiST 操作符类中的所有函数，有必要指定该函数要使用的操作数数据类型。在一个 DROP FUNCTION 子句中，必须指定该函数要支持的操作数数据类型。

sort_family_name

在排序操作符中关联的描述排序顺序的现有btree 操作符名称。

如果即未指定FOR SEARCH也未指定FOR ORDER BY，则默认为FOR SEARCH。

support_number

索引方法与操作员族相关联的函数的支持程序编号。

funcname

作为该操作符族的一种索引方法支持过程的函数的名称（可以是

方案限定的)。

argument_types

该函数的参数数据类型。

new_name

该操作符族的新名称。

new_owner

该操作符族的新拥有者。

new_schema

该操作族中的新模式。

兼容性

在SQL标准中没有ALTER OPERATOR FAMILY语句。

注意

注意DROP语法只通过策略或者支持号以及输入数据类型指定该操作符族中的"slot"。占用这个槽的操作符或函数的名称不会被提及。还有，对于DROP FUNCTION要指定的类型是该函数意图支持的输入数据类型；对于SP_GIST和GIN索引可能无需对该函数的实际输入参数类型做任何事情。

因为索引机制在使用函数之前不会检查其上的访问权限，包括一个操作符族中的函数或操作符都等同于授予了其上的公共执行权限。这对于操作符族中很有用的这类函数来说，这通常不成问题。

操作符不应该由SQL函数定义。一个SQL函数很可能被内联到调用查询中，这将阻止优化器识别出该查询匹配一个索引。

在Greenplum Database 6.0之前，OPERATOR子句可以包括一个RECHECK选项。这个选项不再被支持了。Greenplum数据库现在运行中决定一个索引操作符是否是动态有损，这将允许更多有效处理那些操作符的是否有损的情况。

示例

下列示例命令为一个操作符族增加跨数据类型的操作符和支持函数，该操作符族已经包含用于数据类型int4以及int2的B-树操作符类：

```
ALTER OPERATOR FAMILY integer_ops USING btree ADD

    -- int4 vs int2
    OPERATOR 1 < (int4, int2) ,
    OPERATOR 2 <= (int4, int2) ,
    OPERATOR 3 = (int4, int2) ,
    OPERATOR 4 >= (int4, int2) ,
    OPERATOR 5 > (int4, int2) ,
    FUNCTION 1 btint42cmp(int4, int2) ,

    -- int2 vs int4
    OPERATOR 1 < (int2, int4) ,
    OPERATOR 2 <= (int2, int4) ,
    OPERATOR 3 = (int2, int4) ,
    OPERATOR 4 >= (int2, int4) ,
    OPERATOR 5 > (int2, int4) ,
    FUNCTION 1 btint24cmp(int2, int4) ;
```

再次移除这些项：

```
ALTER OPERATOR FAMILY integer_ops USING btree DROP

    -- int4 vs int2
    OPERATOR 1 (int4, int2) ,
    OPERATOR 2 (int4, int2) ,
    OPERATOR 3 (int4, int2) ,
    OPERATOR 4 (int4, int2) ,
    OPERATOR 5 (int4, int2) ,
    FUNCTION 1 (int4, int2) ,

    -- int2 vs int4
    OPERATOR 1 (int2, int4) ,
    OPERATOR 2 (int2, int4) ,
    OPERATOR 3 (int2, int4) ,
    OPERATOR 4 (int2, int4) ,
    OPERATOR 5 (int2, int4) ,
    FUNCTION 1 (int2, int4) ;
```

另见

[CREATE OPERATOR FAMILY](#), [DROP OPERATOR FAMILY](#), [ALTER OPERATOR CLASS](#), [CREATE OPERATOR CLASS](#), [DROP OPERATOR CLASS](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改一个协议的定义。

概要

```
ALTER PROTOCOL name RENAME TO newname
```

```
ALTER PROTOCOL name OWNER TO newowner
```

描述

ALTER PROTOCOL 更改一个协议的定义，只有协议的名字和所有者可以被更改。

用户必须拥有协议才可以使用ALTER PROTOCOL。要更改所有者，用户还必须是新角色的直接或者间接成员，而且新角色必须在该转换模式下拥有CREATE权限。

这些限制适当的确保修改所有者只能通过删除或重建协议。注意一个超级用户可以修改任何协议的所属关系。

参数

name

现有协议的名称（可选方案限定）。

newname

协议的新名称

newowner

协议的新所有者。

示例

重命名转换GPDBauth为 GPDB_authentication:

```
ALTER PROTOCOL GPDBauth RENAME TO GPDB_authentication;
```



更改转换GPDB_authentication 的所有者为 joe:

```
ALTER PROTOCOL GPDB_authentication OWNER TO joe;
```

兼容性

SQL标准中没有 ALTER PROTOCOL 语句。

另见

[CREATE EXTERNAL TABLE](#), [CREATE PROTOCOL](#)

Parent topic: [SQL Command Reference](#)

GROUP

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLE

ALTER FOREIGN
DATA WRAPPER

ALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改一个资源组的限制项。

概要

```
ALTER RESOURCE GROUP name SET group_attribute value
```

where *group_attribute* is one of:

```
CONCURRENCY integer
CPU_RATE_LIMIT integer
CPUSSET tuple
MEMORY_LIMIT integer
MEMORY_SHARED_QUOTA integer
MEMORY_SPILL_RATIO integer
```

描述

ALTER RESOURCE GROUP 改变一个资源组的限制项。只有超级用户才能修改资源组。

用户可以为控制角色最大活动并行度的资源组设置或者重置并行度的限制。用户还可以通过为资源组重置内存或者CPU资源来控制所有通过资源组提交的查询在每个Segment主机上可以使用的内存或CPU总量。

当用户更改资源组CPU资源的模式或者限制时，新的模式或限制会立即生效。

当用户更改为角色创建的资源组的内存限制时，如果当前资源使用率小于或等于新值且资源组中没有正在运行的事务，则会立即应用新的资源限制。如果当前资源使用量超过新内存限制值，或者其他资源组中存在运行事务在持有某些资源，则Greenplum数据库会推迟分配新限制，直到资源使用率落在新值的范围内。

当用户提升为外部组件创建的资源组的内存限制时，新的资源限制会随着资源可用而逐步生效。如果用户降低为外部组件创建的资源组的



内存限制时，则表现是基于特定组件的。比如说，如果用户降低为PL / Container运行时创建的资源组的内存限制，则正在运行的容器中的查询可能会因内存不足的错误而失败。

用户可以在单个 ALTER RESOURCE GROUP 调用中更改限制的类型。

参数

name

要更改的资源组的名称。

CONCURRENCY *integer*

用户为角色分配的资源组的允许的最大的事务并行数，包括活动和空闲的事务。任何在 CONCURRENCY 的限制值达到后要提交的事务都会进入队列，当一个运行的事务结束后，最早排队的事务会被执行。

CONCURRENCY 值必须是在[0 .. max_connections]范围内的整数。用户为角色创建的资源组默认的 CONCURRENCY 值为 20。

Note: 用户不可以将 admin_group 的 CONCURRENCY 值设为零(0)。

CPU_RATE_LIMIT *integer*

为当前资源组分配的CPU资源的百分比。资源组的CPU资源最小百分比为 1，最大为 100。为Greenplum数据库集群设置的所有资源租的 CPU_RATE_LIMIT 值总共不能超过 100。

如果用户为之前设置过CPUSSET的资源组修

改CPU_RATE_LIMIT，CPUSSET 会被禁用，保留的 CPU 核返回给 Greenplum 数据库，CPUSSET 被设置成 -1。

CPUSSET *tuple*

为当前资源组保留的CPU核。用户在 *tuple* 中指定的CPU核必须在系统上是可用的，而且用户不能将其他资源组中已指定的CPU核交叠分配给当前资源组

tuple 是以逗号分隔的单核数或核心间隔列表。您必须用单引号括起*tuple*，例如'1,3-4'。

如果用户为之前设置过CPU_RATE_LIMIT的资源组修

改CPUSSET，CPU_RATE_LIMIT 会被禁用，保留的CPU资源会返还给Greenplum数据库，然后 CPU_RATE_LIMIT 会被设置为 -1。

用户只能在启用了基于组管理资源的Greenplum数据库集群中为资源组修改CPUSSET。

MEMORY_LIMIT *integer*

为当前资源组分配的内存资源的百分比。资源组的内存资源最小

百分比为 percentage ，最大为 maximum ，为 minimum 。数据库集群中所有资源组定义的MEMORY_LIMIT总值不能超过100。

MEMORY_SHARED_QUOTA *integer*

为当前资源组中分配事务间共享的内存资源的百分比。资源组中最小的共享内存分配比为0，最大为100，默认的MEMORY_SHARED_QUOTA值为20。

MEMORY_SPILL_RATIO *integer*

资源组中发出的事务中内存密集型运算符的内存使用量阈值。资源组的最小内存溢出率百分比为0.最大值为100。MEMORY_SPILL_RATIO 的默认值是 20。

注意

用 `CREATE ROLE` 或者 `ALTER ROLE` 来将一个指定资源组分配给角色（用户）。

用户不能在显式事务或者子事务中提交 `ALTER RESOURCE GROUP` 命令。

示例

更改资源组的活动事务限制：

```
ALTER RESOURCE GROUP rgroup1 SET CONCURRENCY 13;
```

更新资源组的CPU限制：

```
ALTER RESOURCE GROUP rgroup2 SET CPU_RATE_LIMIT 45;
```

更新资源组的内存限制：

```
ALTER RESOURCE GROUP rgroup3 SET MEMORY_LIMIT 30;
```

从默认值中提升资源组的内存溢出率：

```
ALTER RESOURCE GROUP rgroup4 SET MEMORY_SPILL_RATIO 25;
```

为资源组保留CPU核 1：

```
ALTER RESOURCE GROUP rgroup5 SET CPUSED '1';
```

兼容性

ALTER RESOURCE GROUP 语句是Greenplum数据库的扩展，在PostgreSQL标准中没有该命令。

另见

[CREATE RESOURCE GROUP](#), [DROP RESOURCE GROUP](#), [CREATE ROLE](#), [ALTER ROLE](#)

Parent topic: [SQL Command Reference](#)

QUEUE

Greenplum数据库® 6.0文档

更改资源队列的限制。

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLE

ALTER FOREIGN
DATA WRAPPER

ALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

概要

```
ALTER RESOURCE QUEUE name WITH ( queue_attribute=value [ , ... ] )
```

其中 *queue_attribute* 是：

```
ACTIVE_STATEMENTS=integer
MEMORY_LIMIT='memory_units'
MAX_COST=float
COST_OVERCOMMIT={TRUE | FALSE}
MIN_COST=float
PRIORITY={MIN | LOW | MEDIUM | HIGH | MAX}
```

```
ALTER RESOURCE QUEUE name WITHOUT ( queue_attribute [ , ... ] )
```

其中 *queue_attribute* 是：

```
ACTIVE_STATEMENTS
MEMORY_LIMIT
MAX_COST
COST_OVERCOMMIT
MIN_COST
```

Note: 一个资源队列必须拥有 ACTIVE_STATEMENTS 或 MAX_COST 其中的一个值。不能将两者都从资源队列的 queue_attributes 中删除。

描述

`ALTER RESOURCE QUEUE` 更改一个资源队列的限制， 超级用户才能更改资源队列。一个资源队列必须拥有 ACTIVE_STATEMENTS 或者 MAX_COST 值（或者两者都有）。用户可以设置或者重置一个

资源队列的优先级来控制与资源队列相关的查询所使用的CPU可用资源，或者设置资源队列的内存限制来控制资源队列中所有提交的查询在一个segment主机上可以使用的内存总量。

`ALTER RESOURCE QUEUE WITHOUT` 删除之前在资源上指定的限制。一个资源队列必须拥有 `ACTIVE_STATEMENTS` 或者 `MAX_COST` 其中一个值。不能两者都从资源队列的 `queue_attributes` 中删除。

参数

`name`

要更改其限制的资源队列的名称。

`ACTIVE_STATEMENTS integer`

任何时刻系统中允许在该资源队列中的用户提交的活动语句的数量。`ACTIVE_STATEMENTS` 的值应该是一个大于0的整数。要把`ACTIVE_STATEMENTS` 重置为没有限制，则指定值为 -1。

`MEMORY_LIMIT 'memory_units'`

设置从此资源队列中的用户提交的所有语句的总内存配额。内存单位可以指定为kB, MB或GB。资源队列的最小内存配额为10MB。没有最大值；然而，查询执行时间的上边界受到segment主机的物理内存的限制。默认值为无限制(-1)。

`MAX_COST float`

任何时刻系统中允许在该资源队列中的用户提交的语句的查询优化器总代价。`MAX_COST`的值被指定为一个浮点数（例如100.00）或者还可以被指定为一个指数（例如 $1e+2$ ）。要把`MAX_COST`重置为没有限制，输入一个值-1.0。

`COST_OVERCOMMIT boolean`

如果资源队列受到基于查询代价的限制，那么管理员可以允许代价过量使用（默认`COST_OVERCOMMIT=TRUE`）。这意味着一个超过允许的代价阈值的查询将被允许运行，但只能在系统空闲时运行。如果指定`COST_OVERCOMMIT=FALSE`，超过代价限制的查询将总是被拒绝并且绝不会被允许运行。

`MIN_COST float`

代价低于此限制的查询将不会排队而是立即运行。代价是以取得的磁盘页为单位来衡量的。1.0等于一次顺序磁盘页面读取。`MIN_COST`的值被指定为浮点数（例如100.00）或者还能被指定为一个指数（例如 $1e+2$ ）。要把`MIN_COST`重置为没有限制，输入一个值 -1.0。

`PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX}`

设置与资源队列关联的查询的优先级。具有较高优先级的队列中的查询或语句将在竞争中获得更多的可用CPU资源份额。低优

先级队列中的查询可能会被延迟，同时执行更高优先级的查询

注意

GPORCA和Greenplum数据库原生优化器利用不同的查询耗费模型，所以同样的查询可能会计算出不同的消耗值。Greenplum数据库资源队列资源管理模型既不区分也不调整GPORCA和Postgres优化器的成本，它会直接使用优化器返回的消耗值来限制查询。

当基于资源队列的资源管理处于活动状态时，请使用资源队列的MEMORY_LIMIT和ACTIVE_STATEMENTS限制，而不是配置基于成本的限制。即使在使用GPORCA时，Greenplum数据库可能会回到使用postgres规划器进行某些查询，因此使用基于成本的限制可能会导致意外的结果。

示例

更改资源队列的活动查询限制：

```
ALTER RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=20);
```

更改资源队列的内存限制：

```
ALTER RESOURCE QUEUE myqueue WITH (MEMORY_LIMIT='2GB');
```

将资源队列的最大和最小查询代价限制重置为无限制：

```
ALTER RESOURCE QUEUE myqueue WITH (MAX_COST=-1.0,  
MIN_COST= -1.0);
```

将资源队列的查询代价限制重置为 3^{10} (或者是30000000000.0)不允许过量使用：

```
ALTER RESOURCE QUEUE myqueue WITH (MAX_COST=3e+10,  
COST_OVERCOMMIT=FALSE);
```

将与资源队列关联的查询的优先级重置为最小级别：

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

从资源队列中去除 MAX_COST 和 MEMORY_LIMIT 限制：

```
ALTER RESOURCE QUEUE myqueue WITHOUT (MAX_COST,  
MEMORY_LIMIT);
```

兼容性

ALTER RESOURCE QUEUE 语句是 Greenplum 数据库的扩展，在标准 PostgreSQL 中不存在这个命令。

另见

[CREATE RESOURCE QUEUE](#), [DROP RESOURCE QUEUE](#), [CREATE ROLE](#), [ALTER ROLE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改一个数据库角色（用户或组）。

概要

```
ALTER ROLE name [ [ WITH ] option [ ... ] ]
```

其中 *option* 可以是：

```
SUPERUSER | NOSUPERUSER
| CREATEDB | NOCREATEDB
| CREATEROLE | NOCREATEROLE
| CREATEEXTTABLE | NOCREATEEXTTABLE [ ( attribute='value' [ , ... ] )
    where attributes and values are:
        type='readable'|'writable'
        protocol='gpfdist'|'http'
| INHERIT | NOINHERIT
| LOGIN | NOLOGIN
| REPLICATION | NOREPLICATION
| CONNECTION LIMIT connlimit
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
| VALID UNTIL 'timestamp'
```

```
ALTER ROLE name RENAME TO new_name
```

```
ALTER ROLE { name | ALL } [ IN DATABASE database_name ] SET
configuration_parameter { TO | = } { value | DEFAULT }
ALTER ROLE { name | ALL } [ IN DATABASE database_name ] SET
configuration_parameter FROM CURRENT
ALTER ROLE { name | ALL } [ IN DATABASE database_name ] RESET
configuration_parameter
ALTER ROLE { name | ALL } [ IN DATABASE database_name ] RESET ALL
ALTER ROLE name RESOURCE QUEUE { queue_name | NONE }
ALTER ROLE name RESOURCE GROUP { group_name | NONE }
```

描述

ALTER ROLE 更改 Greenplum 数据库角色的属性。此命令有几种变体

WITH *option*

修改可以在 CREATE ROLE 中指定的大多数角色属性。 (包含了所有可能的属性，但不包括添加或删除成员身份的选项。) 那些选项使用 GRANT 和 REVOKE。) 在这个命令中没提到的属性将保留她们原来的值。超级用户可以为任何角色修改任何设置。拥有 CREATEROLE 权限的角色可以修改任意的这些设置，但是仅限于

非超级用户 和非replication角色。 普通用户只可以修改自己的密码。

RENAME

更改角色的名称。数据库超级用户可以重命名任何角色。角色有CREATEROLE特权 可以重命名非超级用户角色。无法重命名当前会话用户（以其他用户身份连接重命名角色）。因为MD5加密的密码使用角色名称作为密钥，如果密码为MD5加密，则重命名角色将清除其密码。

SET | RESET

为指定的配置参数更改角色的会话默认值，对于所有数据库，或者在IN DATABASE子句指定数据库时，仅对命名数据库中的会话进行更改。如果指定了all而不是角色名，则会更改所有角色的设置。在IN DATABASE 使用ALL实际上与使用命令ALTER DATABASE...SET...一样。

每当角色随后启动新会话时，指定的值将成为会话默认值，覆盖服务器配置文件（`postgresql.conf`）中存在的，或从`postgres`命令行接收到的任何设置。这只会发生在登录时，执行`SET ROLE` 或者 `SET SESSION AUTHORIZATION` 不会触发设置新的值

附加到角色的数据库特定设置将覆盖所有数据库的设置。特定数据库或特定角色的设置将覆盖所有角色的设置。

对于没有LOGIN权限的角色，会话默认值无效。普通角色可以更改自己的会话默认值。超级用户可以更改任何人的会话默认值。具有CREATEROLE权限的角色可以更改非超级用户角色的默认值。普通角色只能为自己设置默认值。某些配置变量不能这样设置，或者只能在超级用户发出命令时设置。有关所有用户可设置配置参数的信息，请参阅*Greenplum*数据库参考指南。只有超级用户才能更改所有数据库中所有角色的设置。

RESOURCE QUEUE

将角色分配给工作负载管理资源队列。在发出查询时，角色将受到分配资源队列的限制。指定NONE将角色分配给默认资源队列。一个角色只能属于一个资源队列。对于没有LOGIN特权的角色，会话默认值没有任何作用。参考CREATE RESOURCE QUEUE获取更多信息。

RESOURCE GROUP

为角色分配资源组。然后，角色将受制于为资源组配置的并发事务、内存和CPU限制。可以将单个资源组分配给一个或多个角色。不能将为外部组件创建的资源组分配给角色。参考CREATE RESOURCE GROUP获取更多信息。

参数

name

将被修改属性的角色名。

new_name

该角色的新名称。

database_name

将要设置配置参数的数据库名。

config_parameter=value

将指定配置参数的此角色会话默认值设置为给定值。如果 *value* 是 DEFAULT 指或者指定 RESET，则角色的指定参数设置会被删除，角色将在新会话里面继承系统层面的默认值，使用 RESET ALL 可以清除所有的角色特殊配置。 SET FROM CURRENT 保存会话的当前参数值作为角色指定的值。如果指定了 IN DATABASE，则只会为指定的角色数据库。当随后角色开启新的会话时，指定的参数值成为会话的默认值，覆盖服务器配置文件 (postgresql.conf) 中存在的,或从 postgres 命令行接收到的任何设置。

角色指定的变量设置只会在登录时生效; SET ROLE 和 [SET SESSION AUTHORIZATION](#) 不会处理指定角色的变量设置。

参阅 [服务器配置参数](#) 来获取更多关于用户可设置参数的信息。

group_name

分配给该角色的资源组名称。指定 *group_name* 为 NONE 删除角色当前分配的资源组，基于角色的能力分配一个默认资源组。 SUPERUSER 角色分配 admin_group 资源组，而 default_group 资源组则分配给非admin 角色。不能将为外部组件创建的资源组分配给角色。

queue_name

要分配用户级角色的资源队列的名称。只有 LOGIN 特权的角色可以分配给资源队列。要从资源队列中取消分配角色并将其置于默认资源队列中，请指定 NONE 角色只能属于一个资源队列。

SUPERUSER | NOSUPERUSER

CREATEDB | NOCREATEDB

CREATEROLE | NOCREATEROLE

CREATEUSER | NOCREATEUSER

CREATEUSER 和 NOCREATEUSER 已过期，但是仍被接受为 SUPERUSER 和 NOSUPERUSER。注意，它们不等同于 CREATEROLE 和 NOCREATEROLE 子句。

CREATEEXTTABLE | NOCREATEEXTTABLE [(attribute='value')]

如果 CREATEEXTTABLE 被指定，允许定义的角色创建外部表。

如果没有被指定，默认类型是 readable，并且默认协议

是 gpfdist。NOCREATEEXTTABLE (默认) 拒绝角色有创建外部表的能力。注意使用的外部表 file 或 execute 协议只能由超级用户创建。

INHERIT | NOINHERIT

LOGIN | NOLOGIN
 REPLICATION
 NOREPLICATION
 CONNECTION LIMIT connlimit
 PASSWORD password
 ENCRYPTED | UNENCRYPTED
 VALID UNTIL 'timestamp'

这些子句通过[CREATE ROLE](#)改变了原来设置的角色属性。

DENY deny_point
 DENY BETWEEN deny_point AND deny_point

DENY和DENY BETWEEN关键字设置了在登录时强制执行的基于时间的约束。DENY设置一天或一天的时间来拒绝访问。DENY BETWEEN设置访问被拒绝的间隔。两者都使用以下格式的参数*deny_point*：

DAY day [TIME 'time']

deny_point 两部分参数使用以下格式：：

对于 day：

{'Sunday' | 'Monday' | 'Tuesday' | 'Wednesday' | 'Thursday' | 'Friday' |
 'Saturday' | 0-6 }

对于 time：

{ 00-23 : 00-59 | 01-12 : 00-59 { AM | PM } }

DENY BETWEEN子句使用两种*deny_point*参数。

DENY BETWEEN *deny_point* AND *deny_point*

有关基于时间的约束和示例的更多信息，参阅Greenplum数据库管理员指南中的“管理角色和特权”。

DROP DENY FOR *deny_point*

该DROP DENY FOR子句从角色中删除基于时间的约束。它使用上述的*deny_point*参数。

有关基于时间的约束和示例的更多信息，参阅Greenplum数据库管理员指南中的“管理角色和特权”。

注意

使用[CREATE ROLE](#)新增角色， 使用[DROP ROLE](#)删除角色。

使用 [GRANT](#) 和 [REVOKE](#) 来增加和删除角色成员。

使用此命令指定未加密的密码时，必须小心。密码将以明文形式发送到服务器，也可能会记录在客户端的命令历史记录或服务器日志中。该 psql 命令行客户端包含一个元命令\password 可用于安全地更改角色的密码。

还可以将会话默认值与特定数据库而不是角色绑定。如果存在冲突，则特定于角色的设置将覆盖数据库特定的设置。参阅 [ALTER DATABASE](#)。

示例

更改角色的密码：

```
ALTER ROLE daria WITH PASSWORD 'passwd123';
```

删除角色的密码：

```
ALTER ROLE daria WITH PASSWORD NULL;
```

更改密码失效日期：

```
ALTER ROLE scott VALID UNTIL 'May 4 12:00:00 2015 +1';
```

使密码永久有效：

```
ALTER ROLE luke VALID UNTIL 'infinity';
```

赋予角色创建其他角色和新数据库的能力：

```
ALTER ROLE joelle CREATEROLE CREATEDB;
```

给角色一个非默认设置 `maintenance_work_mem` 参数：

```
ALTER ROLE admin SET maintenance_work_mem = 100000;
```

给角色一个非默认，指定数据库的参数 `client_min_messages` 值：

```
ALTER ROLE fred IN DATABASE devel SET client_min_messages = DEBUG;
```

将角色分配给资源队列：

```
ALTER ROLE sammy RESOURCE QUEUE poweruser;
```

授予创建可写外部表的角色权限：

```
ALTER ROLE load CREATEEXTTABLE (type='writable');
```

更改角色在星期日不允许登录访问：

```
ALTER ROLE user3 DENY DAY 'Sunday';
```

改变角色以消除星期日不允许登录访问的约束：

```
ALTER ROLE user3 DROP DENY FOR DAY 'Sunday';
```

指定一个新的资源组给角色：

```
ALTER ROLE parttime_user RESOURCE GROUP rg_light;
```

兼容性

ALTER ROLE 语句是 Greenplum 数据库的扩展

另见

[CREATE ROLE](#), [DROP ROLE](#), [ALTER DATABASESET](#), [CREATE RESOURCE GROUP](#), [CREATE RESOURCE QUEUE](#), [GRANT](#), [REVOKE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改一个模式定义。

概要

```
ALTER SCHEMA name RENAME TO newname
```

```
ALTER SCHEMA name OWNER TO newowner
```

描述

ALTER SCHEMA 更改一个模式定义。

用户必须拥有该模式才能去使用 ALTER SCHEMA. 要重命名一个模式， 用户还必须拥有该数据库的 CREATE 特权。要更改拥有者， 用户还必须是新拥有角色的一个直接或者间接成员，并且该角色必须具有该数据库上的 CREATE 特权。注意超级用户自动拥有所有这些特权。.

参数

name

现有模式的名称。

newname

该模式的新名称。新名称不能以pg_开头,因为这些名称被保留用于系统模式。

newowner

该模式的新的拥有者。

兼容性

SQL标准中没有ALTER SCHEMA 语句。



另见

[CREATE SCHEMA, DROP SCHEMA](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改一个序列发生器的定义。

概要

```
ALTER SEQUENCE [ IF EXISTS ] name [ INCREMENT [ BY ] increment ]
```

```
[ MINVALUE minvalue | NO MINVALUE ]
[ MAXVALUE maxvalue | NO MAXVALUE ]
[ START [ WITH ] start ]
[ RESTART [ [ WITH ] restart ] ]
[ CACHE cache ] [[ NO ] CYCLE]
[ OWNED BY {table.column | NONE} ]
```

```
ALTER SEQUENCE [ IF EXISTS ] name OWNER TO new_owner
```

```
ALTER SEQUENCE [ IF EXISTS ] name RENAME TO new_name
```

```
ALTER SEQUENCE [ IF EXISTS ] name SET SCHEMA new_schema
```

描述

ALTER SEQUENCE 更改一个现有序列发生器的参数。任何没有被明确设置的参数在 ALTER SEQUENCE 命令，都要维持他们之前的设置。

用户必须拥有该序列才能使用ALTER SEQUENCE。要更改一个序列的模式，用户还必须拥有新模式上的CREATE特权。注意超级用户自动拥有所有的特权。.

要改变所有者，用户必须是新角色的直接或者间接成员，而且新角色必须在序列的模式上拥有CREATE权限(这些限制迫使更改所有者不能执行删除和重新创建序列所无法做的任何事情。但是，超级用户仍然可以更改任何序列的所有权。)

参数

name

要修改的序列的名称（可选方案限定）。

IF EXISTS

如果序列不存在不会抛出错误，而只会触发一次提醒。
increment

子句 `INCREMENT BY increment` 是可选的。一个正值将产生一个上升序列，一个负值会产生一个下降序列。如果未被指定，则旧的增量值将被保持。

minvalue**NO MINVALUE**

可选的子句 `MINVALUE minvalue` 决定一个序列能产生的最小值。如果 `NO MINVALUE` 被指定，上升序列和下降序列的默认值分别是 1 和 -263-1。如果这些选项都没有被指定，将保持当前的最小值。

maxvalue**NO MAXVALUE**

可选子句 `MAXVALUE maxvalue` 决定一个序列能产生的最大值。如果 `NO MAXVALUE` 被指定，上升序列和下降序列的默认值分别是 263-1 和 -1。如果这些选项都没有被指定，将保持当前的最大值。

start

可选子句 `START WITH start` 更改记录的序列起始值。这对 `current` 序列值没有影响。它只是设置将来的 `ALTER SEQUENCE RESTART` 命令将使用的值。

restart

可选子句 `RESTART [WITH restart]` 更改序列的当前值。这等效于使用 `is_called = false` 调用 `setval(sequence, start_val, is_called)` 函数。指定的值将由 `nextval(sequence)` 函数的下一次调用返回。在没有 `restart` 值的情况下写入 `RESTART` 等同于提供由 `CREATE SEQUENCE` 记录或由 `ALTER SEQUENCE START WITH` 最后设置的开始值。

new_owner

序列新所有者的用户名。

cache

`CACHE cache` 子句中使序列号可以预先分配并存储在内存中，以加快访问速度。最小值为 1（一次只能生成一个值，即没有高速缓存）。如果未指定，则将保留旧的缓存值

CYCLE

可选的 `CYCLE` 关键字可用于在序列由升序或降序达到 `maxvalue` 或 `minvalue` 时使序列回绕。如果达到限制，则生成的下一个数字将是各自的 `minvalue` 或 `maxvalue`。

NO CYCLE

如果指定了可选的 `NO CYCLE` 关键字，则在序列达到最大值后，对 `nextval()` 的任何调用都将返回错误。如果未指定 `CYCLE` 或 `NO CYCLE`，则将保留旧的循环行为。

OWNED BY *table.column*

OWNED BY NONE

OWNED BY选项使序列与特定的表列相关联，这样，如果该列（或其整个表）被删除，该序列也将被自动删除。如果指定，则此关联替换该序列的任何先前指定的关联。指定的表必须具有相同的所有者，并且与序列具有相同的架构。指定OWNED BY NONE会删除任何现有的表列关联。

new_name

序列的新名称。

new_schema

序列的新模式。

注意

为了避免阻塞从同一序列中获取数字的并发事务，永远不会回滚ALTER SEQUENCE对序列生成参数的影响。这些更改将立即生效，并且不可逆。但是，OWNED BY，OWNER TO，RENAME TO和SET SCHEMA子句是普通的目录更新，可以回滚。

ALTER SEQUENCE不会立即影响除当前会话以外的具有预分配（缓存）序列值的会话中的nextval()结果。在注意到更改的序列生成参数之前，它们将用尽所有缓存的值。当前会话将立即受到影响。

由于历史原因，ALTER TABLE也可以与序列一起使用。但是序列允许的ALTER TABLE的唯一变体与上述形式等效。

示例

重启一个被称为serial的序列在105：

```
ALTER SEQUENCE serial RESTART WITH 105;
```

兼容性

ALTER SEQUENCE符合SQL标准，START WITH, OWNED BY, OWNER TO, RENAME TO和SET SCHEMA子句除外，它们是Greenplum数据库的扩展。

另见

[CREATE SEQUENCE](#), [DROP SEQUENCE](#), [ALTER TABLE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

修改一个外部服务器的定义。

g概要

```
ALTER SERVER server_name [ VERSION 'new_version' ]
    [ OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ...] ) ]
```

```
ALTER SERVER server_name OWNER TO new_owner
```

```
ALTER SERVER server_name RENAME TO new_name
```

Description

ALTER SERVER更改外部服务器的定义。命令的第一种形式更改服务器的版本字符串或通用选项。Greenplum数据库需要至少一个子句。该命令的第二和第三种形式更改服务器的所有者或名称。

要更改服务器，您必须是服务器的所有者。要更改所有者，用户必须：

- 拥有服务器。
- 成为新拥有角色的直接或间接成员。
- 在服务器的外部数据包装程序上具有USAGE特权。

超级用户自动满足所有这些条件。

参数

server_name

现有服务器的名称。

new_version

新的服务器版本。

OPTIONS ([ADD | SET | DROP] *option* ['*value*'] [, ...])



更改服务器的选项。 ADD, SET和DROP指定要执行的操作。 如果未明确指定任何操作，则默认操作为ADD。 选项名称必须唯一。 Greenplum数据库使用服务器的外部数据包装器库来验证名称和值。

`OWNER TO new_owner`

指定外部服务器的新所有者。

`RENAME TO new_name`

指定外部服务器的新名称。

Examples

通过添加连接选项来更改名为foo的服务器的定义：

```
ALTER SERVER foo OPTIONS (host 'foo', dbname 'foodb');
```

为名为foo的服务器更改名为host的选项，并设置服务器版本：

```
ALTER SERVER foo VERSION '9.1' OPTIONS (SET host 'baz');
```

兼容性

ALTER SERVER 符合 ISO/IEC 9075-9 (SQL/MED)。 OWNER TO 和 RENAME 选项是Greenplum数据库的扩展。

另见

[CREATE SERVER, DROP SERVER](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

 参考指南 SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改一个表的定义。

概要

```
ALTER TABLE [IF EXISTS] [ONLY] name
    action [, ... ]
```

```
ALTER TABLE [IF EXISTS] [ONLY] name
    RENAME [COLUMN] column_name TO new_column_name
```

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name
    RENAME CONSTRAINT constraint_name TO new_constraint_name
```

```
ALTER TABLE [IF EXISTS] name
    RENAME TO new_name
```

```
ALTER TABLE [IF EXISTS] name
    SET SCHEMA new_schema
```

```
ALTER TABLE ALL IN TABLESPACE name [ OWNED BY role_name [, ... ] ]
    SET TABLESPACE new_tablespace [ NOWAIT ]
```

```
ALTER TABLE [IF EXISTS] [ONLY] name SET
    WITH (REORGANIZE=true|false)
    | DISTRIBUTED BY ({column_name [opcclass]}) [, ... ])
    | DISTRIBUTED RANDOMLY
    | DISTRIBUTED REPLICATED
```

```
ALTER TABLE name
    [ ALTER PARTITION { partition_name | FOR (RANK(number))
    | FOR (value) } partition_action [...] ]
    partition_action
```

其中 *action* 是下列之一：

```
ADD [COLUMN] column_name data_type [ DEFAULT default_expr ]
    [column_constraint [, ... ]]
    [ COLLATE collation ]
    [ ENCODING (storage_directive [, ... ]) ]
DROP [COLUMN] [IF EXISTS] column_name [RESTRICT | CASCADE]
ALTER [COLUMN] column_name [ SET DATA ] TYPE type [COLLATE
collation] [USING expression]
ALTER [COLUMN] column_name SET DEFAULT expression
ALTER [COLUMN] column_name DROP DEFAULT
ALTER [COLUMN] column_name { SET | DROP } NOT NULL
ALTER [COLUMN] column_name SET STATISTICS integer
ALTER [COLUMN] column_name SET ( attribute_option=value [, ... ] )
ALTER [COLUMN] column_name RESET ( attribute_option [, ... ] )
ADD table_constraint [NOT VALID]
ADD table_constraint_using_index
```

```

VALIDATE CONSTRAINT constraint_name
DROP CONSTRAINT [IF EXISTS] constraint_name [RESTRICT | CASCADE]
DISABLE TRIGGER [trigger_name | ALL | USER]
ENABLE TRIGGER [trigger_name | ALL | USER]
CLUSTER ON index_name
SET WITHOUT CLUSTER
SET WITH OIDS
SET WITHOUT OIDS
SET (storage_parameter=value)
RESET (storage_parameter [, ... ])
INHERIT parent_table
NO INHERIT parent_table
OF type_name
NOT OF
OWNER TO new_owner
SET TABLESPACE new_tablespace

```

where *partition_action* is one of:

```

ALTER DEFAULT PARTITION
DROP DEFAULT PARTITION [IF EXISTS]
DROP PARTITION [IF EXISTS] { partition_name |
    FOR (RANK(number)) | FOR (value) } [CASCADE]
TRUNCATE DEFAULT PARTITION
TRUNCATE PARTITION { partition_name | FOR (RANK(number)) |
    FOR (value) }
RENAME DEFAULT PARTITION TO new_partition_name
RENAME PARTITION { partition_name | FOR (RANK(number)) |
    FOR (value) } TO new_partition_name
ADD DEFAULT PARTITION name [ ( subpartition_spec ) ]
ADD PARTITION [partition_name] partition_element
[ ( subpartition_spec ) ]
EXCHANGE PARTITION { partition_name | FOR (RANK(number)) |
    FOR (value) } WITH TABLE table_name
    [ WITH | WITHOUT VALIDATION ]
EXCHANGE DEFAULT PARTITION WITH TABLE table_name
[ WITH | WITHOUT VALIDATION ]
SET SUBPARTITION TEMPLATE (subpartition_spec)
SPLIT DEFAULT PARTITION
{ AT (list_value)
| START ([datatype] range_value) [INCLUSIVE | EXCLUSIVE]
    END ([datatype] range_value) [INCLUSIVE | EXCLUSIVE] }
[ INTO ( PARTITION new_partition_name,
        PARTITION default_partition_name ) ]
SPLIT PARTITION { partition_name | FOR (RANK(number)) |
    FOR (value) } AT (value)
[ INTO (PARTITION partition_name, PARTITION partition_name) ]

```

where *partition_element* is:

```

VALUES (list_value [...])
| START ([datatype] 'start_value') [INCLUSIVE | EXCLUSIVE]
    [ END ([datatype] 'end_value') [INCLUSIVE | EXCLUSIVE] ]
| END ([datatype] 'end_value') [INCLUSIVE | EXCLUSIVE]

```

```
[ WITH (partition_storage_parameter=value [, ... ]) ]
[ TABLESPACE tablespace ]
```

where *subpartition_spec* is:

```
subpartition_element [, ...]
```

and *subpartition_element* is:

```
DEFAULT SUBPARTITION subpartition_name
| [SUBPARTITION subpartition_name] VALUES (list_value [,...])
| [SUBPARTITION subpartition_name]
  START ([datatype]'start_value') [INCLUSIVE | EXCLUSIVE]
  [ END ([datatype]'end_value') [INCLUSIVE | EXCLUSIVE] ]
  [ EVERY ( [number / datatype]'interval_value') ]
| [SUBPARTITION subpartition_name]
  END ([datatype]'end_value') [INCLUSIVE | EXCLUSIVE]
  [ EVERY ( [number / datatype]'interval_value') ]
[ WITH (partition_storage_parameter=value [, ... ]) ]
[ TABLESPACE tablespace ]
```

where *storage_parameter* is:

```
appendoptimized={TRUE|FALSE}
blocksize={8192-2097152}
orientation={COLUMN|ROW}
compressstype={ZLIB|ZSTD|QUICLZ|RLE_TYPE|NONE}
compresslevel={0-9}
fillfactor={10-100}
oids[=TRUE|FALSE]
```

描述

ALTER TABLE 更改一个表的定义。下文描述了几种形式：

- **ADD COLUMN** — 向表中增加一个新列，使用和[CREATE TABLE](#)相同的语义。ENCODING 子句只有在追加和列存储表中有效
- **DROP COLUMN [IF EXISTS]** — 从表中删除列。请注意，如果删除用作Greenplum数据库分配键的表列，则表的分配策略将更改为DISTRIBUTED RANDOMLY.。涉及该列的索引和表约束也会自动删除。如果表外的任何内容都取决于列（例如视图），则需要指定CASCADE。如果指定了IF EXISTS且该列不存在，则不会引发任何错误；而是发出通知。
- **IF EXISTS** — 如果表不存在，请不要抛出错误。在这种情况下发

出通知。

- **SET DATA TYPE** — 此表单更改表的列的数据类型。请注意，您不能更改用作分发键或分区键的列数据类型。通过重新解析最初提供的表达式，涉及该列的索引和简单表约束将自动转换为使用新的列类型。可选的COLLATE子句为新列指定排序规则，如果省略，则排序规则是新列类型的默认排序规则。可选的USING子句指定如何从旧的值计算新的列值。如果省略，则默认转换与从旧数据类型到新数据类型的转换相同。如果没有从旧类型转换为新类型的隐式或赋值，则必须提供USING子句。

Note: 仅当查询中的所有列使用相同的排序规则时，GPORCA才支持排序规则。如果查询中的列使用不同的排序规则，则Greenplum使用Postgres Planner。

- **SET/DROP DEFAULT** — 设置或删除列的默认值。默认值仅适用于后续的INSERT或UPDATE命令。它们不会导致表中已有的行发生更改。
- **SET/DROP NOT NULL** — 更改是将列标记为允许空值还是拒绝空值。当列不包含空值时，只能使用SET NOT NULL。
- **SET STATISTICS** — 为后续的ANALYZE操作设置每个列的统计信息收集目标。可以在100到10000的范围内设置目标，也可以设置为-1以使用系统默认统计信息目标
(default_statistics_target) 恢复为目标。
- **SET (*attribute_option = value [, ...]*)**
RESET (*attribute_option [, ...]*) — 设置或重置每个属性选项。当前，唯一定义的按属性的选项是n_distinct和n_distinct_inherited，它们覆盖了后续ANALYZE操作所做的不同值估计数。n_distinct影响表本身的统计信息，而n_distinct_inherited影响表及其继承子级收集的统计信息。当设置为正值时，ANALYZE将假定该列恰好包含指定数量的不同非空值。当设置为负值（必须大于或等于-1）时，ANALYZE将假定列中不同的非空值的数量在表的大小中是线性的；确切的计数应通过将估计的表大小乘以给定数字的绝对值来计算。例如，值-1表示该列中的所有值都是不同的，而值-0.5表示每个值平均出现两次。当表的大小随时间变化时，这很有用，因为直到查询计划时间才执行表中行数的乘法。将值指定为0可恢复为通常估计不同值的数量
- **ADD *table_constraint* [NOT VALID]** — 使用与CREATE TABLE相同的语法向表（不仅仅是分区）添加新约束。当前仅将NOT VALID选项用于外键和CHECK约束。如果约束标记为NOT VALID，则Greenplum数据库将跳过可能冗长的初始检查，以验证表中的所有行均满足约束。约束将仍然针对后续的插入或更新（即，对于外键而言，除非在引用表中有匹配的行，否则它们将失败；对于外键，除非新行与指定的检查匹配，否则它们将失败）约束

束)。但是，除非使用VALIDATE CONSTRAINT选项对其进行验证，否则数据库将不假定该约束对表中的所有行均有效。创建表时将跳过约束检查，因此CREATE TABLE语法不包括此选项。

- **VALIDATE CONSTRAINT** — 该形式通过扫描表以确保没有不满足该约束的行，从而验证了以前创建为NOT VALID的外键约束。如果约束已被标记为有效，则什么也不会发生。将验证与约束的初始创建分开的好处是，与约束创建相比，验证对表的锁定更少。
- **ADD *table_constraint_using_index*** — 根据现有的唯一索引将新的PRIMARY KEY或UNIQUE约束添加到表中。索引的所有列都将包含在约束中。索引不能具有表达式列，也不能是部分索引。另外，它必须是具有默认排序顺序的b树索引。这些限制确保索引等于由常规ADD PRIMARY KEY或ADD UNIQUE命令建立的索引。如果指定了PRIMARY KEY，并且索引的列尚未标记为NOT NULL，则此命令将尝试对每个此类列执行ALTER COLUMN SET NOT NULL。这需要全表扫描，以验证列不包含空值。在所有其他情况下，这是一个快速的操作。
如果提供了约束名称，那么索引将被重命名以匹配约束名称。否则，约束将被命名为与索引相同。
执行此命令后，索引将由约束“拥有”，就像使用常规ADD PRIMARY KEY或ADD UNIQUE命令构建索引一样。特别是，删除约束将使索引也消失。
Note: 在需要添加新约束而不长时间阻止表更新的情况下，使用现有索引添加约束可能会有所帮助。为此，请使用CREATE INDEX CONCURRENTLY 创建索引，然后使用此语法将其安装为正式约束。请参见下面的示例。
- **DROP CONSTRAINT [IF EXISTS]** — 将指定的约束放在表上。如果指定了IF EXISTS且该约束不存在，则不会引发任何错误。在这种情况下，将发出通知。
- **DISABLE/ENABLE TRIGGER** — 禁用或启用属于该表的触发器。禁用的触发器对于系统仍然是已知的，但是在其触发事件发生时不会执行。对于延迟的触发器，将在事件发生时而不是在实际执行触发器功能时检查启用状态。可以禁用或启用由名称指定的单个触发器，或表上的所有触发器，或仅由用户创建的触发器。禁用或启用约束触发器需要超级用户特权。
Note: Greenplum数据库中不支持触发器。由于Greenplum数据库的并行性，触发器通常具有非常有限的功能。
- **CLUSTER ON/SET WITHOUT CLUSTER** — 选择或删除默认索引以用于将来的CLUSTER操作。它实际上并没有重新群集表。请注意，建议不要使用CLUSTER对Greenplum数据库中的表进行物理重新排序，因为它会花费很长时间。最好使用CREATE TABLE AS重新创建表并按索引列对其进行排序。
Note: 追加优化表不支持CLUSTER ON。

- **SET WITH OIDS** — 往表中添加一个系统列 oid。如果表中已有OID列则不会做任何操作，这与ADD COLUMN oid oid的操作并不相等，ADD COLUMN oid oid会增加一个正常的名为oid的正常列，而不是增加一个系统列。OIDs不允许增加在分区表或者追加优化列组织表。

Warning: Greenplum强烈建议您在创建表时不要启用OIDS。在大型表（例如典型的Greenplum数据库系统中的表）上，对表行使使用OID可能会导致32位OID计数器的折回。一旦计数器回绕，就不能再认为OID是唯一的，这不仅使OID对用户应用程序无用，而且还会在Greenplum数据库系统目录表中引起问题。此外，从表中排除OID会使每行将表存储在磁盘上所需的空间减少了每行4个字节，从而略微提高了性能。

- **SET WITHOUT OIDS** — 从表中删除OID系统列。
- **SET (FILLFACTOR = *value*) / RESET (FILLFACTOR)** — 更改表的填充因子。表格的填充系数是10到100之间的百分比。默认值为100（完全打包）。当指定较小的填充因子时，INSERT操作仅将表页面打包到指定的百分比；每个页面上的剩余空间都保留用于更新该页面上的行。这样，UPDATE就有机会将行的更新副本与原始副本放置在同一页面上，这比将其放置在另一页面上更为有效。对于一个条目从不更新的表，完全打包是最佳选择，但在更新频繁的表中，较小的填充因子是合适的。请注意，此命令不会立即修改表内容。您将需要重写表以获得所需的效果。可以使用VACUUM或强制表重写的ALTER TABLE形式之一来完成。
- **SET DISTRIBUTED** — 更改表的分配策略。更改哈希分发策略，或更改为复制策略或从复制策略更改将导致表数据在磁盘上进行物理重新分发，这可能会占用大量资源。
- **INHERIT *parent_table* / NO INHERIT *parent_table*** — 添加或删除目标表作为指定父表的子表。对父级的查询将包括其子表的记录。要作为子项添加，目标表必须已经包含与父项相同的所有列（它也可以具有其他列）。这些列必须具有匹配的数据类型，并且如果它们在父级中具有NOT NULL约束，那么它们在子级中也必须具有NOT NULL约束。对于父级的所有CHECK约束，还必须有匹配的子表约束，但在父级中标记为不可继承的（即用ALTER TABLE ... ADD CONSTRAINT ... NO INHERIT创建）的约束除外。匹配的所有子表约束均不得标记为不可继承。当前不考虑UNIQUE, PRIMARY KEY和FOREIGN KEY约束，但是将来可能会改变。
- **OF *type_name*** — 这种形式将表链接到复合类型，就像CREATE TABLE OF已经形成了它一样。该表的列名和类型列表必须与组合类型的列表完全匹配；oid系统列的存在可以不同。该表不得从任何其他表继承。这些限制确保CREATE TABLE OF将允许等效的表定义。

- **NOT OF** — 这种形式将类型化表与其类型分离。
- **OWNER** — 将表，序列或视图的所有者更改为指定的用户。
- **SET TABLESPACE** — 将表的表空间更改为指定的表空间，并将与表关联的数据文件移动到新表空间。表中的索引（如果有）不会移动；但是可以使用其他SET TABLESPACE命令分别移动它们。
可以使用ALL IN TABLESPACE表单移动表空间中当前数据库中的所有表，该表单将锁定所有要先移动的表，然后再移动每个表。此表单还支持OWNED BY，该操作仅移动指定角色所拥有的表。如果指定了NOWAIT选项，则如果该命令无法立即获取所有必需的锁，则该命令将失败。请注意，此命令不会移动系统目录，请根据需要使用ALTER DATABASE或显式ALTER TABLE调用。
information_schema关系不视为系统目录的一部分，将被移动。另请参见CREATE TABLESPACE。如果更改分区表的表空间，则所有子表分区也将移至新表空间。
- **RENAME** — 更改表（或索引，序列或视图）的名称，表中单个列的名称或表的约束的名称。对存储的数据没有影响。请注意，Greenplum数据库分发密钥列不能重命名。
- **SET SCHEMA** — 将表移到另一个架构。表列拥有的关联索引，约束和序列也将移动。
- **ALTER PARTITION | DROP PARTITION | RENAME PARTITION | TRUNCATE PARTITION | ADD PARTITION | SPLIT PARTITION | EXCHANGE PARTITION | SET SUBPARTITION TEMPLATE** — 更改分区表的结构。在大多数情况下，您必须遍历父表才能更改其子表分区之一。

Note: 如果将分区添加到具有子分区编码的表中，则新分区将继承该子分区的存储指令。有关压缩设置优先级的更多信息，请参阅*Greenplum Database Administrator Guide*中的“使用压缩”。

除 RENAME 和 SET SCHEMA 之外，所有作用于单个表的 ALTER TABLE 形式都可以组合成多个更改列表以一起应用。例如，可以在单个命令中添加几列和/或更改几列的类型。这对于大型表尤其有用，因为只需要对表进行一次遍历。

您必须拥有该表才能使用 ALTER TABLE。要更改表的架构或表空间，您还必须对新架构或表空间具有 CREATE 特权。要将表添加为父表的新子级，您还必须拥有父表。要更改所有者，您还必须是新拥有角色的直接或间接成员，并且该角色必须对表的架构具有 CREATE 特权。要添加列或更改列类型或使用 OF 子句，您还必须对数据类型具有 USAGE 特权。超级用户自动具有这些特权。

Note: 如果表具有多个分区，表具有压缩功能或表的块大小很大，则内存使用量会显著增加。如果与该表关联的关系的数量很大，则这种

情况可能会迫使对该表进行的操作使用更多的内存。例如，如果该表是一个CO表并具有大量列，则每个列都是一个关系。诸如ALTER TABLE ALTER COLUMN之类的操作将打开表中的所有列，以分配关联的缓冲区。如果CO表具有40列和100个分区，并且这些列被压缩并且块大小为2 MB（系统系数为3），则系统尝试分配24 GB，即 $(40 \times 100) \times (2 \times 3)$ MB或24 GB。

参数

ONLY

仅对指定的表名执行操作。如果不使用 ONLY关键字，则将在命名表以及与该表关联的任何子表分区上执行该操作。

Note: 只允许在父表或子表中添加或删除列，或更改列的类型。

父表及其后代必须始终具有相同的列和类型。

name

要更改的现有表的名称（可能是模式限定的）。如果ONLY指定，则仅更改该表。如果未指定ONLY，则更新表及其所有后代表（如果有）。

Note: 约束只能添加到整个表，不能添加到分区。由于该限制，*name*参数只能包含表名，而不能包含分区名。

column_name

新列或现有列的名称。请注意，Greenplum数据库分发键列必须格外小心。更改或删除这些列可以更改表的分发策略。

new_column_name

现有列的新名称。

new_name

表的新名称。

type

新列的数据类型，或现有列的新数据类型。如果更改Greenplum分布键列的数据类型，则只能将其更改为兼容类型（例如，text到varchar可以，但text到int则不能）。

table_constraint

表的新表约束。请注意，Greenplum数据库当前不支持外键约束。此外，表仅允许一个唯一约束，并且唯一性必须在Greenplum数据库分发密钥内。

constraint_name

要删除的现有约束的名称。

CASCADE

自动删除依赖于已删除列或约束的对象（例如，引用该列的视图）。

RESTRICT

如果有任何相关对象，则拒绝删除列或约束。这是默认行为。

trigger_name

要禁用或启用的单个触发器的名称。请注意，Greenplum数据库不支持触发器。

ALL

禁用或启用属于该表的所有触发器，包括与约束相关的触发器。如果任何触发器是内部生成的约束触发器（例如用于实现外键约束或可延迟的唯一性和排除约束的触发器），则这需要超级用户特权。

USER

禁用或启用属于该表的所有触发器，但内部生成的约束触发器（例如用于实现外键约束或可延迟的唯一性和排除约束的触发器除外）除外。

index_name

表应标记为集群的索引名称。请注意，建议不要使用CLUSTER对Greenplum数据库中的表进行物理重新排序，因为它会花费很长时间。最好使用CREATE TABLE AS重新创建表并按索引列对其进行排序。

FILLFACTOR

设置表格的填充系数百分比。

value

FILLFACTOR参数的新值，介于10到100之间的百分比。默认值为100。

DISTRIBUTED BY ({*column_name* [*opclass*]}) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED

指定表的分发策略。更改哈希分发策略会导致物理上重新分配表数据，这可能会占用大量资源。如果声明相同的哈希分配策略或从哈希更改为随机分配，则除非声明SET WITH (REORGANIZE=true)，否则不会重新分配数据。

更改为复制的分发策略或从复制的分发策略更改将导致表数据被重新分发。

REORGANIZE=true|false

当哈希分配策略未更改或从哈希更改为随机分配，并且无论如何都希望重新分配数据时，请使用REORGANIZE=true。

parent_table

父表要与此表关联或取消关联。

new_owner

表的新所有者的角色名称。

new_tablespace

该表将被移动到的表空间的名称。

new_schema

表将被移动到的模式的名称。

parent_table_name

更改分区表时，顶级父表的名称。

ALTER [DEFAULT] PARTITION

如果要更改的分区比第一级分区深，请使用ALTER

PARTITION子句指定要更改层次结构中的哪个子分区

DROP [DEFAULT] PARTITION

删除指定的分区。如果分区具有子分区，则子分区也会自动删除。

TRUNCATE [DEFAULT] PARTITION

截断指定的分区。如果分区具有子分区，则子分区也会自动被截断。

RENAME [DEFAULT] PARTITION

更改分区的分区名称（而不是关系名称）。分区表是使用以下命名约定创建的：

<parentname>_<level>_prt_<partition_name>.

ADD DEFAULT PARTITION

将默认分区添加到现有分区设计中。当数据与现有分区不匹配时，会将其插入默认分区。没有默认分区的分区设计将拒绝与现有分区不匹配的传入行。必须为默认分区指定名称。

ADD PARTITION

partition_element - 使用表（范围或列表）的现有分区类型，定义要添加的新分区的边界。

name - 此新分区的名称。

VALUES - 对于列表分区，定义分区将包含的值。

START - 对于范围分区，定义分区的起始范围值。默认情况下，起始值为INCLUSIVE。例如，如果您声明开始日期为“2016-01-01”，则分区将包含所有大于或等于“2016-01-01”的日期。通常，START表达式的数据类型与分区键列的类型相同。如果不是这种情况，则必须显式转换为预期的数据类型。

END - 对于范围分区，定义分区的结束范围值。默认情况下，最终值为EXCLUSIVE。例如，如果您声明结束日期为“'2016-02-01”，则分区将包含所有小于但不等于“'2016-02-01”的日期。通常，END表达式的数据类型与分区键列的类型相同。如果不是这种情况，则必须显式转换为预期的数据类型。

WITH - 设置分区的表存储选项。例如，您可能希望将较旧的分区作为追加优化表，而将较新的分区作为常规堆表。有关存储选项的说明，请参见[CREATE TABLE](#)。

TABLESPACE - 要在其中创建分区的表空间的名称。

subpartition_spec - 仅允许在没有子分区模板的情况下创建的分区设计。声明要添加的新分区的子分区规范。如果分区表最初是使用子分区模板定义的，则该模板将用于自动生成子分区。

EXCHANGE [DEFAULT] PARTITION

将另一个表交换到分区层次结构中，替换为现有分区的位置。

在多级分区设计中，您只能交换最低级别的分区（包含数据的分区）。

Greenplum数据库服务器配置参

数gp_enable_exchange_default_partition控

制EXCHANGE DEFAULT PARTITION子句的可用性。该参数的默认值是off。该子句不可用，如果在ALTER TABLE命令中指定了该子句，则Greenplum数据库将返回错误。

有关该参数的信息，请参阅[服务器配置参数](#)。

Warning: 交换默认分区之前，必须确保要交换的表中的数据（新的默认分区）对默认分区有效。例如，新的默认分区中的数据不得包含在分区表的其他叶子分区中有效的数据。否则，由GPORCA执行的具有交换的默认分区的分区表查询可能会返回错误的结果。

WITH TABLE *table_name* - 您要交换到分区设计中的表的名称。您可以交换一个表，其中表数据存储在数据库中。例如，该表是使用CREATE TABLE命令创建的。该表必须具有与父表相同的列数，列顺序，列名，列类型和分发策略。

使用EXCHANGE PARTITION子句，您还可以将可读的外部表（使用CREATE EXTERNAL TABLE命令创建）交换到分区层次结构中，而不是现有的叶子子分区。如果您指定了可读的外部表，则还必须指定WITHOUT VALIDATION子句，以针对要交换的分区的CHECK约束跳过表验证。

如果分区表包含具有检查约束或NOT NULL约束的列，则不支持与外部表交换叶子分区。

您无法与复制表交换分区。不支持将分区与分区表或分区表的子分区交换。

WITH | WITHOUT VALIDATION - 验证表中的数据是否与您要交换的分区的CHECK约束相匹配。默认设置是根据CHECK约束验证数据。

Warning: 如果指定WITHOUT VALIDATION子句，则必须确保针对现有子叶分区交换的表中的数据对于分区上的CHECK约束是有效的。

SET SUBPARTITION TEMPLATE

修改现有分区的子分区模板。设置新的子分区模板后，所有添加的新分区将具有新的子分区设计（现有分区不会被修改）。

SPLIT DEFAULT PARTITION

分割默认分区。在多级分区中，只能拆分范围分区，而不能拆分列表分区，并且只能拆分最低级别的默认分区（包含数据的分区）。拆分默认分区将创建一个包含指定值的新分区，并保留默认分区，其中包含与现有分区不匹配的任何值。

AT - 对于列表分区表，指定一个列表值，该值应用作拆分条件。

START - 对于范围分区表，指定新分区的起始值。

END - 对于范围分区表，指定新分区的结束值。

INTO - 允许您为新分区指定名称。使用INTO子句拆分默认分区时，指定的第二个分区名称应始终为现有默认分区的名称。如果您不知道默认分区的名称，则可以使用*pg_partitions*视图进行查找。

SPLIT PARTITION

将现有分区分为两个分区。在多级别分区中，只能拆分范围分区，而不能拆分列表分区，并且只能拆分最低级别的分区（包含数据的分区）。

AT - 指定一个单一值，该值应用作分割条件。该分区将分为两个新分区，指定的分割值是后一个分区的起始范围。

INTO - 允许用户为分裂创建两个新分区指定名字。

partition_name

给定的分区名称。

FOR (RANK(number))

对于范围分区，分区在范围中的排名。

FOR ('value')

通过声明一个落在分区边界说明中的值来指定一个分区。如果用FOR声明的值匹配一个分区和它的一个子分区（例如，如果值是一个日期并且表先按月分区然后按日分区），那么FOR将在第一个找到匹配的层次上操作（例如，每月的分区）。如果用户的目的是在子分区上操作，则必须按如下的方式声明：ALTER TABLE *name* ALTER PARTITION FOR ('2016-10-01') DROP PARTITION FOR ('2016-10-01');

注解

ALTER TABLE 命令中指定的表名不能是一个表中的分区名。

在修改或者删除作为Greenplum数据库分布键一部分的列时要特别小心，因为这可能会改变表的分布策略。.

Greenplum数据库当前不支持外键约束。对于要在Greenplum数据库中实施的唯一约束，表必须被哈希分布（不能用DISTRIBUTED RANDOMLY），并且所有的分布键列必须和唯一约束列中前部的列相同。

增加CHECK或者NOT NULL约束要求扫描表以验证现有的行是否符合约束。

当使用ADD COLUMN添加列时，表中的所有现有行都使用该列的默认值初始化，如果未指定DEFAULT子句，则初始化为NULL。添加具有非空默认值的列或更改现有列的类型将需要重写整个表和索引。作为例外，如果USING子句不更改列的内容，并且旧类型可以强制转换为新类型或新类型不受限制的域，则不需要重写表，但是受影响列上的任何索引都必须仍在重建中。添加或删除系统oid列还需要重写整个表。对于大型表，表和/或索引的重建可能会花费大量时间；并且暂时需要多达两倍的磁盘空间。

您可以在单个ALTER TABLE命令中指定多个更改，这些更改将在表上一次传递。

DROP COLUMN表单不会物理删除列，而只是使它对SQL操作不可见。表中随后的插入和更新操作将为该列存储一个空值。因此，删除列很快，但是不会立即减小表的磁盘大小，因为删除的列所占用的空间不会被回收。随着现有行的更新，空间将随着时间的推移而回收。但是，如果删除系统oid列，则表将立即被重写。

要强制立即回收被删除的列占用的空间，您可以执行ALTER TABLE的一种形式来重写整个表。这将导致重建的每一行，并将删除的列替换为空值。

ALTER TABLE的重写形式不是MVCC安全的。在表重写之后，如果并发事务使用的是在重写发生之前拍摄的快照，则该表将对并发事务显示为空。有关更多详细信息，请参见[MVCC Caveats](#)。

SET DATA TYPE的USING选项实际上可以指定涉及该行的旧值的任何表达式。也就是说，它可以引用其他列以及要转换的列。这允许使用SET DATA TYPE语法完成非常通用的转换。由于具有这种灵活性，因此USING表达式不会应用于列的默认值（如果有）；结果可能不是默认值所需的常量表达式。这意味着当没有从旧类型到新类型的隐式或赋值转换时，即使提供了USING子句，SET DATA TYPE也可能无法转换默认值。在这种情况下，请使用DROP DEFAULT删除默认值，执行ALTER TYPE，然后使用SET DEFAULT添加合适的新默认值。类似的考虑适用于涉及该列的索引和约束。

如果表已分区或具有任何后代表，则不允许在父表中添加，重命名或更改列的类型或重命名继承的约束，而无需对后代进行相同的操作。这样可以确保后代始终具有与父代匹配的列。

要查看分区表的结构，可以使用视图[pg_partitions](#)。该视图可以帮助您识别您可能要更改的特定分区。

仅当后代不从任何其他父级继承该列并且从未对该列进行独立定义时，递归DROP COLUMN操作才会删除后代表的列。非递归DROP COLUMN（仅ALTER TABLE ONLY ... DROP COLUMN）从不删除任何后代列，而是将它们标记为独立定义而不是继承。

TRIGGER, CLUSTER, OWNER和 TABLESPACE操作永远不会递归到后代表。也就是说，它们始终像指定ONLY那样起作用。仅对未标记为NO INHERIT的CHECK约束重复添加约束。

如果包含已被交换以使用外部表的叶子分区的分区表上的数据没有更改，则支持这些ALTER PARTITION操作。否则，将返回错误。

- Adding or dropping a column.
- Changing the data type of column.

分区表不支持这些ALTER PARTITION操作，该分区表包含已被交换以使用外部表的叶子分区：

- Setting a subpartition template.
- Altering the partition properties.
- Creating a default partition.
- Setting a distribution policy.
- Setting or dropping a NOT NULL constraint of column.
- Adding or dropping constraints.
- Splitting an external partition.

不允许更改系统目录表的任何部分。

Examples

向列中添加列：

```
ALTER TABLE distributors ADD COLUMN address varchar(30);
```

重命名现有列：

```
ALTER TABLE distributors RENAME COLUMN address TO city;
```

重命名现有表：

```
ALTER TABLE distributors RENAME TO suppliers;
```

向列添加非空约束：

```
ALTER TABLE distributors ALTER COLUMN street SET NOT NULL;
```

重命名现有约束：

```
ALTER TABLE distributors RENAME CONSTRAINT zipchk TO zip_check;
```

向表及其所有子级添加检查约束：

```
ALTER TABLE distributors ADD CONSTRAINT zipchk CHECK  
(char_length(zipcode) = 5);
```

要将检查约束仅添加到表而不是其子表，请执行以下操作：

```
ALTER TABLE distributors ADD CONSTRAINT zipchk CHECK  
(char_length(zipcode) = 5) NO INHERIT;
```

(检查约束也不会被将来的子代继承。)

从表及其所有子级中删除检查约束：

```
ALTER TABLE distributors DROP CONSTRAINT zipchk;
```

仅从一个表中删除检查约束：

```
ALTER TABLE ONLY distributors DROP CONSTRAINT zipchk;
```

(对于任何继承distributors的子表，检查约束仍然存在。)

将表移动到不同的模式：

```
ALTER TABLE myschema.distributors SET SCHEMA yourschema;
```

将表的分发策略更改为已复制：

```
ALTER TABLE myschema.distributors SET DISTRIBUTED REPLICATED;
```

将新分区添加到分区表：

```
ALTER TABLE sales ADD PARTITION  
START (date '2017-02-01') INCLUSIVE  
END (date '2017-03-01') EXCLUSIVE;
```

向现有分区设计添加默认分区：

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

重命名分区：

```
ALTER TABLE sales RENAME PARTITION FOR ('2016-01-01') TO  
jan08;
```

删除范围序列中的第一个（最旧的）分区：

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

将表交换到用户的分区设计中：

```
ALTER TABLE sales EXCHANGE PARTITION FOR ('2016-01-01') WITH
TABLE jan08;
```

拆分默认分区(现有的默认分区名称other)为2017年1月添加新的每月分区：

```
ALTER TABLE sales SPLIT DEFAULT PARTITION
START ('2017-01-01') INCLUSIVE
END ('2017-02-01') EXCLUSIVE
INTO (PARTITION jan09, PARTITION other);
```

将每月分区分成两个分区，第一个分区包含日期1月1日至15日，第二个分区包含日期1月16日至31日：

```
ALTER TABLE sales SPLIT PARTITION FOR ('2016-01-01')
AT ('2016-01-16')
INTO (PARTITION jan081to15, PARTITION jan0816to31);
```

要重新创建主键约束，而在重建索引时不阻止更新：

```
CREATE UNIQUE INDEX CONCURRENTLY dist_id_temp_idx ON distributors
(dist_id);
ALTER TABLE distributors DROP CONSTRAINT distributors_pkey,
ADD CONSTRAINT distributors_pkey PRIMARY KEY USING INDEX
dist_id_temp_idx;
```

兼容性

ADD (不包含 USING INDEX), DROP, SET DEFAULT和SET DATA TYPE (不包含 USING) 符合SQL标准。其他形式是SQL标准的Greenplum数据库扩展。同样，在单个ALTER TABLE命令中指定多个操纵的功能也是一种扩展。

ALTER TABLE DROP COLUMN可用于删除表的唯一列，而保留零列表。这是SQL的扩展，不允许使用零列表。

另见

[CREATE TABLE, DROP TABLE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改表空间的定义。

概要

```
ALTER TABLESPACE name RENAME TO new_name
```

```
ALTER TABLESPACE name OWNER TO new_owner
```

```
ALTER TABLESPACE name SET ( tablespace_option = value [, ...] )
```

```
ALTER TABLESPACE name RESET ( tablespace_option [, ...] )
```

描述

`ALTER TABLESPACE`更改表空间的定义。

您必须拥有表空间才能使用`ALTER TABLESPACE`。要更改所有者，您还必须是新拥有角色的直接或间接成员。（请注意，超级用户会自动拥有这些特权。）

参数

name

一个现有表空间的名称。

new_name

新的表空间的名称。新的表空间名称不能以`pg_`或`gp_`开头。（这类名称保留用于系统表空间）。

new_owner

该表空间的新拥有者。

tablespace_parameter

要设置或重置的表空间参数。当前，唯一可用的参数是`seq_page_cost`和`random_page_cost`。为特定表空间设置这两个值将覆盖计划器通常对从该表空间中的表读取页面的开销的估计，这是由同名的配置参数确定的（请参见`seq-page-cost`，`random-page-cost`）。如果一个表空间位于比I/O子系统的其余

部分更快或更慢的磁盘上，则这可能很有用。

示例

重命名表空间 index_space 为 fast_raid:

```
ALTER TABLESPACE index_space RENAME TO fast_raid;
```

更改表空间index_space的所有者:

```
ALTER TABLESPACE index_space OWNER TO mary;
```

兼容性

在SQL标准中没有ALTER TABLESPACE语句。

另见

[CREATE TABLESPACE, DROP TABLESPACE](#)

Parent topic: [SQL Command Reference](#)

CONFIGURATION

Greenplum数据库® 6.0文档

更改文本搜索配置的定义。

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

概要

```

ALTER TEXT SEARCH CONFIGURATION name
    ALTER MAPPING FOR token_type [ , ... ] WITH
        dictionary_name [ , ... ]
ALTER TEXT SEARCH CONFIGURATION name
    ALTER MAPPING REPLACE old_dictionary WITH
        new_dictionary
ALTER TEXT SEARCH CONFIGURATION name
    ALTER MAPPING FOR token_type [ , ... ] REPLACE
        old_dictionary WITH new_dictionary
ALTER TEXT SEARCH CONFIGURATION name
    DROP MAPPING [ IF EXISTS ] FOR token_type [ , ... ]
ALTER TEXT SEARCH CONFIGURATION name RENAME TO new_name
ALTER TEXT SEARCH CONFIGURATION name OWNER TO new_owner
ALTER TEXT SEARCH CONFIGURATION name SET SCHEMA new_schema

```

Description

ALTER TEXT SEARCH CONFIGURATION更改文本搜索配置的定义。您可以修改其从令牌类型到字典的映射，或者更改配置的名称或所有者。

您必须是配置的所有者才能使用ALTER TEXT SEARCH CONFIGURATION。

参数

name

现有文本搜索配置的名称（可选，模式限定）。

token_type

配置的解析器发出的令牌类型的名称。

dictionary_name

用于指定令牌类型的文本搜索字典的名称。如果列出了多个词



典，则将以指定顺序对其进行查询。

old_dictionary

映射中要替换的文本搜索字典的名称。

new_dictionary

用来替换*old_dictionary*的文本搜索字典的名称

new_name

文本搜索配置的新名称。

new_owner

文本搜索配置的新所有者。

new_schema

文本搜索配置的新架构。

ADD MAPPING FOR表单会安装一个字典列表，供您查询指定的令牌类型；如果已经有任何令牌类型的映射，则为错误。 ADD MAPPING FOR表单的功能相同，但是首先删除那些令牌类型的任何现有映射。

ADD MAPPING FOR表单用*new_dictionary*替换*old_dictionary*的任何地方。当出现FOR时，仅对指定的令牌类型执行此操作，否则不进行配置的所有映射。 DROP MAPPING表单删除指定令牌类型的所有字典，从而使这些类型的令牌被文本搜索配置忽略。如果没有令牌类型的映射，这是一个错误，除非出现IF EXISTS。

示例

以下示例在my_config中使用english的任何地方，用swedish词典替换english词典。

```
ALTER TEXT SEARCH CONFIGURATION my_config  
ALTER MAPPING REPLACE english WITH swedish;
```

兼容性

在SQL标准中没有ALTER TEXT SEARCH CONFIGURATION语句。

另见

[CREATE TEXT SEARCH CONFIGURATION, DROP TEXT SEARCH CONFIGURATION](#)

Parent topic: [SQL Command Reference](#)

DICTIONARY

Greenplum数据库® 6.0文档

更改文本搜索词典的定义。

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLE

ALTER FOREIGN
DATA WRAPPER

ALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

概要

```
ALTER TEXT SEARCH DICTIONARY name (
    option [ = value ] [, ... ]
)
ALTER TEXT SEARCH DICTIONARY name RENAME TO new_name
ALTER TEXT SEARCH DICTIONARY name OWNER TO new_owner
ALTER TEXT SEARCH DICTIONARY name SET SCHEMA new_schema
```

Description

ALTER TEXT SEARCH DICTIONARY更改文本搜索词典的定义。您可以更改词典的特定于模板的选项，或更改词典的名称或所有者。

您必须是字典的所有者，才能使用ALTER TEXT SEARCH DICTIONARY。

参数

name

现有文本搜索字典的名称（可选，用模式限定）。

option

要为此字典设置的模板特定选项的名称。

value

用于模板特定选项的新值。如果省略了等号和值，则将从字典中删除该选项的所有先前设置，从而允许使用默认值。

new_name

文本搜索字典的新名称。

new_owner

文本搜索字典的新所有者。

new_schema

文本搜索字典的新模式。



模板特定的选项可以按任何顺序出现。

示例

以下示例命令更改基于Snowball的字典的停用词列表。 其他参数保持不变。

```
ALTER TEXT SEARCH DICTIONARY my_dict ( StopWords =  
newrussian );
```

以下示例命令将language选项更改为dutch，并完全删除了停用词选项。

```
ALTER TEXT SEARCH DICTIONARY my_dict ( language =  
dutch, StopWords );
```

下面的示例命令“更新”字典的定义，而无需实际更改任何内容。

```
ALTER TEXT SEARCH DICTIONARY my_dict ( dummy );
```

(这样做的原因是，如果没有这样的选项，则选项删除代码不会抱怨。) 此技巧在更改字典的配置文件时很有用：ALTER将强制现有的数据库会话重新读取配置文件，否则，如果他们早些阅读它们，他们将永远做不到。

兼容性

SQL标准中没有ALTER TEXT SEARCH DICTIONARY语句。

另见

[CREATE TEXT SEARCH DICTIONARY](#), [DROP TEXT SEARCH DICTIONARY](#)

Parent topic: [SQL Command Reference](#)

PARSER

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)

[ABORT](#)

[ALTER AGGREGATE](#)

[ALTER COLLATION](#)

[ALTER CONVERSION](#)

[ALTER DATABASE](#)

[ALTER DEFAULT PRIVILEGES](#)

[ALTER DOMAIN](#)

[ALTER EXTENSION](#)

[ALTER EXTERNAL TABLE](#)

[ALTER FOREIGN DATA WRAPPER](#)

[ALTER FOREIGN TABLE](#)

[ALTER FUNCTION](#)

[ALTER GROUP](#)

[ALTER INDEX](#)

[ALTER LANGUAGE](#)

[ALTER OPERATOR](#)

描述

更改文本搜索解析器的定义。

概要

```
ALTER TEXT SEARCH PARSER name RENAME TO new_name
ALTER TEXT SEARCH PARSER name SET SCHEMA new_schema
```

描述

ALTER TEXT SEARCH PARSER更改文本搜索解析器的定义。当前，唯一支持的功能是更改解析器的名称。

您必须是超级用户才能使用ALTER TEXT SEARCH PARSER。

参数

name

现有文本搜索解析器的名称（可选，由模式限定）。

new_name

文本搜索解析器的新名称。

new_schema

文本搜索解析器的新架构。

兼容性

SQL标准中没有ALTER TEXT SEARCH PARSER语句。



另见

[CREATE TEXT SEARCH PARSER](#), [DROP TEXT SEARCH PARSER](#)

Parent topic: [SQL Command Reference](#)

TEMPLATE

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLE

ALTER FOREIGN
DATA WRAPPER

ALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

描述

更改文本搜索模板的定义。

概要

```
ALTER TEXT SEARCH TEMPLATE name RENAME TO new_name
ALTER TEXT SEARCH TEMPLATE name SET SCHEMA new_schema
```

描述

ALTER TEXT SEARCH TEMPLATE更改文本搜索解析器的定义。当前，唯一受支持的功能是更改解析器的名称。

您必须是超级用户才能使用ALTER TEXT SEARCH TEMPLATE.

参数

name

现有文本搜索模板的名称（可选，由模式限定）。

new_name

文本搜索模板的新名称。

new_schema

文本搜索模板的新架构。

兼容性

SQL标准中没有ALTER TEXT SEARCH TEMPLATE语句。



另见

[CREATE TEXT SEARCH TEMPLATE](#), [DROP TEXT SEARCH TEMPLATE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改一个数据类型的定义。

概要

```
ALTER TYPE name action [, ... ]
ALTER TYPE name OWNER TO new_owner
ALTER TYPE name RENAME ATTRIBUTE attribute_name TO
new_attribute_name [ CASCADE | RESTRICT ]
ALTER TYPE name RENAME TO new_name
ALTER TYPE name SET SCHEMA new_schema
ALTER TYPE name ADD VALUE [ IF NOT EXISTS ] new_enum_value [ {
BEFORE | AFTER } existing_enum_value ]
ALTER TYPE name SET DEFAULT ENCODING ( storage_directive )
```

其中 *action* 是下列选项之一：

```
ADD ATTRIBUTE attribute_name data_type [ COLLATE collation] [
CASCADE | RESTRICT ]
DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRICT ]
ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [
COLLATE collation] [ CASCADE | RESTRICT ]
```

其中 *storage_directive* 是：

```
COMPRESSTYPE={ZLIB | ZSTD | QUICKLZ | RLE_TYPE | NONE}
COMPRESSLEVEL={0-19}
BLOCKSIZE={8192-2097152}
```

描述

ALTER TYPE 改变现有类型的定义。有几个子形式：

- **ADD ATTRIBUTE** — 使用与CREATE TYPE相同的语法向复合类型添加新属性。
- **DROP ATTRIBUTE [IF EXISTS]** — 从复合类型中删除属性。如果指定了IF EXISTS且该属性不存在，则不会引发任何错误。在这种情况下，将发出通知。
- **SET DATA TYPE** — 改变复合类型的属性的类型。
- **OWNER** — 改变类型的所有者。



- **RENAME** — 更改类型的名称或复合类型的单个属性的名称。
- **SET SCHEMA** — 将类型移动到另一个架构。
- **ADD VALUE [IF NOT EXISTS] [BEFORE | AFTER]** — 将新值添加到枚举类型。可以将新值在枚举顺序中的位置指定为现有值之一BEFORE或AFTER。否则，新项目将添加到值列表的末尾。
如果指定了IF NOT EXISTS，则该类型已经包含新值就不是错误；发出通知，但不采取其他措施。否则，如果新值已经存在，将发生错误。
- **CASCADE** — 自动将操作传播到要更改的类型的类型表及其后代。
- **RESTRICT** — 如果要更改的类型是类型表的类型，则拒绝该操作。这是默认值。

可以将ADD ATTRIBUTE, DROP ATTRIBUTE和ALTER ATTRIBUTE操作组合为多个更改列表，以并行应用。例如，可以在单个命令中添加多个属性和/或更改多个属性的类型。

您可以更改名称，所有者和类型的架构。您还可以添加或更新标量类型的存储选项。

Note: Greenplum数据库不支持为行或复合类型添加存储选项。

您必须拥有该类型才能使用ALTER TYPE。要更改类型的架构，您还必须对新架构具有CREATE特权。要更改所有者，您还必须是新拥有角色的直接或间接成员，并且该角色必须对类型的架构具有CREATE特权。（这些限制迫使更改所有者不能执行删除和重新创建类型的任何操作。但是，超级用户可以更改任何类型的所有权。）要添加属性或更改属性类型，还必须具有USAGE数据类型的特权。

ALTER TYPE ... ADD VALUE（向枚举类型添加新值的形式）不能在事务块内执行。

涉及增加的枚举值的比较有时会比仅涉及枚举类型的原始成员的比较慢。通常只有在使用BEFORE或AFTER设置新值的排序位置（而不是列表的末尾）时，才会发生这种情况。但是，有时即使将新值添加到末尾也会发生（如果自从最初创建枚举类型以来，OID计数器“环绕”，就会发生这种情况）。增长速度通常很小。但是，如果重要的话，可以通过删除并重新创建枚举类型，或者通过转储并重新加载数据库来获得最佳性能。

参数

name

要更改的现有类型的名称（可选的模式限定）。

new_name

类型的新名称。

new_owner

该类型的新所有者的用户名。

new_schema

类型的新架构。

attribute_name

要添加、更改或删除的属性的名称。

new_attribute_name

要重命名的属性的新名称。

data_type

要添加的属性的数据类型，或要更改的属性的新类型。

new_enum_value

要添加的属性的数据类型，或要更改的属性的新类型。

existing_enum_value

应该在枚举类型的排序顺序之前或之后立即添加新值的现有枚举值。像所有枚举文字一样，也需要用引号引起。

storage_directive

在表列定义中指定时，标识该类型的默认存储选项。选项包括COMPRESSTYPE, COMPRESSLEVEL和BLOCKSIZE。

COMPRESSTYPE — 设置为ZLIB（默认设置），ZSTD，RLE_TYPE或QUICKLZ¹以指定使用的压缩类型。

Note: ¹QuickLZ压缩仅在商业版本的Pivotal Greenplum数据库中可用。

COMPRESSLEVEL — 对于Zstd压缩，将其设置为1（最快压缩）到19（最高压缩率）之间的整数值。对于 zlib压缩，有效范围是1到9。QuickLZ压缩级别只能设置为1。对于RLE_TYPE，压缩级别可以设置为从1（最快压缩）到4（最高压缩率）的整数值。缺省压缩级别为1。

BLOCKSIZE — 设置为列中每个块的大小（以字节为单位）。BLOCKSIZE必须介于8192和2097152字节之间，并且是8192的倍数。默认块大小为32768。

Note: 在表或列级别定义的*storage_directives*会覆盖为类型定义的默认存储选项。

示例

要重命名名为electronic_mail的数据类型：

```
ALTER TYPE electronic_mail RENAME TO email;
```

更改用户自定义类型email的所有者为joe：

```
ALTER TYPE email OWNER TO joe;
```

更改用户自定义类型email的模式为customers：

```
ALTER TYPE email SET SCHEMA customers;
```

设置或更改名为int33的用户定义类型的压缩类型和压缩级别：

```
ALTER TYPE int33 SET DEFAULT ENCODING (compressstype=zlib,  
compresslevel=7);
```

要将新属性添加到类型：

```
ALTER TYPE compfoo ADD ATTRIBUTE f3 int;
```

要将新值添加到特定排序位置的枚举类型：

```
ALTER TYPE colors ADD VALUE 'orange' AFTER 'red';
```

兼容性

添加和删除属性的变体是SQL标准的一部分。 其他变体是Greenplum数据库扩展。

另见

[CREATE TYPE, DROP TYPE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0 文档

 参考指南 SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改数据库用户（角色）的定义。

概要

ALTER USER *name* RENAME TO *newname*ALTER USER *name* SET *config_parameter* {TO | =} {*value* | DEFAULT}ALTER USER *name* RESET *config_parameter*ALTER USER *name* RESOURCE QUEUE {*queue_name* | NONE}ALTER USER *name* RESOURCE GROUP {*group_name* | NONE}ALTER USER *name* [[WITH] *option* [...]]其中 *option* 可以是：

```

SUPERUSER | NOSUPERUSER
| CREATEDB | NOCREATEDB
| CREATEROLE | NOCREATEROLE
| CREATEUSER | NOCREATEUSER
| CREATEEXTTABLE | NOCREATEEXTTABLE
[ ( attribute='value'[, ...] ) ]
      where attributes and value are:
      type='readable' | 'writable'
      protocol='gpfdist' | 'http'
| INHERIT | NOINHERIT
| LOGIN | NOLOGIN
| REPLICATION | NOREPLICATION
| CONNECTION LIMIT connlimit
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
| VALID UNTIL 'timestamp'
| [ DENY deny_point ]
| [ DENY BETWEEN deny_point AND deny_point ]
| [ DROP DENY FOR deny_point ]

```

描述

ALTER USER 是 ALTER ROLE 的别名。参阅 [ALTER ROLE](#) 以获取更多信息。



兼容性

ALTER USER语句是一个Greenplum数据库扩展。 SQL标准中使用用户的定义来实现。

See Also

[ALTER ROLE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改外部服务器的用户映射的定义。

概要

```
ALTER USER MAPPING FOR { username | USER | CURRENT_USER |
PUBLIC }
    SERVER servername
    OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ...]
) )
```

Description

`ALTER USER MAPPING`更改外部服务器的用户映射的定义。

外部服务器的所有者可以为任何用户更改该服务器的用户映射。同样，在服务器上被授予`USAGE`特权的用户可以更改其自己的用户名的用户映射。

参数

username

映射的用户名。`CURRENT_USER`和`USER`与当前用户的名称匹配。`PUBLIC`用于匹配系统中所有当前和将来的用户名。

servername

用户映射的服务器名称。

`OPTIONS ([ADD | SET | DROP] option ['value'] [, ...])`

更改用户映射的选项。新选项将覆盖所有先前指定的选项。`ADD`, `SET`和`DROP`指定要执行的操作。如果未明确指定任何操作，则默认操作为`ADD`。选项名称必须唯一。Greenplum数据库使用服务器的外部数据包装器来验证名称和值。

示例

更改用户映射bob，服务器foo的密码：

```
ALTER USER MAPPING FOR bob SERVER foo OPTIONS (SET password  
'public');
```

兼容性

ALTER USER MAPPING符合ISO / IEC 9075-9 (SQL / MED)。存在一个微妙的语法问题：标准省略了FOR关键字。由于CREATE USER MAPPING和DROP USER MAPPING都在相似的位置使用FOR，因此Greenplum Database在此处出于一致性和互操作性而与标准有所不同。

另见

[CREATE USER MAPPING, DROP USER MAPPING](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT

PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改一个视图的定义。

概要

```
ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name
SET DEFAULT expression
```

```
ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name
DROP DEFAULT
```

```
ALTER VIEW [ IF EXISTS ] name OWNER TO new_owner
```

```
ALTER VIEW [ IF EXISTS ] name RENAME TO new_name
```

```
ALTER VIEW [ IF EXISTS ] name SET SCHEMA new_schema
```

```
ALTER VIEW [ IF EXISTS ] name SET ( view_option_name [= view_option_value] [, ...] )
```

```
ALTER VIEW [ IF EXISTS ] name RESET ( view_option_name [, ...] )
```

描述

ALTER VIEW更改视图的各种辅助属性。 (如果要修改视图的定义查询, 请使用CREATE OR REPLACE VIEW。)

要执行此命令, 您必须是视图的所有者。要更改视图的架构, 您还必须对新架构具有CREATE特权。要更改所有者, 您还必须是新拥有角色的直接或间接成员, 并且该角色必须在视图的架构上具有CREATE特权。这些限制迫使更改所有者不能执行删除和重新创建视图所无法做的任何事情。但是, 超级用户可以更改任何视图的所有权。

参数

name

现有视图的名称 (可选, 由模式限定)。

IF EXISTS

如果该视图不存在, 不会抛出错误, 而是发出通知。



SET/DROP DEFAULT

这些表格设置或删除列的默认值。在为视图应用任何规则或触发器之前，将视图列的默认值替换为目标为视图的任何INSERT或UPDATE命令。因此，视图的默认值将优先于基础关系中的任何默认值。

new_owner

视图的新所有者。

new_name

视图的新名称。

new_schema

视图的新架构。

```
SET ( view_option_name [= view_option_value] [ , ... ] )
```

```
RESET ( view_option_name [ , ... ] )
```

设置或重置视图选项。当前支持的选项是：

check_option (string)

更改视图的检查选项。该值必须是 local 或 cascaded。

security_barrier (boolean)

更改视图的安全屏障属性。该值必须是布尔值，例如 true 或 false。

注意

由于历史原因，ALTER TABLE 也可以与视图一起使用；但是，视图允许的ALTER TABLE 的唯一变体与上面显示的语句等效。

将视图 myview 重命名为 newview： Rename the view myview to newview：

```
ALTER VIEW myview RENAME TO newview;
```

示例

要将视图 foo 重命名为 bar：

```
ALTER VIEW foo RENAME TO bar;
```

要将默认列值附加到可更新视图：

```
CREATE TABLE base_table (id int, ts timestamptz);
CREATE VIEW a_view AS SELECT * FROM base_table;
ALTER VIEW a_view ALTER COLUMN ts SET DEFAULT now();
INSERT INTO base_table(id) VALUES(1); -- ts will receive a
NULL
INSERT INTO a_view(id) VALUES(2); -- ts will receive the
current time
```

兼容性

ALTER VIEW是SQL标准的Greenplum数据库扩展。

另见

[CREATE VIEW](#), [DROP VIEW](#) in the *Greenplum Database Utility Guide*

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

收集有关一个数据库的统计信息。

概要

```
ANALYZE [VERBOSE] [table [ (column [, ...] ) ]]
```

```
ANALYZE [VERBOSE] {root_partition|leaf_partition} [ (column [, ...] ) ]
```

```
ANALYZE [VERBOSE] ROOTPARTITION {ALL | root_partition [ (column [, ...] ) ]}
```

描述

ANALYZE收集有关数据库中表内容的统计信息，并将结果存储在系统表`pg_statistic`中。随后，Greenplum数据库使用这些统计信息来帮助确定最有效的查询执行计划。有关收集的表统计信息的信息，请参阅[Notes](#)。

如果不使用任何参数，则ANALYZE会收集当前数据库中每个表的统计信息。您可以指定表名称以收集单个表的统计信息。您可以指定一组列名，在这种情况下，仅收集这些列的统计信息。

ANALYZE不会收集外部表的统计信息。

对于分区表，ANALYZE在叶子分区上收集其他统计信息，即HyperLogLog (HLL) 统计信息。HLL统计信息用于得出针对分区表的查询的不同值 (NDV) 数量。

- 当汇总多个叶子分区的NDV估计值时，HLL统计信息比标准表统计信息生成更准确的NDV估计值。
- 更新HLL统计信息时，仅在已更改的叶子分区上才需要ANALYZE操作。例如，如果叶子子分区数据已更改，或者叶子子分区已与另一个表交换，则ANALYZE是必需的。有关更新分区表统计信息的更多信息，请参见[Notes](#)。

Important: 如果要在启用GPORCA的分区表上执行查询（），则必须使用ANALYZE或ANALYZE ROOTPARTITION命令在分区表的根分区上收集统计信息。有关收集分区表上的统计信息以及何时需要ROOTPARTITION关键字的信息，请参阅[Notes](#)。有

关于GPORCA的信息，请参阅中的[GPORCA概述](#)。

Note: 您还可以使用Greenplum数据库实用程序`analyzedb`更新表统计信息。`analyzedb`可以同时更新多个表的统计信息。该实用程序还可以检查表统计信息并仅在统计信息不是当前统计信息或不存在时更新统计信息。有关该实用程序的信息，请参阅[Greenplum Database Utility Guide](#)。

参数

{ *root_partition* | *leaf_partition* } [(*column* [, ...])]

收集分区表的统计信息，包括HLL统计信息。HLL统计信息仅在叶子分区上收集。

`ANALYZE root_partition`, 收集所有叶子分区和根分区的统计信息。

`ANALYZE leaf_partition`, 收集有关叶子分区的统计信息。

默认情况下，如果指定叶子分区，并且所有其他叶子分区都具有统计信息，则`ANALYZE`更新根分区统计信息。如果不是所有叶子子分区都具有统计信息，则`ANALYZE`记录有关没有统计信息的叶子子分区的信息。有关何时收集根分区统计信息的信息，请参阅[Notes](#)。

ROOTPARTITION [ALL]

仅基于分区表中的数据收集分区表的根分区上的统计信息。如果可能，`ANALYZE`使用叶子分区统计信息生成根分区统计信息。否则，`ANALYZE`通过对叶子分区数据进行采样来收集统计信息。未在叶子分区上收集统计信息，仅对数据进行采样。不会收集HLL统计信息。

有关何时需要`ROOTPARTITION`关键字的信息，请参阅[Notes](#)。

指定`ROOTPARTITION`时，必须指定`ALL`或分区表的名称。

如果将`ROOTPARTITION`指定为`ALL`，则Greenplum Database会收集数据库中所有分区表的根分区的统计信息。如果数据库中没有分区表，则会返回一条消息，指出没有分区表。对于不是分区表的表，不会收集统计信息。

如果使用`ROOTPARTITION`指定表名，并且该表不是分区表，则不会为该表收集任何统计信息，并且会返回警告消息。

`ROOTPARTITION`子句不适用于`VACUUM ANALYZE`。`VACUUM ANALYZE ROOTPARTITION`命令返回错误。

运行`ANALYZE ROOTPARTITION`的时间类似于分析具有相同数据的非分区表的时间，因为`ANALYZE ROOTPARTITION`仅采样叶子分区数据。

对于分区表 `sales_curr_yr`，此示例命令仅在分区表的根分区上收集统计信息。`ANALYZE ROOTPARTITION`

```
sales_curr_yr; ;
```

此示例ANALYZE命令收集有关数据库中所有分区表的根分区的统计信息。

```
ANALYZE ROOTPARTITION ALL;
```

VERBOSE

启用显示进度消息。 启用显示进度消息。 指定时，ANALYZE发出此信息

- 正在处理的表。
- 执行该查询以生成示例表。
- 要为其计算统计信息的列。
- 发出以收集单个列的不同统计信息的查询。
- 收集的统计信息。

table

要分析的特定表的名称（可能是模式限定的）。 如果省略，则分析当前数据库中的所有常规表（而不是外部表）。

column

要分析的特定列的名称。 默认为所有列。

注意

仅在明确选择外表时才进行分析。 并非所有外部数据包装器都支持ANALYZE。 如果表的包装器不支持ANALYZE，则该命令将显示警告并且不执行任何操作。

最好定期或在对表内容进行重大更改之后立即运行ANALYZE。 准确的统计信息有助于Greenplum数据库选择最合适的选择计划，从而提高查询处理的速度。 只读数据库的常见策略是在一天的低使用时间内每天运行一次[VACUUM](#)和ANALYZE。（如果有大量更新活动，这是不够的。）您可以使用gp_toolkit模式中的gp_stats_missing视图来检查缺少统计信息的表：

```
SELECT * from gp_toolkit.gp_stats_missing;
```

ANALYZE要求对目标表进行SHARE UPDATE EXCLUSIVE锁定。 此锁与以下锁冲突：SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE。

如果您在不包含数据的表上运行ANALYZE，则不会为该表收集统计信息。 例如，如果您对具有统计信息的表执行TRUNCATE操作，然后对

该表运行ANALYZE，则统计信息不会更改。

对于分区表，如果分区表具有大量已分析的分区，而只有几个叶子分区具有分区，则指定要分析的表部分，根分区或子分区（叶子分区表）可能会有用改变了。

Note: 当您使用CREATE TABLE命令创建分区表时，Greenplum数据库将创建您指定的表（根分区或父表），并根据您指定的分区层次结构（子表）创建表层次结构。

- 在根分区表上运行ANALYZE时，将收集所有叶子分区的统计信息。叶子子分区是Greenplum数据库创建供子表使用的子表层次结构中的最低级表。
- 在叶子分区上运行ANALYZE时，仅收集该叶子分区和根分区的统计信息。如果叶子分区中的数据已更改（例如，您对叶子子分区数据进行了重大更新或交换了叶子子分区），则可以在叶子子分区上运行ANALYZE来收集表统计信息。默认情况下，如果所有其他叶子分区都具有统计信息，则该命令将更新根分区统计信息。
例如，如果您在具有大量分区的分区表上收集统计信息，然后仅在几个叶子分区中更新数据，则可以仅在那些分区上运行ANALYZE来更新分区的统计信息和根分区的统计信息。
- 在不是叶子分区的子表上运行ANALYZE时，不会收集统计信息。
例如，您可以创建一个分区表，其中包含2006年至2016年的分区以及每年每个月的子分区。如果您在2013年的子表上运行ANALYZE，则不会收集任何统计信息。如果您在2013年3月在叶子分区上运行ANALYZE，则仅收集该叶子分区的统计信息。

对于包含已被交换以使用外部表的叶子分区的分区表，ANALYZE不会收集外部表分区的统计信息：

- 如果在外部表分区上运行ANALYZE，则不会分析该分区。
- 如果在根分区上运行ANALYZE或ANALYZE ROOTPARTITION，则不对外部表分区进行采样，并且根表统计信息不包括外部表分区。
- 如果指定了VERBOSE子句，则会显示一条参考消息：skipping external table。

Greenplum数据库服务器配置参数

`optimizer_analyze_root_partition`影响何时在分区表的根分区上收集统计信息。如果该参数为on（默认值），则在运行ANALYZE时，不需要ROOTPARTITION关键字来收集根分区上的统计信息。在根分区上运行ANALYZE或在分区表的子叶分区上运行ANALYZE且其他子叶分区具有统计信息时，将收集根分区统计信息。如果该参数是off，则必须运行ANALYZE ROOTPARTITION来收集根分区统计信息。

ANALYZE收集的统计信息通常包括每列中一些最常用值的列表以及显

示每列中近似数据分布的直方图。如果ANALYZE认为它们不重要（例如，在唯一键列中没有公共值），或者列数据类型不支持适当的运算符，则可以忽略其中一个或两个。

对于大型表，ANALYZE会从表内容中随机抽取一个样本，而不是检查每一行。这样就可以在很短的时间内分析非常大的表。但是请注意，统计信息仅是近似的，并且每次运行ANALYZE都会略有变化，即使实际的表内容没有变化。这可能会导致EXPLAIN所显示的计划者估算成本发生细微变化。在极少数情况下，这种不确定性将导致查询优化器在ANALYZE运行之间选择不同的查询计划。为了避免这种情况，请通过调整*default_statistics_target*配置参数来提高ANALYZE收集的统计信息的数量，或者通过使用ALTER TABLE ... ALTER COLUMN ... SET (n_distinct ...)（请参阅ALTER TABLE）。目标值设置最常用值列表中的最大条目数和直方图中的最大bin数。默认目标值是100，但是可以向上或向下调整该值以权衡规划器估计的准确性与ANALYZE所花费的时间以及pg_statistic中占用的空间量。特别是，将统计目标设置为零会禁用该列的统计收集。对于从未用作查询的WHERE, GROUP BY或ORDER BY子句一部分的列，执行此操作可能很有用，因为计划器将不会使用此类列的统计信息。

要分析的列中最大的统计信息目标确定为准备统计信息而采样的表行数。增加目标会导致进行ANALYZE所需的时间和空间成比例增加。

ANALYZE估计的值之一是出现在每列中的不同值的数量。因为仅检查了行的子集，所以即使使用最大可能的统计目标，此估计有时也可能非常不准确。如果此错误导致查询计划不正确，则可以手动确定更准确的值，然后与ALTER TABLE ... ALTER COLUMN ... SET STATISTICS DISTINCT一起安装（请参阅ALTER TABLE）。

当Greenplum数据库执行ANALYZE操作以收集表的统计信息并检测到所有采样的表数据页均为空（不包含有效数据）时，Greenplum数据库将显示一条消息，指出应该执行VACUUM FULL操作。如果采样页为空，则表统计信息将不准确。对表进行大量更改（例如删除大量行）后，页面将变为空。VACUUM FULL操作可删除空白页，并允许ANALYZE操作收集准确的统计信息。

如果该表没有统计信息，则服务器配置参数*gp_enable_relsize_collection*将控制Postgres查询优化器使用默认统计信息文件还是使用pg_relation_size函数估计表的大小。默认情况下，如果统计信息不可用，Postgres优化器将使用默认的统计信息文件来估计行数。

示例

收集表mytable的统计信息：

```
ANALYZE mytable;
```

兼容性

SQL标准中没有ANALYZE语句。

另见

[ALTER TABLE](#), [EXPLAIN](#), [VACUUM](#), `analyzedb` utility in the *Greenplum Database Utility Guide*.

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

启动事务块。

概要

```
BEGIN [WORK | TRANSACTION] [transaction_mode]
```

其中*transaction_mode*是：

```
ISOLATION LEVEL {READ UNCOMMITTED | READ COMMITTED |
REPEATABLE READ | SERIALIZABLE}
READ WRITE | READ ONLY
[ NOT ] DEFERRABLE
```

描述

BEGIN启动事务块，即BEGIN命令之后的所有语句将在单个事务中执行，直到给出显式的COMMIT或ROLLBACK。默认情况下（没有BEGIN），Greenplum数据库以自动提交模式执行事务，即，每个语句在其自己的事务中执行，并且在语句末尾隐式执行提交（如果执行成功，否则回滚）。

在事务块中，语句的执行速度更快，因为事务启动/提交需要大量的CPU和磁盘活动。在事务中执行多个语句对于确保进行一些相关更改时的一致性也很有用：其他会话将无法看到中间状态，其中不是所有相关更新都完成。

如果指定了隔离级别，读/写模式或可延迟模式，则新事务具有那些特征，就像执行SET TRANSACTION一样。

参数

WORK

TRANSACTION

可选关键字。它们没有作用。

SERIALIZABLE

READ COMMITTED



READ UNCOMMITTED

SQL标准定义了四个事务隔离级别： READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ和SERIALIZABLE。

READ UNCOMMITTED允许事务查看未提交的并发事务所做的更改。在Greenplum数据库中这是不可能的，因此READ UNCOMMITTED与READ COMMITTED相同。

READ COMMITTED是Greenplum数据库中的默认隔离级别，可确保语句只能看到在开始之前提交的行。如果在第一次执行该语句后又提交了另一个并发事务，则在一个事务中执行两次的同一条语句可能会产生不同的结果。

REPEATABLE READ隔离级别确保事务只能看到在事务开始之前提交的行。REPEATABLE READ是Greenplum数据库支持的最严格的事务隔离级别。由于序列化失败，必须准备使用REPEATABLE READ隔离级别的应用程序以重试事务。

SERIALIZABLE事务隔离级别确保执行多个并发事务与串行执行相同的事务产生的效果相同。如果指定SERIALIZABLE，则Greenplum数据库将退回到REPEATABLE READ。

指定DEFERRABLE在Greenplum数据库中无效，但是支持语法以与PostgreSQL兼容。仅当事务为READ ONLY且SERIALIZABLE，并且Greenplum数据库不支持SERIALIZABLE事务时，才可以推迟该事务。

注解

[START TRANSACTION](#)具有与BEGIN相同的功能。

使用[COMMIT](#)或[ROLLBACK](#)终止事务块。

如果已经在事务块中，则发出BEGIN会引起警告消息。事务状态不受影响。要将事务嵌套在事务块中，请使用保存点（请参见SAVEPOINT）。

示例

要开始事务块：

```
BEGIN;
```

要以可重复读隔离级别开始事务块：

```
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

兼容性

BEGIN是Greenplum数据库语言扩展。 它等效于SQL标准命令[START TRANSACTION](#)。

DEFERRABLE *transaction_mode*是Greenplum数据库语言的扩展。

附带地， BEGIN关键字在嵌入式SQL中用于不同的目的。 建议您在移植数据库应用程序时注意事务语义。

另见

[COMMIT](#), [ROLLBACK](#), [START TRANSACTION](#), [SAVEPOINT](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)[ABORT](#)[ALTER AGGREGATE](#)[ALTER COLLATION](#)[ALTER CONVERSION](#)[ALTER DATABASE](#)[ALTER DEFAULT PRIVILEGES](#)[ALTER DOMAIN](#)[ALTER EXTENSION](#)[ALTER EXTERNAL TABLE](#)[ALTER FOREIGN DATA WRAPPER](#)[ALTER FOREIGN TABLE](#)[ALTER FUNCTION](#)[ALTER GROUP](#)[ALTER INDEX](#)[ALTER LANGUAGE](#)[ALTER OPERATOR](#)

强制执行事务日志检查点。

概要

CHECKPOINT

描述

检查点是事务日志序列中的一点，所有数据文件都已在该点更新，以反映日志中的信息。所有数据文件将刷新到磁盘。

发出该命令时，CHECKPOINT命令将强制立即执行检查点，而无需等待系统安排的常规检查点。CHECKPOINT不适合在常规操作中使用。

如果在恢复期间执行，则CHECKPOINT命令将强制重新启动点，而不是写入新的检查点。

只有超级用户可以调用CHECKPOINT。

兼容性

CHECKPOINT命令是Greenplum数据库扩展。

Parent topic: [SQL Command Reference](#)



Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)[ABORT](#)[ALTER AGGREGATE](#)[ALTER COLLATION](#)[ALTER CONVERSION](#)[ALTER DATABASE](#)[ALTER DEFAULT PRIVILEGES](#)[ALTER DOMAIN](#)[ALTER EXTENSION](#)[ALTER EXTERNAL TABLE](#)[ALTER FOREIGN DATA WRAPPER](#)[ALTER FOREIGN TABLE](#)[ALTER FUNCTION](#)[ALTER GROUP](#)[ALTER INDEX](#)[ALTER LANGUAGE](#)[ALTER OPERATOR](#)

关闭游标。

概要

`CLOSE cursor_name`

描述

`CLOSE`释放与打开的游标关联的资源。关闭游标后，不允许对其进行后续操作。不再需要游标时，应将其关闭。

当事务通过`COMMIT`或`ROLLBACK`终止时，每个非持有的打开游标都会隐式关闭。如果创建游标的事务通过`ROLLBACK`中止，则该游标将隐式关闭。如果创建事务成功提交，则持有的游标将保持打开状态，直到执行了显式`CLOSE`或客户端断开连接为止。

参数

cursor_name

要关闭的打开的游标的名称。

注解

Greenplum数据库没有显式的`OPEN`游标语句。当游标被声明时，它被认为是打开的。使用`DECLARE`语句声明（并打开）游标。

您可以通过查询`pg_cursors`系统视图来查看所有可用的游标。

如果在保存点之后关闭游标，该保存点随后会回滚，则`CLOSE`不会回滚；即光标保持关闭状态。



示例

关闭游标portala:

```
CLOSE portala;
```

兼容性

CLOSE完全符合SQL标准。

另见

[DECLARE](#) , [FETCH](#) , [MOVE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

根据索引对磁盘上的堆存储表进行物理重新排序。在Greenplum数据库中不建议执行此操作。

概要

```
CLUSTER indexname ON tablename
CLUSTER [VERBOSE] tablename
CLUSTER [VERBOSE]
```

描述

CLUSTER根据索引对堆存储表进行排序。追加优化存储表不支持CLUSTER。对索引进行聚簇意味着根据索引信息在磁盘上对记录进行物理排序。如果您需要的记录随机分布在磁盘上，则数据库必须跨磁盘搜索以获取请求的记录。如果将这些记录更紧密地存储在一起，则从磁盘进行的获取将更加有序。聚集索引的一个很好的例子是在date列上，该列按日期顺序对数据进行排序。对特定日期范围的查询将导致从磁盘中有序地进行访问，从而利用了更快的顺序访问。

聚簇是一项一次性操作：表在随后进行更新时，更改不会被聚簇。也就是说，没有尝试根据新的或更新的行的索引顺序来存储它们。如果您愿意，可以通过再次发出命令来定期重新聚簇。将表的FILLFACTOR存储参数设置为小于100%可以有助于在更新期间保留聚簇顺序，因为如果那里有足够的空间，则更新的行将保留在同一页上。

当使用此命令将表聚簇时，Greenplum数据库会记住该表在哪个索引上聚簇。形式CLUSTER *tablename*将表重新聚集在与聚簇相同的索引上。您可以使用`ALTER TABLE`的CLUSTER或SET WITHOUT CLUSTER形式来设置索引以用于将来的聚簇操作，或清除任何先前的设置。不带任何参数的CLUSTER会重新聚簇调用用户所拥有的当前数据库中所有先前聚簇的表，或者重新聚簇所有表（如果由超级用户调用）。这种形式的CLUSTER不能在事务块内执行。

对表进行聚簇时，将在其上获取ACCESS EXCLUSIVE锁。这样可以防止在表CLUSTER完成之前对表执行任何其他数据库操作（读取和写入）。

参数

indexname

索引名称。

VERBOSE

当每个表聚簇时，打印进度报告。

tablename

表的名称（可以由模式指定）。

注解

如果您要随机访问表中的单个行，则表中数据的实际顺序并不重要。但是，如果您倾向于访问比其他数据更多的数据，并且有一个将它们分组在一起的索引，则可以从使用CLUSTER中受益。如果要从表中请求某个范围的索引值，或者是具有多个匹配行的单个索引值，则CLUSTER会有所帮助，因为一旦索引为匹配的第一行标识了表页面，则所有其他匹配的行都可能已经在同一表页面上，因此您可以节省磁盘访问并加快查询速度。

CLUSTER可以使用对指定索引的索引扫描或顺序扫描后再排序来对表进行重新排序（如果索引是b树）。它将尝试根据优化器成本参数和可用的统计信息来选择更快的方法。

使用索引扫描时，将创建表的临时副本，该副本包含按索引顺序排列的表数据。还将创建表上每个索引的临时副本。因此，磁盘上需要的可用空间至少等于表大小和索引大小的总和。

当使用顺序扫描和排序时，还会创建一个临时排序文件，因此临时空间的峰值需求是表大小加索引大小的两倍。此方法通常比索引扫描方法快，但是如果磁盘空间需求无法忍受，则可以通过将enable_sort 配置参数临时设置为off来禁用此选择。

建议在聚簇之前将maintenance_work_mem 配置参数设置为一个相当大的值（但不要大于您可以用于CLUSTER操作的RAM数量）。

因为查询优化器会记录有关表顺序的统计信息，所以建议在新的聚簇表上运行ANALYZE。否则，优化器可能会选择错误的查询计划。

由于CLUSTER会记住对哪些索引进行了聚簇，因此您可以在第一次手动对要聚簇的表进行聚簇，然后设置一个定期维护脚本，该脚本无需任何参数即可执行CLUSTER，以便定期重新聚簇所需的表。

Note: 追加优化表不支持CLUSTER。

示例

根据索引emp_ind对表employees进行聚簇：

```
CLUSTER emp_ind ON emp;
```

通过重新创建大型表并将其以正确的索引顺序加载来对大型表进行聚簇：

```
CREATE TABLE newtable AS SELECT * FROM table ORDER BY
column;
DROP table;
ALTER TABLE newtable RENAME TO table;
CREATE INDEX column_ix ON table (column);
VACUUM ANALYZE table;
```

兼容性

SQL标准中没有CLUSTER语句。

另见

[CREATE TABLE AS](#) , [CREATE INDEX](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义或更改对象的注释。

概要

```

COMMENT ON
{ TABLE object_name |
COLUMN relation_name.column_name |
AGGREGATE agg_name (agg_type [, ...]) |
CAST (source_type AS target_type) |
COLLATION object_name
CONSTRAINT constraint_name ON table_name |
CONVERSION object_name |
DATABASE object_name |
DOMAIN object_name |
EXTENSION object_name |
FOREIGN DATA WRAPPER object_name |
FOREIGN TABLE object_name |
FUNCTION func_name ([[argmode] [argname] argtype [, ...]]) |
INDEX object_name |
LARGE OBJECT large_object_oid |
OPERATOR operator_name (left_type, right_type) |
OPERATOR CLASS object_name USING index_method |
[PROCEDURAL] LANGUAGE object_name |
RESOURCE GROUP object_name |
RESOURCE QUEUE object_name |
ROLE object_name |
RULE rule_name ON table_name |
SCHEMA object_name |
SEQUENCE object_name |
SERVER object_name |
TABLESPACE object_name |
TRIGGER trigger_name ON table_name |
TYPE object_name |
VIEW object_name }
IS 'text'

```

描述

COMMENT存储有关数据库对象的注释。每个对象仅存储一个注释字符串。要删除注释，请在文本字符串的位置写入NULL。删除对象时，注释将自动删除。

对于大多数对象，只有对象的所有者才能设置注释。角色没有所有者，因此COMMENT ON ROLE的规则是您必须是超级用户才能对超级用户角色进行评论，或者具有CREATEROLE特权才能对非超级用户角色添加注释。当然，超级用户可以对任何对象添加注释。



使用psql元命令\dd, \d+和\l+可以轻松提取注释。可以在psql使用的相同内置函数
(即obj_description, col_description和shobj_description)
之上构建其他用于提取注释的用户界面。

参数

object_name

relation_name.column_name

agg_name

constraint_name

func_name

operator_name

rule_name

trigger_name

要注释的对象的名称。表，聚合，排序规则，转换，域，外部表，函数，索引，运算符，运算符类，运算符族，序列，文本搜索对象，类型和视图的名称可以通过模式指定。在对列进行注释时，*relation_name*必须引用表，视图，组合类型或外部表。

Note: Greenplum数据库不支持触发器。

aggregate_type

聚合函数所基于的输入数据类型。要引用零参数聚合函数，请写*代替输入数据类型列表。

source_type

强制转换的源数据类型的名称。

target_type

强制转换的目标数据类型的名称。

argmode

函数参数的模式：IN, OUT, INOUT或VARIADIC。如果省略，则默认值为IN。注意，由于仅需要输入参数来确定函数的标识，因此COMMENT ON FUNCTION实际上并不对OUT参数给予任何关注。因此，列出IN, INOUT和VARIADIC参数就足够了。

argname

函数参数的名称。请注意，COMMENT ON FUNCTION实际上并不关注参数名称，因为仅需要参数数据类型来确定函数的身份。

argtype

函数参数的数据类型（可以由模式指定）（如果有）。

large_object_oid

大对象的OID。

Note: Greenplum数据库不支持PostgreSQL[大对象工具](#)来流存储在大对象结构中的用户数据。

left_type

right_type

运算符参数的数据类型（可以由模式指定）。为前缀或后缀运算符的缺少参数写NONE。

PROCEDURAL

这是一个干扰词。

text

新注释，以字符串文字形式表示；或NULL以删除注释。

注解

当前没有用于查看注释的安全机制：连接到数据库的任何用户都可以查看该数据库中对象的所有注释。对于共享对象，例如数据库，角色和表空间，注释是全局存储的，因此连接到集群中任何数据库的任何用户都可以查看共享对象的所有注释。因此，请勿在评论中添加对安全性至关重要的信息。

示例

在表mytable上添加注释：

```
COMMENT ON TABLE mytable IS 'This is my table.';
```

再次删除：

```
COMMENT ON TABLE mytable IS NULL;
```

其他示例：

```
COMMENT ON AGGREGATE my_aggregate (double precision) IS  
'Computes sample variance';  
COMMENT ON CAST (text AS int4) IS 'Allow casts from text to  
int4';  
COMMENT ON COLLATION "fr_CA" IS 'Canadian French';  
COMMENT ON COLUMN my_table.my_column IS 'Employee ID number';  
COMMENT ON CONVERSION my_conv IS 'Conversion to UTF8';  
COMMENT ON CONSTRAINT bar_col_cons ON bar IS 'Constrains column  
col';  
COMMENT ON DATABASE my_database IS 'Development Database';  
COMMENT ON DOMAIN my_domain IS 'Email Address Domain';  
COMMENT ON EXTENSION hstore IS 'implements the hstore data  
type';  
COMMENT ON FOREIGN DATA WRAPPER mywrapper IS 'my foreign data  
wrapper';  
COMMENT ON FOREIGN TABLE my_foreign_table IS 'Employee  
Information in other database';  
COMMENT ON FUNCTION my_function (timestamp) IS 'Returns Roman  
Numeral';  
COMMENT ON INDEX my_index IS 'Enforces uniqueness on employee  
ID';  
COMMENT ON LANGUAGE plpython IS 'Python support for stored  
procedures';
```

```
COMMENT ON LARGE OBJECT 346344 IS 'Planning document';
COMMENT ON OPERATOR ^ (text, text) IS 'Performs intersection of
two texts';
COMMENT ON OPERATOR - (NONE, integer) IS 'Unary minus';
COMMENT ON OPERATOR CLASS int4ops USING btree IS '4 byte integer
operators for btrees';
COMMENT ON OPERATOR FAMILY integer_ops USING btree IS 'all
integer operators for btrees';
COMMENT ON ROLE my_role IS 'Administration group for finance
tables';
COMMENT ON RULE my_rule ON my_table IS 'Logs updates of employee
records';
COMMENT ON SCHEMA my_schema IS 'Departmental data';
COMMENT ON SEQUENCE my_sequence IS 'Used to generate primary
keys';
COMMENT ON SERVER myserver IS 'my foreign server';
COMMENT ON TABLE my_schema.my_table IS 'Employee Information';
COMMENT ON TABLESPACE my_tablespace IS 'Tablespace for indexes';
COMMENT ON TEXT SEARCH CONFIGURATION my_config IS 'Special word
filtering';
COMMENT ON TEXT SEARCH DICTIONARY swedish IS 'Snowball stemmer
for Swedish language';
COMMENT ON TEXT SEARCH PARSER my_parser IS 'Splits text into
words';
COMMENT ON TEXT SEARCH TEMPLATE snowball IS 'Snowball stemmer';
COMMENT ON TRIGGER my_trigger ON my_table IS 'Used for RI';
COMMENT ON TYPE complex IS 'Complex number data type';
COMMENT ON VIEW my_view IS 'View of departmental costs';
```

兼容性

SQL标准中没有COMMENT语句。

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)[ABORT](#)[ALTER AGGREGATE](#)[ALTER COLLATION](#)[ALTER CONVERSION](#)[ALTER DATABASE](#)[ALTER DEFAULT PRIVILEGES](#)[ALTER DOMAIN](#)[ALTER EXTENSION](#)[ALTER EXTERNAL TABLE](#)[ALTER FOREIGN DATA WRAPPER](#)[ALTER FOREIGN TABLE](#)[ALTER FUNCTION](#)[ALTER GROUP](#)[ALTER INDEX](#)[ALTER LANGUAGE](#)[ALTER OPERATOR](#)

提交当前事务。

概要

COMMIT [WORK | TRANSACTION]

描述

COMMIT提交当前事务。 事务所做的所有更改对其他人都可见，并且如果发生崩溃，则保证是持久的。

参数

WORK

TRANSACTION

可选关键字。它们没有作用。

注解

使用[ROLLBACK](#)中止事务。

如果不在事务内部，则发出commit不会造成任何危害，但是会引发警告消息。

示例

提交当前事务并使所有更改永久生效：

COMMIT;



兼容性

SQL标准仅指定两种形式： COMMIT和COMMIT WORK。 否则，此命令完全符合要求。

另见

[BEGIN , END , START TRANSACTION , ROLLBACK](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

在文件和表之间复制数据。

概要

```
COPY table_name [(column_name [, ...])]  
FROM {'filename' | PROGRAM 'command' | STDIN}  
[ [ WITH ] ( option [, ...] ) ]  
[ ON SEGMENT ]  
  
COPY { table_name [(column_name [, ...])] | (query) }  
TO {'filename' | PROGRAM 'command' | STDOUT}  
[ [ WITH ] ( option [, ...] ) ]  
[ ON SEGMENT ]
```

其中*option*可以是以下之一：

```
FORMAT format_name  
OIDS [ boolean ]  
FREEZE [ boolean ]  
DELIMITER 'delimiter_character'  
NULL 'null string'  
HEADER [ boolean ]  
QUOTE 'quote_character'  
ESCAPE 'escape_character'  
FORCE_QUOTE { ( column_name [, ...] ) | * }  
FORCE_NOT_NULL ( column_name [, ...] )  
ENCODING 'encoding_name'  
FILL MISSING FIELDS  
LOG ERRORS [ SEGMENT REJECT LIMIT count [ ROWS | PERCENT ] ]  
IGNORE EXTERNAL PARTITIONS
```

描述

COPY在Greenplum数据库表和标准文件系统文件之间移动数据。

COPY TO将表的内容复制到一个文件（如果在ON SEGMENT上复制，则将基于segment ID复制到多个文件），而COPY FROM将数据从文件复制到表（将数据追加到表中已有的任何内容）。COPY TO还可以复制SELECT查询的结果。

如果指定了列列表，则COPY仅将指定列中的数据复制到文件中或从文件复制。如果表中的任何列不在列列表中，则COPY FROM将为这些



列插入默认值。

带有文件名的COPY指示Greenplum数据库master主机直接从文件读取或写入文件。该文件必须可供master主机访问，并且必须从master主机的角度指定名称。

当COPY与ON SEGMENT子句一起使用时，ON SEGMENT导致segment创建单独的面向segment的文件，这些文件保留在segment主机上。ON SEGMENT的*filename*参数采用字符串文字<SEGID>（必需），并使用绝对路径或<SEG_DATA_DIR>字符串文字。运行COPY操作时，segment ID和segment数据目录的路径将替换为字符串文字值。

使用复制表（DISTRIBUTED REPLICATED）作为源的COPY TO将创建一个文件，其中包含来自单个segment的行，以便目标文件不包含重复的行。将COPY TO与ON SEGMENT子句一起使用，并将复制表作为源，则在包含所有表行的segment主机上创建目标文件。

ON SEGMENT子句允许您将表数据复制到segment主机上的文件中，以用于诸如在集群之间迁移数据或执行备份之类的操作。可以使用诸如gpfldist之类的工具来恢复由ON SEGMENT子句创建的细分数据，这对于高速数据加载非常有用。

Warning: 建议仅对专业用户使用ON SEGMENT子句。

指定PROGRAM时，服务器将执行给定命令并从程序的标准输出中读取或写入程序的标准输入。该命令必须从服务器的角度指定，并且可由gpadmin用户执行。

当指定STDIN或STDOUT时，数据将通过客户端和master之间的连接进行传输。STDIN和STDOUT不能与ON SEGMENT子句一起使用。

如果使用SEGMENT REJECT LIMIT，则COPY FROM操作将在单行错误隔离模式下运行。在此版本中，单行错误隔离模式仅适用于输入文件中格式错误的行 - 例如，多余或缺失的属性，错误的数据类型的属性或无效的客户端编码序列。约束错误（例如违反NOT NULL，CHECK或UNIQUE约束）仍将在“全有或全无”输入模式下处理。用户可以指定可接受的错误行数（基于每个segment），之后将终止整个COPY FROM操作，并且不会加载任何行。错误行的计数是按segment而不是整个加载操作计数的。如果未达到每个segment拒绝的限制，那么将加载所有不包含错误的行，并丢弃所有错误行。要保留错误行以供进一步检查，请指定LOG ERRORS子句以捕获错误日志信息。错误信息和该行存储在Greenplum内部数据库中。

输出

成功完成后，COPY命令将返回以下形式的命令标签，其中*count*是复制的行数：

```
COPY count
```

如果以单行错误隔离模式运行COPY FROM命令，如果由于格式错误而未加载任何行，则将返回以下通知消息，其中*count*是拒绝的行数：

```
NOTICE: Rejected count badly formatted rows.
```

参数

table_name

现有表的名称（可以由模式指定）。

column_name

要复制的列的可选列表。如果未指定列列表，则将复制表的所有列。

当以文本格式复制时，默认情况下，bytea类型的列中的一行数据最多可为256MB。

query

SELECT或VALUES命令，其结果将被复制。请注意，查询周围需要括号。

filename

输入或输出文件的路径名。输入文件名可以是绝对路径或相对路径，但输出文件名必须是绝对路径。Windows用户可能需要使用“'”字符串并将路径名中使用的所有反斜杠加倍。

PROGRAM '*command*'

指定要执行的命令。在COPY FROM中，从命令的标准输出中读取输入，而在COPY TO中，将输出写入命令的标准输入中。必须从Greenplum数据库master主机系统的角度指定该*command*，并且该命令必须由Greenplum数据库管理员用户（gpadmin）执行。

该*command*由shell程序调用。将参数传递给shell时，请删除或转义对shell具有特殊含义的任何特殊字符。出于安全原因，最好使用固定的命令字符串，或者至少避免在字符串中传递任何用户输入。

当指定ON SEGMENT时，该命令必须由Greenplum数据库管理员用户（gpadmin）在所有Greenplum数据库primary segment主机上可执行。该命令由每个Greenplum segment实例执行。
<SEGID>在*command*中是必需的。

有关命令语法要求和指定该子句时要复制的数据的信息，请参

见ON SEGMENT子句。

STDIN

指定输入来自客户端应用程序。 STDIN不支持ON SEGMENT子句。

STDOUT

指定将输出发送到客户端应用程序。 STDOUT不支持ON SEGMENT子句。

boolean

指定是打开还是关闭所选选项。 您可以写入TRUE, ON或1以启用该选项， 而可以写入FALSE, OFF或0以禁用该选项。 布尔值也可以省略，在这种情况下，假定为TRUE。

FORMAT

选择要读取或写入的数据格式： text, csv (逗号分隔值) 或binary。 默认为text。

OIDS

指定为每行复制OID。 (如果为没有OID的表指定了OIDS，或者在复制查询的情况下，将引发错误。)

FREEZE

请求复制已冻结的行的数据，就像在运行VACUUM FREEZE命令之后一样。 这旨在用作初始数据加载的性能选项。 仅当在当前子事务中已创建或截断了要加载的表，没有打开游标并且该事务没有任何较旧的快照时，行才会被冻结。

请注意，一旦成功加载数据，所有其他会话将立即能够看到数据。 这违反了MVCC可见性的常规规则，指定此选项的用户应注意可能引起的潜在问题。

DELIMITER

指定用于分隔文件每一行（行）中各列的字符。 默认为text格式的制表符，csv格式的逗号。 这必须是一个单字节字符。 使用binary格式时，不允许使用此选项。

NULL

指定表示空值的字符串。 默认值为文本格式\N (反斜杠-N)，以及csv格式的无引号的空字符串。 对于不想将空值与空字符串区分开的情况，甚至可能以text格式使用空字符串。 使用binary格式时，不允许使用此选项。

Note: 使用COPY FROM时，与该字符串匹配的任何数据项都将存储为空值，因此您应确保使用与COPY TO相同的字符串。

HEADER

指定文件包含标题行，其中包含文件中每一列的名称。 输出时，第一行包含表中的列名，输入时，第一行被忽略。 仅当使用csv格式时才允许使用此选项。

QUOTE

指定在引用数据值时要使用的引用字符。 默认值为双引号。 这必须是一个单字节字符。 仅当使用csv格式时才允许使用此选项。

ESCAPE

指定应该出现在与QUOTE值匹配的数据字符之前的字符。缺省值与QUOTE值相同（因此，如果引号字符出现在数据中，则将引号字符加倍）。这必须是一个单字节字符。仅当使用csv格式时才允许使用此选项。

FORCE_QUOTE

强制将引号用于每个指定列中的所有非NULL值。NULL输出从不引用。如果指定*，则在所有列中都引用非NULL值。仅在COPY TO中和使用csv格式时才允许使用此选项。

FORCE_NOT_NULL

不要将指定列的值与空字符串匹配。在默认情况下，空字符串为空时，这意味着空值将被读取为零长度的字符串，而不是空值，即使没有引号也是如此。仅在COPY FROM中和使用csv格式时才允许使用此选项。

ENCODING

指定文件以*encoding_name*编码。如果省略此选项，则使用当前的客户端编码。有关更多详细信息，请参见下面的注释。

ON SEGMENT

在segment主机上指定各个segment数据文件。每个文件都包含由primary segment实例管理的表数据。例如，当使用COPY TO... ON SEGMENT命令从表中将数据复制到文件时，该命令会在segment主机上为主机上的每个segment实例创建一个文件。每个文件都包含由segment实例管理的表数据。

COPY命令不会从mirror实例和segment数据文件之间复制数据。
ON SEGMENT不支持关键字STDIN和STDOUT。

<SEG_DATA_DIR>和<SEGID>字符串文字用于使用以下语法指定绝对路径和文件名：

```
COPY table [TO|FROM]
'<SEG_DATA_DIR>/gpdumpname<SEGID>_suffix' ON SEGMENT;
```

<SEG_DATA_DIR>

字符串文字，表示用于ON SEGMENT复制的segment实例数据目录的绝对路径。尖括号(<和>)是用于指定路径的字符串文字的一部分。运行COPY时，COPY用segment路径替换字符串文字。可以使用绝对路径代替<SEG_DATA_DIR>字符串文字。

<SEGID>

字符串文字，表示在复制ON SEGMENT时要复制的segment实例的content ID号。当指定ON SEGMENT时，<SEGID>是文件名的必需部分。尖括号是用于指定文件名的字符串文字的一部分。
使用COPY TO，当运行COPY命令时，字符串文字将被segment实例的content ID替换。

使用COPY FROM, 在文件名中指定segment实例content ID, 然后将该文件放在segment实例主机上。每个主机上的每个primary segment实例必须有一个文件。运行COPY FROM命令时, 数据从文件复制到segment实例。

当指定了PROGRAM *command*子句时, *command*中需要<SEGID>字符串文字, 而<SEG_DATA_DIR>字符串文字是可选的。请参阅[示例](#)。

对于COPY FROM...ON SEGMENT命令, 将数据复制到表中时将检查表分配策略。默认情况下, 如果数据行违反表分发策略, 则返回错误。您可以使用服务器配置参数gp_enable_segment_copy_checking禁用分发策略检查。请参阅[注释](#)。

NEWLINE

指定数据文件中使用的换行符 - LF (换行, 0x0A), CR (回车, 0x0D), 或CRLF (回车加换行, 0x0D 0x0A)。如果未指定, Greenplum数据库segment将通过查看接收到的第一行数据并使用遇到的第一个换行符类型来检测换行符类型。

CSV

选择逗号分隔值 (CSV) 模式。请参阅[CSV格式](#)。

FILL MISSING FIELDS

在TEXT和CSV的COPY FROM more中, 指定FILL MISSING FIELDS会将行尾缺少数据字段的数据行设置为NULL (而不是报告错误)。空白行, 具有NOT NULL约束的字段以及行尾的定界符仍然会报告错误。

LOG ERRORS

这是一个可选的子句, 可以在SEGMENT REJECT LIMIT子句之前, 以捕获有关具有格式错误的行的错误日志信息。

错误日志信息存储在内部, 并可以通过Greenplum数据库内置的SQL函数gp_read_error_log()进行访问。

请参阅[注解](#)以获取有关错误日志的信息以及用于查看和管理错误日志信息的内置函数。

SEGMENT REJECT LIMIT count [ROWS | PERCENT]

在单行错误隔离模式下运行COPY FROM操作。如果输入行存在格式错误, 只要在加载操作期间未在任何Greenplum数据库segment实例上达到拒绝限制计数, 它们将被丢弃。拒绝限制计数可以指定为行数 (默认) 或总行数的百分比 (1-100)。如果使用PERCENT, 则只有在处理了参

数gp_reject_percent_threshold指定的行数之后, 每个segment才开始计算错误行百分比。

gp_reject_percent_threshold的默认值为300行。约束错误 (例如违反NOT NULL, CHECK或UNIQUE约束) 仍将在“全有或全无”输入模式下处理。如果未达到限制, 则将加载所有正确的行, 并丢弃所有错误行。

Note: 如果未先触发或未指定SEGMENT REJECT LIMIT，则Greenplum数据库会限制可能包含格式错误的初始行数。如果前1000行被拒绝，则COPY操作将停止并回滚。

可以使用Greenplum数据库服务器配置参数gp_initial_bad_row_limit更改初始拒绝行数的限制。有关参数的信息，请参阅[服务器配置参数](#)。

IGNORE EXTERNAL PARTITIONS

从分区表复制数据时，不会从外部表的叶子分区复制数据。不复制数据时，将在日志文件中添加一条消息。

如果未指定此子句，并且Greenplum数据库尝试从作为外部表的叶子分区中复制数据，则返回错误。

有关指定SQL查询以从作为外部表的叶子分区中复制数据的信息，请参见下一节“注释”。

注解

COPY只能与表一起使用，而不能与外部表或视图一起使用。但是，您可以执行COPY (SELECT * FROM viewname) TO ...

当指定ON SEGMENT子句时，COPY命令不支持在COPY TO命令中指定SELECT语句。例如，不支持此命令。

```
COPY (SELECT * FROM testtbl) TO '/tmp/mytst<SEGID>' ON
SEGMENT
```

COPY仅处理特定名称的表，它不会在子表之间复制数据。因此，例如COPY table TO显示的数据与SELECT * FROM ONLY table相同。但是COPY (SELECT * FROM table) TO ...可用于转储继承层次结构中的所有数据。

同样，要从具有作为外部表的叶子分区的分区表中复制数据，请使用SQL查询来复制数据。例如，如果表my_sales包含带有作为外部表的叶子分区，则此命令COPY my_sales TO stdout返回错误。此命令将数据发送到stdout：

BINARY关键字使所有数据以二进制格式而不是文本形式存储/读取。它比普通的文本模式要快一些，但是二进制格式的文件在计算机体系结构和Greenplum数据库版本之间的移植性较差。另外，如果数据为二进制格式，则不能以单行错误隔离模式运行COPY FROM。

您必须对通过COPY TO读取的表具有SELECT特权，并且对于通

过COPY FROM将值插入其中的表具有插入特权。在命令中列出的列上具有列特权就足够了。

COPY命令中命名的文件由数据库服务器而不是客户端应用程序直接读取或写入。因此，它们必须位于Greenplum数据库master主机上（而不是客户端）或可被其访问。Greenplum数据库系统用户（服务器运行时使用的用户ID），而不是客户端，必须可以访问它们并对其进行读写。COPY命名文件只允许数据库超级用户使用，因为它允许读取或写入服务器有权访问的任何文件。

COPY FROM将调用任何触发器并检查目标表上的约束。但是，它不会调用重写规则。请注意，在此版本中，不针对单行错误隔离模式评估违反约束的情况。

COPY输入和输出受DateStyle影响。为了确保可移植到可能使用非默认DateStyle设置的其他Greenplum数据库安装，在使用COPY TO之前，应将DateStyle设置为ISO。避免转储IntervalStyle设置为sql_standard的数据也是一个好主意，因为对于IntervalStyle不同设置的服务器可能会误解负间隔值。

输入数据将根据ENCODING选项或当前客户端编码进行解释，而输出数据将以ENCODING或当前客户端编码进行编码，即使数据没有通过客户端，而是由服务器直接从文件读取或写入文件。

在文本模式下从文件复制XML数据时，服务器配置参数xmloption会影响对复制的XML数据的验证。如果该值为content（默认值），则将XML数据验证为XML内容片段。如果参数值为document，则将XML数据验证为XML文档。如果XML数据无效，则COPY返回错误。

默认情况下，COPY在第一个错误时停止操作。如果是COPY TO，这应该不会导致问题，但是目标表已经在COPY FROM中接收到了较早的行。这些行将不可见或不可访问，但仍会占用磁盘空间。如果故障在大型COPY FROM操作中发生，则可能会浪费大量磁盘空间。您可能希望调用VACUUM来恢复浪费的空间。另一种选择是使用单行错误隔离模式来过滤掉错误行，同时仍然加载正确的行。

运行COPY FROM...ON SEGMENT命令时，服务器配置参数gp_enable_segment_copy_checking控制在将数据复制到表中时是否检查表分发策略（来自表DISTRIBUTED子句）。默认设置为检查分发策略。如果数据行违反了segment实例的分配策略，则返回错误。有关该参数的信息，请参阅[服务器配置参数](#)。

COPY TO...ON SEGMENT命令生成的表数据可用于通过COPY FROM...ON SEGMENT恢复表数据。但是，使用COPY TO命令生成

文件时，将根据表分发策略来分发恢复到segment的数据。如果尝试还原表数据并且在运行COPY FROM...ON SEGMENT之后更改了表分发策略，则COPY命令可能会返回表分发策略错误。

Note: 如果运行COPY FROM...ON SEGMENT，并且服务器配置参数gp_enable_segment_copy_checking为false，则可能需要手动重新分配表数据。请参阅ALTER TABLE子句WITH REORGANIZE。

当您指定LOG ERRORS子句时，Greenplum数据库将捕获读取外部表数据时发生的错误。您可以查看和管理捕获的错误日志数据。

- 使用内置的SQL函数gp_read_error_log('table_name')。它要求对具有SELECT特权。本示例使用COPY命令显示加载数据到表ext_expenses的错误日志信息：

```
SELECT * from gp_read_error_log('ext_expenses');
```

有关错误日志格式的信息，请参阅Greenplum数据库管理员指南中的[查看错误日志中的坏行](#)。

如果不存在，则该函数返回FALSE。

- 如果指定表存在错误日志数据，则新的错误日志数据将附加到现有错误日志数据中。错误日志信息不会复制到mirror。
- 使用内置的SQL函数gp_truncate_error_log('table_name')删除的错误日志数据。它需要表所有者特权。本示例删除将数据移入表ext_expenses时捕获的错误日志信息：

```
SELECT gp_truncate_error_log('ext_expenses');
```

如果不存在，则该函数返回FALSE。

指定*通配符可删除当前数据库中所有表的错误日志信息。指定字符串*.*以删除所有数据库错误日志信息，包括由于先前的数据库问题而未被删除的错误日志信息。如果指定*，则需要数据库所有者特权。如果指定*.*，则需要操作系统超级用户特权。

当不是超级用户的Greenplum数据库用户运行COPY命令时，该命令可以由资源队列控制。资源队列必须配置有ACTIVE_STATEMENTS参数，该参数指定分配给该队列的角色可以执行的查询数量的最大限制。Greenplum数据库不会将成本值或内存值应用于COPY命令，仅有成本或内存限制的资源队列不会影响COPY命令的运行。

非超级用户只能运行以下类型的COPY命令：

- 源为stdin的COPY FROM命令

- 目标为stdout的COPY TO命令

有关资源队列的信息，请参阅*Greenplum*数据库管理员指南中的“使用资源队列进行资源管理”。

文件格式

COPY支持的文件格式。

文本格式

使用文本格式时，读取或写入的数据是一个文本文件，每行一行。行中的列由*delimiter_character*分隔（默认为制表符）。列值本身是由每个属性的数据类型的输出函数生成的字符串，或输入函数可接受的字符串。使用空字符串代替空列。如果输入文件的任何行包含的列比预期的多或少，则COPY FROM将引发错误。如果指定了OIDS，则将OID读取或写入为用户数据列之前的第一列。

数据文件具有两个保留字符，它们对于COPY具有特殊含义：

- 指定的分隔符（默认为制表符），用于分隔数据文件中的字段。
- UNIX样式的换行符（\n或0x0a），用于在数据文件中指定新行。强烈建议生成COPY数据的应用程序将数据换行转换为UNIX样式的换行，而不是Microsoft Windows样式的回车换行（\r\n或0x0a 0xd）。

如果数据包含这些字符中的任何一个，则必须转义该字符，以便COPY将其视为数据而不是字段分隔符或新行。

默认情况下，对于文本格式的文件，转义字符是\（反斜杠），对于csv格式的文件，转义字符是"（双引号）。如果要使用其他转义字符，则可以使用ESCAPE AS子句来实现。确保选择一个在数据文件中任何地方都没有使用的转义字符作为实际数据值，也可以通过使用ESCAPE 'OFF'来禁用文本格式文件中的转义。

例如，假设您有一个包含三列的表，并且想使用COPY加载以下三个字段。

- 百分号 = %
- 竖线 = |
- 反斜杠 = \

您指定的*delimiter_character*为|（竖线字符），而您指定的转义字符

为*（星号）。数据文件中的格式化行如下所示：

```
percentage sign = % | vertical bar = *| | backslash = \
```

请注意，如何使用星号字符（*）对作为数据一部分的管道字符进行转义。另请注意，由于我们使用的是替代转义字符，因此我们不需要转义反斜杠。

如果以下字符作为列值的一部分出现，则必须在其后加上转义字符：转义字符本身，换行符，回车符和当前定界符。您可以使用`ESCAPE AS`子句指定其他转义字符。

CSV格式

此格式选项用于导入和导出许多其他程序（例如电子表格）使用的逗号分隔值（CSV）文件格式。它生成并识别常见的CSV转义机制，而不是Greenplum数据库标准文本格式使用的转义规则。

每个记录中的值由`DELIMITER`字符分隔。如果值包含定界符，`QUOTE`字符，`ESCAPE`字符（默认情况下为双引号），`NULL`字符串，回车符或换行符，则整个值将以`QUOTE`字符作为前缀和后缀。在特定列中输出非`NULL`值时，也可以使用`FORCE_QUOTE`强制使用引号。

CSV格式没有区分`NULL`值和空字符串的标准方法。Greenplum数据库`COPY`通过引用进行处理。输出`NULL`作为`NULL`参数字符串，并且不加引号，而与`NULL`字符串匹配的非`NULL`值被加引号。例如，使用默认设置，将`NULL`写入未加引号的空字符串，而将空字符串数据值写入双引号（" "）。读取值遵循类似的规则。您可以用`FORCE_NOT_NULL`来防止对特定列进行`NULL`输入比较。

由于反斜杠不是csv格式的特殊字符，因此\.（数据结尾标记）也可能会显示为数据值。为避免任何误解，使用\.在行上显示为单独条目的数据值会在输出上自动加引号，并且在输入上（如果加引号）也不会被解释为数据结束标记。如果要加载的文件是由另一个应用程序创建的，该文件具有未加引号的单列并且可能具有\.值，则可能需要在输入文件中使用该值。

Note: 在csv格式中，所有字符均有效。用引号括起来的空格或除`DELIMITER`以外的任何字符都将包含这些字符。如果从空白行填充到固定宽度的CSV行的系统中导入数据，则可能会导致错误。如果出现这种情况，在将数据导入Greenplum数据库之前，可能需要预处理CSV文件以删除尾随空白。

CSV格式将识别并生成带有引用值的CSV文件，这些值包含嵌入式回车符和换行符。因此，与文本格式的文件相比，文件并非严格限于每行一行。

Note: 许多程序会生成奇怪的（有时是错误的）CSV文件，因此文件格式更像是约定俗成的标准文件。因此，您可能会遇到一些无法使用此机制导入的文件，并且COPY可能会生成其他程序无法处理的文件。

二进制格式

二进制格式选项使所有数据以二进制格式而不是文本形式存储/读取。它比文本和CSV格式要快一些，但是二进制格式的文件在计算机体系结构和Greenplum数据库版本之间的可移植性较差。同样，二进制格式是非常特定于数据类型的。例如，即使在文本格式下也可以正常工作，但从smallint列输出二进制数据并将其读入整数列将不起作用。

二进制文件格式由文件头，包含行数据的零个或多个元组和文件尾部组成。文件头和数据按网络字节序排列。

- 文件头 — 文件头由15个字节的固定字段组成，后跟可变长度的头扩展区。固定字段是：
 - 签名 — 11字节序列PGCOPY\n\377\r\n\0 - 请注意，零字节是签名的必需部分。（签名旨在方便识别非8位纯净传输所破坏的文件。此签名将通过行尾翻译过滤器，丢失的零字节，丢失的高位或奇偶性更改。）
 - 标识字段 — 32位整数位掩码，表示文件格式的重要方面。位的编号从0 (LSB) 到31 (MSB) 。请注意，该字段以网络字节序存储（最高有效字节在前），文件格式中使用的所有整数字段也是如此。保留位16-31标志关键的文件格式问题；如果读取器发现在此范围内设置了意外的位，则应该中止。保留位0-15标志向后兼容的格式问题；读者只需忽略此范围内设置的任何意外位。当前仅定义一个标志，其余标志必须为零（如果数据具有OID，则位16为1，否则为0）。
 - 文件头扩展区长度 — 32位整数，文件头其余部分的字节长度，不包括其自身。当前，它为零，并且第一个元组紧随其后。将来对格式进行的更改可能会允许其他数据出现在文件头中。读取器应静默跳过任何不知道该怎么处理的文件头扩展数据。文件头扩展区被设想为包含一系列自识别块。标志字段不是要告诉读取器扩展区中的内容。文件头扩展内容的特定设计留给以后的版本。
- 元组 — 每个元组都以元组中字段数的16位整数计数开头。（当前，表中的所有元组将具有相同的计数，但可能并不总是正确的。）然后，对于元组中的每个字段重复一次，存在一个32位长的字，后跟那么多字节的字段数据。（长度字不包括自身，并且可以

为零。) 在特殊情况下，-1表示NULL字段值。在NULL情况下，没有值字节。

字段之间没有对齐填充或任何其他额外数据。

当前，二进制格式文件中的所有数据值都假定为二进制格式（格式代码1）。可以预期，将来的扩展可能会添加一个文件头字段，以允许指定每列格式代码。

如果文件中包含OID，则OID字段紧跟在字段计数字之后。这是一个普通字段，只是它不包括在字段计数中。特别是它有一个长度字 - 这将允许处理4字节和8字节的OID，而不会带来太多麻烦，并且如果确实需要的话，也可以将OID显示为null。

- 文件尾 — 文件尾部由一个包含-1的16位整数字组成。这很容易与元组的字段计数字区分开。如果字段计数字既不是-1也不是预期的列数，则读取器应报告错误。这提供了额外的检查，以防止以某种方式与数据不同步。

示例

使用竖线 (|) 作为字段定界符，将表复制到客户端：

```
COPY country TO STDOUT (DELIMITER '|');
```

将数据从文件复制到country表：

```
COPY country FROM '/home/usr1/sql/country_data';
```

仅将名称以'A'开头的国家复制到文件中：

```
COPY (SELECT * FROM country WHERE country_name LIKE 'A%')
TO
'/home/usr1/sql/a_list_countries.copy';
```

使用单行错误隔离模式将数据从文件复制到sales表中并记录错误：

```
COPY sales FROM '/home/usr1/sql/sales_data' LOG ERRORS
SEGMENT REJECT LIMIT 10 ROWS;
```

若要复制segment数据供以后使用，请使用ON SEGMENT子句。COPY TO ON SEGMENT命令的使用形式如下：

```
COPY table TO '<SEG_DATA_DIR>/gpdumpname<SEGID>_suffix' ON
SEGMENT;
```

<SEGID>是必需的。但是，您可以用绝对路径替换路径中的<SEG_DATA_DIR>字符串文字。

将字符串文字<SEG_DATA_DIR>和<SEGID>传递给COPY时，运行操作时COPY将填充适当的值。

例如，如果您的mytable包含以下segment和mirror：

contentid	dbid	file	segment	location
0	1	/home/usr1	data1	/gpsegdir0
0	3	/home/usr1	data_mirror1	/gpsegdir0
1	4	/home/usr1	data2	/gpsegdir1
1	2	/home/usr1	data_mirror2	/gpsegdir1

运行命令：

```
COPY mytable TO '<SEG_DATA_DIR>/gpbackup<SEGID>.txt' ON
SEGMENT;
```

将产生如下行：

```
/home/usr1/data1/gpsegdir0/gpbackup0.txt
/home/usr1/data2/gpsegdir1/gpbackup1.txt
```

第一列中的内容ID是插入文件路径中的标识符（例如，上面的gpsegdir0/gpbackup0.txt）。文件是在segment上而不是在master上创建的，就像在标准COPY操作中那样。使用ON SEGMENT复制时，不会为mirror创建任何数据文件。

如果指定了绝对路径，而不是<SEG_DATA_DIR>，例如在语句中

```
COPY mytable TO '/tmp/gpdir/gpbackup_<SEGID>.txt' ON
SEGMENT;
```

文件将放置在每个segment的/tmp/gpdir中。如果需要重新分发，gpfdist工具还可用于使用ON SEGMENT选项还原由COPY TO生成的数据文件。

Note: 可以使用诸如gpfdist之类的工具来还原数据。备份/还原工具不适用于使用COPY TO ON SEGMENT手动生成的文件。

本示例从lineitem表复制数据，并使用PROGRAM子句使用cat实用程序将数据添加到/tmp/lineitem_program.csv文件。该文件放置在Greenplum数据库master上。

```
COPY LINEITEM TO PROGRAM 'cat > /tmp/lineitem.csv' CSV;
```

本示例使用PROGRAM和ON SEGMENT子句将数据复制到segment主机上的文件中。在segment主机上，COPY命令将<SEGID>替换为segment content ID，以为segment主机上的每个segment实例创建一个文件。

```
COPY LINEITEM TO PROGRAM 'cat >
/tmp/lineitem_program<SEGID>.csv' ON SEGMENT CSV;
```

本示例使用PROGRAM和ON SEGMENT子句从segment主机上的文件复制数据。从文件复制数据时，COPY命令用segment content ID替换<SEGID>。在segment主机上，每个segment实例必须有一个文件，其中文件名包含segment主机上的segment content ID。

```
COPY LINEITEM_4 FROM PROGRAM 'cat
/tmp/lineitem_program<SEGID>.csv' ON SEGMENT CSV;
```

兼容性

SQL标准中没有COPY语句。

在较早版本的Greenplum数据库中使用了以下语法，仍然被支持：

```
COPY table_name [(column_name [, ...])] FROM
{'filename' | PROGRAM 'command' |
STDIN}
[ [WITH]
[ON SEGMENT]
[BINARY]
[OIDS]
[HEADER]
[DELIMITER [ AS ] 'delimiter_character']
[NULL [ AS ] 'null string']
[ESCAPE [ AS ] 'escape' | 'OFF']
[NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
[CSV [QUOTE [ AS ] 'quote']
[FORCE NOT NULL column_name [, ...]]
[FILL MISSING FIELDS]
[[LOG ERRORS]
SEGMENT REJECT LIMIT count [ROWS | PERCENT] ]]

COPY { table_name [(column_name [, ...])] | (query) } TO
{'filename' | PROGRAM 'command' | STDOUT}
[ [WITH]
[ON SEGMENT]
[BINARY]
[OIDS]
[HEADER]
[DELIMITER [ AS ] 'delimiter_character']]
```

```
[NULL [ AS ] 'null string']
[ESCAPE [ AS ] 'escape' | 'OFF']
[CSV [QUOTE [ AS ] 'quote']
    [FORCE QUOTE column_name [, ...]] | * ]
[IGNORE EXTERNAL PARTITIONS]
```

请注意，在此语法中，`BINARY`和`CSV`被视为独立的关键字，而不是`FORMAT`选项的参数。

另见

[CREATE EXTERNAL TABLE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)

[ABORT](#)

[ALTER AGGREGATE](#)

[ALTER COLLATION](#)

[ALTER CONVERSION](#)

[ALTER DATABASE](#)

[ALTER DEFAULT PRIVILEGES](#)

[ALTER DOMAIN](#)

[ALTER EXTENSION](#)

[ALTER EXTERNAL TABLE](#)

[ALTER FOREIGN DATA WRAPPER](#)

[ALTER FOREIGN TABLE](#)

[ALTER FUNCTION](#)

[ALTER GROUP](#)

[ALTER INDEX](#)

[ALTER LANGUAGE](#)

[ALTER OPERATOR](#)

定义一个新的聚集函数

概要

argname

```
CREATE AGGREGATE name ( [ argmode ] [ ] arg_data_type [ ,  
... ] ) (  
    SFUNC = sfunc,  
    STYPE = state_data_type  
    [ , SSPACE = state_data_size ]  
    [ , FINALFUNC = ffunc ]  
    [ , FINALFUNC_EXTRA ]  
    [ , COMBINEFUNC = combinefunc ]  
    [ , SERIALFUNC = serialfunc ]  
    [ , DESERIALFUNC = deserialfunc ]  
    [ , INITCOND = initial_condition ]  
    [ , MSFUNC = msfunc ]  
    [ , MINVFUNC = minvfunc ]  
    [ , MSTYPE = mstate_data_type ]  
    [ , MSSPACE = mstate_data_size ]  
    [ , MFFINALFUNC = mffunc ]  
    [ , MFFINALFUNC_EXTRA ]  
    [ , MINITCOND = minitial_condition ]  
    [ , SORTOP = sort_operator ]  
)
```

```
CREATE AGGREGATE name ( [ [ argmode ] [ argname ]  
arg_data_type [ , ... ] ]  
    ORDER BY [ argmode ] [ argname ] arg_data_type [ ,  
... ] ) (
```

```
    SFUNC = sfunc,  
    STYPE = state_data_type  
    [ , SSPACE = state_data_size ]  
    [ , FINALFUNC = ffunc ]  
    [ , FINALFUNC_EXTRA ]  
    [ , COMBINEFUNC = combinefunc ]  
    [ , SERIALFUNC = serialfunc ]  
    [ , DESERIALFUNC = deserialfunc ]  
    [ , INITCOND = initial_condition ]  
    [ , HYPOTHETICAL ]  
)
```

or the old syntax

```
CREATE AGGREGATE name (  
    BASETYPE = base_type,  
    SFUNC = sfunc,  
    STYPE = state_data_type  
    [ , SSPACE = state_data_size ]
```



```

[ , FINALFUNC = ffunc ]
[ , FINALFUNC_EXTRA ]
[ , COMBINEFUNC = combinefunc ]
[ , SERIALFUNC = serialfunc ]
[ , DESERIALFUNC = deserialfunc ]
[ , INITCOND = initial_condition ]
[ , MSFUNC = msfunc ]
[ , MINVFUNC = minvfunc ]
[ , MSTYPE = mstate_data_type ]
[ , MSSPACE = mstate_data_size ]
[ , MFINALFUNC = mffunc ]
[ , MFINALFUNC_EXTRA ]
[ , MINITCOND = minitial_condition ]
[ , SORTOP = sort_operator ]
)

```

描述

`CREATE AGGREGATE` 定义一个新的聚合函数。Greenplum数据库中已经提供了一些基本且常用的聚合函数，例如`count`, `min`, `max`, `sum`, `avg`等。如果定义了新类型或需要尚未提供的聚合函数，则可以使用`CREATE AGGREGATE` 提供所需的功能。

如果指定了模式名称（例如`CREATE AGGREGATE myschema.myagg ...`），则将在指定的模式中创建聚合函数。否则，它将在当前模式中创建。

聚合函数由其名称和输入数据类型标识。如果同一模式中的两个聚合函数对不同的输入类型进行操作，则它们可以具有相同的名称。聚合函数的名称和输入数据类型也必须与同一模式中每个普通函数的名称和输入数据类型不同。此行为与普通函数名称的重载相同。请参阅[CREATE FUNCTION](#)。

一个简单的聚合函数由一个，两个或三个普通函数（必须是IMMUTABLE函数）组成：

- 状态转换函数 *sfunc*
- 可选的最终计算函数 *ffunc*
- 可选的合并函数 *combinefunc*

这些函数的用法如下：

```

sfunc( internal-state, next-data-values ) ---> next-
internal-state
ffunc( internal-state ) ---> aggregate-value

```

```
combinefunc( internal-state, internal-state ) ---> next-  
internal-state
```

Greenplum数据库创建一个数据类型为 `stype` 的临时变量，以保存聚合函数的当前内部状态。在每个输入行，将计算聚合参数值，并使用当前状态值和新参数值调用状态转换函数以计算新的内部状态值。处理完所有行后，将一次调用最终函数以计算合计返回值。如果没有最终函数，则按原样返回最终状态值。

您可以将 `combinefunc` 指定为优化聚合执行的方法。通过指定 `combinefunc`，可以首先在 `segment` 上并行执行聚合，然后在 `master` 上并行执行。当执行两级执行时，在 `segment` 上执行 `sfunc` 以生成部分聚合结果，而在 `master` 上执行 `combinefunc` 来聚合 `segment` 中的部分结果。如果执行单级聚合，则将所有行发送到 `master`，并将 `sfunc` 应用于行。

单级聚合和二级聚合是等效的执行策略。两种类型的聚合都可以在查询计划中实现。在实现函数 `combinefunc` 和 `sfunc` 时，必须确保在 `segment` 实例上调用 `sfunc`，然后在 `master` 上调用 `combinefunc` 产生与将所有行发送到 `master` 然后仅应用 `sfunc` 到行的单级聚合相同的结果。

聚合函数可以提供可选的初始条件，即内部状态值的初始值。它是作为 `text` 类型的值指定并存储在数据库中的，但是它必须是状态值数据类型的常量的有效外部表示形式。如果未提供，则状态值开始为 `NULL`。

如果状态转换函数声明为 `STRICT`，则不能使用 `NULL` 输入调用它。使用这种转换函数，聚合执行的行为如下。输入值为空的行将被忽略（不调用该函数，并且保留先前的状态值）。如果初始状态值为 `NULL`，则在具有所有非空输入值的第一行，第一个参数值将替换状态值，并且在具有所有非空输入值的后续行中调用转换函数。这对于实现像 `max` 这样的聚合很有用。请注意，只有当 `state_data_type` 与第一个 `input_data_type` 相同时，此行为才可用。当这些类型不同时，必须提供一个非空的初始条件或使用一个非严格的转换函数。

如果未将状态转换函数声明为 `STRICT`，则将在每个输入行无条件调用该状态转换函数，并且必须自己处理 `NULL` 输入和 `NULL` 转换值。这使聚合作者可以完全控制聚合对 `NULL` 值的处理。

如果将最终函数声明为 `STRICT`，则当结束状态值为 `NULL` 时将不调用该函数；而是将自动返回 `NULL` 结果。（这是 `STRICT` 函数的正常行为。）在任何情况下，最终函数都可以选择返回 `NULL` 值。例如，当 `avg` 的最终函数看到输入行为零时，将返回 `NULL`。

有时，将最终函数声明为不仅接受状态值，而且接受与聚合的输入值相对应的额外参数是有用的。这样做的主要原因是，如果最终函数是多态的，并且状态值的数据类型不足以固定结果类型。这些额外的参数始终以NULL的形式传递（因此，在使用FINALFUNC_EXTRA选项时，最后的函数一定不能严格），但是它们仍然是有效的参数。最终函数可以例如使用get_fn_expr_argtype来标识当前调用中的实际参数类型。

聚合可以选择支持移动聚合模式，如PostgreSQL文档中的[移动聚合模式](#)中所述。这需要指定`mfunc`, `minfunc`和`mstype`函数，以及可选的`mspace`, `mfinalfunc`,

`mfinalfunc_extra`和`minitcond`函数。除了`minvfunc`以外，这些函数的工作方式与不带`m`的相应简单聚合函数相同；它们定义了包含逆转换函数的聚合的单独实现。

参数列表中带有`ORDER BY`的语法会创建一种特殊的聚合类型，称为有序集聚合。或者，如果指定了`HYPOTHETICAL`，则创建一个假设集合。这些聚合以顺序相关的方式对排序值的组进行操作，因此指定输入排序顺序是调用的重要组成部分。而且，它们可以具有直接参数，这些参数是每个聚合仅评估一次的参数，而不是每个输入行评估一次的参数。假设集合聚合是有序集合聚合的子类，其中一些直接参数需要在数量和数据类型上与聚合参数列匹配。这允许将那些直接自变量的值作为附加的“假设”行添加到集合输入行的集合中。

有时可以通过查看索引而不是扫描每个输入行来优化单参数聚合函数（例如`min`或`max`）。如果可以如此优化此聚合，请通过指定排序运算符进行指示。基本要求是，集合必须在运算符引起的排序顺序中产生第一个元素；换一种说法：

```
SELECT agg(col) FROM tab;
```

必须等于：

```
SELECT col FROM tab ORDER BY col USING sortop LIMIT 1;
```

进一步假设聚合函数将忽略NULL输入，并且当且仅当不存在非null输入时，它才会传递NULL结果。通常，数据类型的`<`运算符是`MIN`的适当排序运算符，`>`是`MAX`的适当的排序运算符。请注意，除非指定的运算符是B树索引运算符类的“小于”或“大于”策略成员，否则优化实际上不会生效。

为了能够创建聚合函数，您必须对参数类型，状态类型和返回类型具有`USAGE`特权，并且对过渡和最终函数具有`EXECUTE`特权。

参数

name

要创建的聚合函数的名称（可以由模式指定）。

argmode

参数的模式：IN或VARIADIC。（聚合函数不支持OUT参数。）如果省略，则默认值为IN。只有最后一个参数可以标记为VARIADIC。

argname

参数的名称。当前这仅用于文档目的。如果省略，则参数没有名称。

arg_data_type

此聚合函数在其上操作的输入数据类型。要创建零参数聚合函数，请写*代替参数说明列表。（此类汇总的一个示例是count(*)。）

base_type

在CREATE AGGREGATE的旧语法中，输入数据类型由**basetype**参数指定，而不是写在聚合名称旁边。请注意，此语法仅允许一个输入参数。要使用此语法定义零参数聚合函数，请将**basetype**指定为"ANY"（不是*）。不能使用旧语法定义有序集的聚合。

sfunc

每个输入行要调用的状态转换函数的名称。对于普通的N参数聚合函数，**sfunc**必须采用N + 1个参数，第一个参数为**state_data_type**类型，其余参数与聚合的声明输入数据类型匹配。该函数必须返回**state_data_type**类型的值。该函数获取当前状态值和当前输入数据值，并返回下一个状态值。对于有序集（包括假设集）聚合，状态转换函数仅接收当前状态值和聚合参数，而不接收直接参数。否则它是相同的。

state_data_type

聚合状态值的数据类型。

state_data_size

聚合状态值的近似平均大小（以字节为单位）。如果省略此参数或将其设置为零，则基于**state_data_type**使用默认估计。优化器使用此值来估计分组聚合查询所需的内存。此参数的较大值不鼓励使用哈希聚合。

ffunc

遍历所有输入行后，用于计算合计结果的最终函数的名称。该函数必须采用**state_data_type**类型的单个参数。聚集的返回数据类型定义为该函数的返回类型。如果未指定**ffunc**，则将结束状态值用作汇总结果，返回类型为**state_data_type**。

对于有序聚集（包括假设集合），最终函数不仅接收最终状态

值，而且还接收所有直接参数的值。

如果指定了FINALFUNC_EXTRA，则除了最终状态值和任何直接参数之外，最终函数还将接收与聚合的常规（聚合）参数相对应的额外NULL值。当定义多态聚合时，这主要用于允许正确解析聚合结果类型。

serialfunc

仅当*state_data_type*为internal的聚合函数具有*serialfunc*函数时，该函数才能参与并行聚合，该函数必须将聚合状态序列化为bytea值以传输到另一个进程。该函数必须接受一个类型为internal的单个参数，并返回bytea类型。还需要相应的*deserialfunc*。

deserialfunc

将先前序列化的聚合状态反序列化回*state_data_type*。此函数必须接受bytea和internal类型的两个参数，并产生一个internal类型的结果。（注意：第二个internal参数未使用，但出于类型安全的原因是必需的。）

initial_condition

状态值的初始设置。这必须是数据类型*state_data_type*接受的字符串常量。如果未指定，则状态值开始为null。

msfunc

在移动聚合模式下要为每个输入行调用的前向状态转换函数的名称。这与常规转换函数完全相同，除了它的第一个参数和结果的类型为*mstate_data_type*，这可能与*state_data_type*不同。

minvfunc

在移动聚集模式下使用的逆状态转换函数的名称。该函数的参数和结果类型与*msfunc*相同，但是用于从当前聚合状态中删除一个值，而不是向其添加值。逆转换函数必须具有与前向状态转换函数相同的严格性属性。

mstate_data_type

使用移动聚合模式时，聚合状态值的数据类型。

mstate_data_size

使用移动聚合模式时，聚合状态值的近似平均大小（以字节为单位）。这与*state_data_size*相同。

mffunc

使用移动聚合模式时，在遍历所有输入行之后将用来计算聚合结果的最终函数的名称。它与*ffunc*相同，不同之处在于其第一个参数的类型为*mstate_data_type*，并且通过编写FINALFUNC_EXTRA指定了额外的伪参数。

由*mffunc*或*mstate_data_type*确定的聚合结果类型必须与由聚合的常规实现确定的结果类型匹配。

minitial_condition

使用移动聚合模式时，状态值的初始设置。这与*initial_condition*相同。

sort_operator

类似于MIN或MAX的聚合的关联排序运算符。这只是一个运算符

名称（可以由模式指定）。假定运算符具有与聚合相同的输入数据类型（必须是单参数常规聚合）。

HYPOTHETICAL

仅对于有序集合聚合，此标志指定将根据假设集合聚合的要求处理聚合参数：也就是说，最后几个直接参数必须与聚合(WITHIN GROUP)参数的数据类型匹配。HYPOTHETICAL标志对运行时行为没有影响，仅对数据类型的解析时解析和聚合参数的排序规则有影响。

combinefunc

组合函数的名称。这是两个参数的函数，两个参数的类型均为state_data_type。它必须返回state_data_type类型的值。组合函数采用两个过渡状态值，并返回代表组合聚合的新过渡状态值。在Greenplum数据库中，如果以分段方式计算聚合函数的结果，则对各个内部状态调用合并函数，以将它们组合为结束内部状态。

请注意，该函数在segment内的哈希聚合模式下也被调用。因此，如果您在没有合并函数的情况下调用此聚合函数，则永远不会选择哈希聚合。由于哈希聚合非常有效，因此请考虑尽可能定义一个合并函数。

注解

必须先定义用于定义新聚合函数的普通函数。请注意，在此版本的Greenplum数据库中，要求将用于创建聚合的sfinc, ffunc和Combinefunc函数定义为IMMUTABLE。

如果Greenplum数据库服务器配置参数gp_enable_multiphase_agg的值关闭，则仅执行单级聚合。

用于自定义函数的所有已编译代码（共享库文件）必须放在Greenplum数据库阵列（master和所有segment）中每个主机的相同位置。此位置也必须位于LD_LIBRARY_PATH中，以便服务器可以找到文件。

在以前的Greenplum数据库版本中，有一个有序聚合的概念。从版本6开始，可以使用以下语法将任何聚合称为有序聚合：

```
name ( arg [ , ... ] [ ORDER BY sortspec [ , ... ] ] )
```

接受ORDERED关键字是为了实现向后兼容，但会被忽略。

在以前的Greenplum数据库版本中，COMBINEFUNC选项称为PREFUNC。为了向后兼容，它仍然被接受，作为COMBINEFUNC的

同义词。

Example

以下简单示例创建一个聚合函数，该函数计算两列的总和。

在创建聚合函数之前，创建两个用作聚合函数的*sfunc*和*combinefunc*函数的函数。

该函数在聚合函数中指定为*sfunc*函数。

```
CREATE FUNCTION mysfunc_accum(numeric, numeric, numeric)
RETURNS numeric
AS 'select $1 + $2 + $3'
LANGUAGE SQL
IMMUTABLE
RETURNS NULL ON NULL INPUT;
```

该函数在聚合函数中被指定为*combinefunc*函数。

```
CREATE FUNCTION mycombine_accum(numeric, numeric )
RETURNS numeric
AS 'select $1 + $2'
LANGUAGE SQL
IMMUTABLE
RETURNS NULL ON NULL INPUT;
```

此CREATE AGGREGATE命令创建添加两列的聚合函数。

```
CREATE AGGREGATE agg_prefunc(numeric, numeric) (
SFUNC = mysfunc_accum,
STYPE = numeric,
COMBINEFUNC = mycombine_accum,
INITCOND = 0 );
```

以下命令创建一个表，添加一些行，并运行聚合函数。

```
create table t1 (a int, b int) DISTRIBUTED BY (a);
insert into t1 values
(10, 1),
(20, 2),
(30, 3);
select agg_prefunc(a, b) from t1;
```

此EXPLAIN命令显示两阶段聚合。

```
explain select agg_prefunc(a, b) from t1;

QUERY PLAN
-----
-----
Aggregate (cost=1.10..1.11 rows=1 width=32)
 -> Gather Motion 2:1 (slice1; segments: 2)
(cost=1.04..1.08 rows=1
 width=32)
 -> Aggregate (cost=1.04..1.05 rows=1 width=32)
      -> Seq Scan on t1 (cost=0.00..1.03 rows=2 width=8)
(4 rows)
```

兼容性

CREATE AGGREGATE是Greenplum数据库语言的扩展。 SQL标准不提供用户定义的聚合函数。

另见

[ALTER AGGREGATE](#) , [DROP AGGREGATE](#) , [CREATE FUNCTION](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一种新的造型。

概要

```
CREATE CAST (sourcetype AS targettype)
    WITH FUNCTION funcname (argtype [, ...])
    [AS ASSIGNMENT | AS IMPLICIT]
```

```
CREATE CAST (sourcetype AS targettype)
    WITHOUT FUNCTION
    [AS ASSIGNMENT | AS IMPLICIT]
```

```
CREATE CAST (sourcetype AS targettype)
    WITH INOUT
    [AS ASSIGNMENT | AS IMPLICIT]
```

描述

CREATE CAST 定义一个新的造型。造型指定如何在两种数据类型之间执行转换。例如，

```
SELECT CAST(42 AS float8);
```

通过调用先前指定的函数（在本例中为float8(int4)）将整数常量42转换为float8类型。如果未定义合适的造型，则转换将失败。

两种类型可以是二进制强制的，这意味着可以在不调用任何功能的情况下将这些类型相互转换。这要求相应的值使用相同的内部表示形式。例如，类型text和varchar在两个方向上都是二进制可强制的。二进制强制性不一定是对称关系。例如，在本实现中，可以无代价执行从xml到text的转换，但是相反的方向需要至少执行语法检查的函数。（两种都可以二进制强制的类型也称为二进制兼容。）

您可以使用WITH INOUT语法将转换定义为I/O转换造型。通过调用源数据类型的输出函数并将结果字符串传递给目标数据类型的输入函数来执行I/O转换造型。在许多常见情况下，此功能无需单独的造型函数即可进行转换。I/O转换造型与基于常规函数的造型相同；只是实现不同。

默认情况下，只能由显式造型请求调用造型，该请求是显式CAST(x AS *typename*)或x::: *typename*构造。

如果造型标记为AS ASSIGNMENT，则在为目标数据类型的列分配值时可以隐式调用它。例如，假设foo.f1是text类型的列，则：

```
INSERT INTO foo (f1) VALUES (42);
```

如果从integer类型到text类型的造型标记为AS ASSIGNMENT，则将被允许，否则被禁止。术语分配转换通常用于描述这种转换。

如果造型标记为AS IMPLICIT，则可以在任何上下文中隐式调用它，无论是赋值还是在表达式内部。术语隐式造型通常用于描述这种造型。例如，考虑以下查询：

```
SELECT 2 + 4.0;
```

解析器最初将常量标记为integer和numeric。系统catalog中没有integer + numeric运算符，但是有numeric + numeric运算符。如果存在从integer到numeric的造型（确实存在）并且标记为AS IMPLICIT（实际上是），则此查询成功。解析器仅应用隐式造型，并按以下方式解析查询：

```
SELECT CAST ( 2 AS numeric ) + 4.0;
```

catalog还提供了从numeric到integer的转换。如果该类型转换被标记为AS IMPLICIT（实际不是），则解析器将面临在上述解释与将numeric常量转换为integer并应用integer + integer运算符的选择之间进行选择的选择。缺乏对选择哪种选择的了解，解析器将放弃并声明模糊查询。这两个造型中只有一个隐式的，这是我们教导解析器将混合的numeric和integer表达式解析为numeric的方式。解析器对此没有内置的知识。

保守地将造型标记为隐式是明智的。过多的隐式转换路径可能会使Greenplum数据库选择令人惊讶的命令解释，或者因为存在多种可能的解释而根本无法解析命令。一个好的经验法则是，仅对于同一通用类型类别中的类型之间的信息保留转换，隐式调用造型。例如，从int2到int4的造型可以合理地是隐式的，但是从float8到int4的造型应该仅是赋值的。跨类型类别的造型（例如，text到int4）最好设为显式。

Note: 有时，出于可用性或符合标准的原因，有必要在一组类型之间提供多个隐式造型，从而导致如上所述的无法避免的歧义。解析器使

用基于类型类别和首选类型的后备启发式方法，有助于在这种情况下提供所需的行为。有关更多信息，请参见[CREATE TYPE](#)。

为了能够创建造型，您必须拥有源或目标数据类型，并对其他类型具有`USAGE`特权。要创建二进制造型，您必须是超级用户。（之所以做出此限制，是因为错误的二进制可强制造型转换很容易使服务器崩溃。）

参数

sourcetype

造型的源数据类型的名称。

targettype

造型的目标数据类型的名称。

funcname(argtype [, ...])

用于执行造型的函数。函数名称可以是模式限定的。如果不是，Greenplum数据库将在模式搜索路径中查找该函数。函数的结果数据类型必须与造型的目标类型匹配。

造型实现函数可能具有一到三个参数。第一个参数类型必须与造型的源类型相同或可以从二进制强制转换。第二个参数（如果存在）必须为`integer`类型；它接收与目标类型关联的类型修饰符；如果没有，则返回`-1`。第三个参数（如果存在）必须为`boolean`类型。如果造型是显式造型，则为`true`；否则为`false`。在某些情况下，SQL规范要求对显式和隐式造型使用不同的行为。为必须实现此类造型的函数提供此参数。不建议您以此方式设计自己的数据类型。

造型函数的返回类型必须与造型的目标类型相同或二进制强制。通常，造型必须具有不同的源和目标数据类型。但是，如果它具有一个使用多个参数的造型实现，则可以声明具有相同源类型和目标类型的造型。它用于表示系统catalog中特定于类型的长度强制函数。命名函数用于将类型的值强制为由其第二个参数给出的类型修饰符值。

当造型具有不同的源类型和目标类型以及一个函数使用多个参数时，造型将从一种类型转换为另一种类型，并在一个步骤中施加长度强制。如果没有此类条目，则强制使用类型修饰符的类型涉及两个步骤，一个步骤在数据类型之间进行转换，第二个步骤应用该修饰符。

域类型的造型当前无效。向域或从域进行强制转换使用与其基础类型关联的造型。

WITHOUT FUNCTION

指示源类型对目标类型是二进制强制的，因此不需要任何函数即可执行造型。

WITH INOUT

指示造型是I/O转换造型，通过调用源数据类型的输出函数并将结果字符串传递给目标数据类型的输入函数来执行。

AS ASSIGNMENT

指示造型可以在分配上下文中隐式调用。

AS IMPLICIT

指示在任何上下文中都可以隐式调用造型。

注解

请注意，在此版本的Greenplum数据库中，必须将用户定义的造型中使用的用户定义的函数定义为IMMUTABLE。用于自定义函数的所有已编译代码（共享库文件）必须放在Greenplum数据库阵列（master和所有segment）中每个主机的相同位置。此位置也必须位于LD_LIBRARY_PATH中，以便服务器可以找到文件。

请记住，如果您希望能够以两种方式转换类型，则需要显式地声明两种类型的造型。

通常不必在用户定义的类型和标准字符串类型

(text, varchar和char(n)以及定义为字符串类别的用户定义的类型)之间创建造型。Greenplum数据库为此提供了自动的I/O转换造型。字符串类型的自动造型被视为赋值强制造型，而字符串类型的自动造型仅是显式的。您可以通过声明自己的造型来替换自动造型来覆盖此行为，但是通常这样做的唯一原因是，如果您希望转换比标准分配或仅显式设置更容易调用。另一个可能的原因是，您希望转换的行为不同于类型的I/O函数 - 在执行此操作之前请三思。（少数内置类型的确确实具有不同的转换行为，主要是由于SQL标准的要求。）

建议您在目标数据类型之后遵循命名造型实现函数的约定，因为已命名内置造型实现函数。许多用户习惯于使用函数样式表示法（即`typename(x)`）来转换数据类型。

在两种情况下，函数调用构造被视为转换请求，而没有将其与实际函数匹配。如果函数调用`name(x)`与现有函数不完全匹配，但是`name`是数据类型的名称，并且`pg_cast`提供了从`x`类型到该类型的二进制强制转换，则该调用将被解释为二进制强制转换。Greenplum数据库例外，因此即使缺少任何函数，也可以使用函数语法调用二进制强制转换。同样，如果没有`pg_cast`条目，但强制类型转换为字符串类型或从字符串类型强制类型转换，则该调用将被解释为I/O转换造型。此异常允许使用函数语法调用I/O转换造型。

上述异常是一个例外：不能使用函数语法调用从复合类型到字符串类型的I/O转换造型，而必须使用显式的转换语法（CAST或::表示法）

编写。之所以存在该异常，是因为在引入自动提供的I/O转换造型后，当您打算使用函数或列引用时，发现很容易意外调用此类造型。

示例

要使用函数int4(bigint)创建从类型bigint到类型int4的赋值造型（此造型已在系统中预先定义）：

```
CREATE CAST (bigint AS int4) WITH FUNCTION int4(bigint) AS  
ASSIGNMENT;
```

参数

CREATE CAST命令符合SQL标准，但SQL不为二进制可强制类型或实现函数的额外参数提供准备。AS IMPLICIT也是Greenplum数据库扩展。

另见

[CREATE FUNCTION](#) , [CREATE TYPE](#) , [DROP CAST](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

使用指定的操作系统语言环境或通过复制现有的排序规则来定义新的排序规则。

概要

```
CREATE COLLATION name (
    [ LOCALE = locale, ]
    [ LC_COLLATE = lc_collate, ]
    [ LC_CTYPE = lc_ctype ])
```

```
CREATE COLLATION name FROM existing_collation
```

描述

为了能够创建排序规则，您必须在目标模式上具有CREATE特权。

参数

name

排序规则的名称。排序规则名称可以由模式指定。如果不是，则在当前模式中定义排序规则。排序规则名称在该模式中必须唯一。（系统catalog可以包含与其他编码相同名称的排序规则，但是如果数据库编码不匹配，则将忽略这些排序规则。）

locale

这是一次设置LC_COLLATE和LC_CTYPE的快捷方式。如果指定此选项，则不能指定这些参数中的任何一个。

lc_collate

将指定的操作系统语言环境用于LC_COLLATE语言环境类别。语言环境必须适用于当前数据库编码。（有关详细规则，请参见[CREATE DATABASE](#)。）

lc_ctype

将指定的操作系统语言环境用于LC_CTYPE语言环境类别。语言环境必须适用于当前数据库编码。（有关详细规则，请参见[CREATE DATABASE](#)。）

existing_collation

要复制的现有排序规则的名称。新的归类将具有与现有归类相

同的属性，但是它将是一个独立的对象。

注解

为了能够创建排序规则，您必须在目标模式上具有CREATE特权。

使用DROP COLLATION删除用户定义的排序规则。

有关Greenplum数据库中归类支持的更多信息，请参见PostgreSQL文档中的[归类支持](#)。

示例

要从操作系统语言环境fr_FR.utf8创建排序规则（假设当前数据库编码为UTF8）： UTF8) :

```
CREATE COLLATION french (LOCALE = 'fr_FR.utf8');
```

要从现有排序规则创建排序规则：

```
CREATE COLLATION german FROM "de_DE";
```

能够在应用程序中使用独立于操作系统的排序规则名称可能很方便。

兼容性

SQL标准中有一个CREATE COLLATION语句，但是它仅限于复制现有的排序规则。创建新排序规则的语法是Greenplum数据库扩展。

另见

[ALTER COLLATION, DROP COLLATION](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义新的编码转换。

概要

```
CREATE [DEFAULT] CONVERSION name FOR source_encoding TO
dest_encoding FROM funcname
```

描述

`CREATE CONVERSION` 定义了字符集编码之间的新转换。 转换名称可以在转换函数中使用，以指定特定的编码转换。 另外，标记为`DEFAULT`的转换可用于客户端和服务器之间的自动编码转换。 为此，必须定义从编码A到B和从编码B到A的两次转换。

要创建转换，您必须在函数上具有`EXECUTE`特权，并在目标模式上具有`CREATE`特权。

参数

DEFAULT

指示此转换是此特定源到目标编码的默认转换。 模式中的编码对应该只有一种默认编码。

name

转换的名称。 转换名称可以由模式指定。 如果不是，则在当前模式中定义转换。 转换名称在模式中必须唯一。

source_encoding

源编码名称。

dest_encoding

目标编码名称。

funcname

用于执行转换的函数。 函数名称可以由模式指定。 如果不是，将在路径中查找该函数。 该函数必须具有以下签名

```
conv_proc(
    integer, -- source encoding ID
    integer, -- destination encoding ID
```

```
    cstring, -- source string (null terminated C
    string)
        internal, -- destination (fill with a null
        terminated C string)
            integer -- source string length
    ) RETURNS void;
```

注解

请注意，在此版本的Greenplum数据库中，用户定义转换中使用的用户定义函数必须定义为IMMUTABLE。用于自定义函数的所有已编译代码（共享库文件）必须放在Greenplum数据库阵列（master和所有segment）中每个主机的相同位置。此位置也必须位于LD_LIBRARY_PATH中，以便服务器可以找到文件。

示例

要使用myfunc创建从UTF8编码到LATIN1的转换，请执行以下操作：

```
CREATE CONVERSION myconv FOR 'UTF8' TO 'LATIN1' FROM
myfunc;
```

兼容性

SQL标准中没有CREATE CONVERSION语句，但是在目的和语法上有一个非常相似的CREATE TRANSLATION语句。

另见

[ALTER CONVERSION](#) , [CREATE FUNCTION](#) , [DROP CONVERSION](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

创建一个新的数据库。

概要

```
CREATE DATABASE name [ [WITH] [OWNER [=] user_name]
[TEMPLATE [=] template]
[ENCODING [=] encoding]
[LC_COLLATE [=] lc_collate]
[LC_CTYPE [=] lc_ctype]
[TABLESPACE [=] tablespace]
[CONNECTION LIMIT [=] connlimit ] ]
```

描述

`CREATE DATABASE` 创建一个新的数据库。 要创建数据库，您必须是超级用户或具有特殊的`CREATEDB`特权。

默认情况下，创建者成为新数据库的所有者。 超级用户可以使用`OWNER`子句创建其他用户拥有的数据库。 他们甚至可以创建没有特殊特权的用户所拥有的数据库。 具有`CREATEDB`特权的非超级用户只能创建自己拥有的数据库。

默认情况下，将通过克隆标准系统数据库`template1`来创建新数据库。 可以通过写入`TEMPLATE name`来指定其他模板。 特别是，通过编写`TEMPLATE template0`，您可以创建一个仅包含Greenplum数据库预定义的标准对象的干净数据库。 如果您希望避免复制任何可能已添加到`template1`的本地安装对象，则此功能很有用。

参数

name

要创建的数据库的名称。

*user_name*拥有新数据库的用户的名称，或者使用默认所有者（行命令的用户）`DEFAULT`。*template*创建新数据库的模板的名称，或者`DEFAULT`使用默认模板的模

板 (*template1*)。
encoding

在新数据库中使用的字符集编码。指定字符串常量（例如 'SQL_ASCII'），整数编码数字或DEFAULT以使用默认编码。有关更多信息，请参见[字符集支持](#)。

lc_collate

在新数据库中使用的排序规则 (LC_COLLATE)。这会影响应用于字符串的排序顺序，例如在使用ORDER BY的查询中，以及在文本列的索引中使用的顺序。默认设置是使用模板数据库的排序规则。有关其他限制，请参见“注释”部分。

lc_ctype

在新数据库中使用的字符分类 (LC_CTYPE)。这会影响字符的分类，例如下，上和数字。默认值是使用模板数据库的字符分类。有关其他限制，请参见下文。

tablespace

将与新数据库关联的表空间的名称，或者为DEFAULT以使用模板数据库的表空间。该表空间将是用于在此数据库中创建的对象的默认表空间。

connlimit

可能的最大并发连接数。默认值-1表示没有限制。

注解

CREATE DATABASE不能在事务块内执行。

通过将数据库名称指定为模板来复制数据库时，在复制数据库过程中，不能将其他会话连接到模板数据库。模板数据库的新连接被锁定，直到CREATE DATABASE完成。

对超级用户没有强制CONNECTION LIMIT。

为新数据库指定的字符集编码必须与所选的语言环境设置 (LC_COLLATE和LC_CTYPE) 兼容。如果语言环境为C (或等价的POSIX)，则允许所有编码，但是对于其他语言环境设置，只有一种编码可以正常工作。CREATE DATABASE将允许超级用户指定SQL_ASCII编码，而与语言环境设置无关，但是不建议使用此选项，并且如果与语言环境不兼容的数据存储在数据库中，则可能会导致字符串函数出现异常。

编码和语言环境设置必须与模板数据库的设置匹配，除非将*template0*用作模板。这是因为COLLATE和CTYPE影响索引的顺序，因此从模板数据库复制的任何索引在具有不同设置的新数据库中都将无效。但是，已知*template0*不包含任何会受到影响的数据或

索引。

示例

要创建一个新的数据库：

```
CREATE DATABASE gpdb;
```

要创建用户salesapp拥有的数据库sales并使用salesspace的默认表空间：

```
CREATE DATABASE sales OWNER salesapp TABLESPACE salesspace;
```

要创建支持ISO-8859-1字符集的数据库music：

```
CREATE DATABASE music ENCODING 'LATIN1' TEMPLATE template0;
```

在此示例中，仅当template1的编码不是ISO-8859-1时才需要TEMPLATE template0子句。请注意，更改编码可能还需要选择新的LC_COLLATE和LC_CTYPE设置。

兼容性

SQL标准中没有CREATE DATABASE语句。数据库等效于目录，目录的创建是实现定义的。

另见

[ALTER DATABASE , DROP DATABASE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一个新域。

概要

```
CREATE DOMAIN name [ AS ] data_type [ DEFAULT expression ]
    [ COLLATE collation ]
    [ CONSTRAINT constraint_name
    | NOT NULL | NULL
    | CHECK (expression) [...] ]
```

描述

CREATE DOMAIN创建一个新域。 域本质上是具有可选约束（对允许的值集的约束）的数据类型。 定义域的用户将成为其所有者。 域名在其模式中存在的数据类型和域之间必须唯一。

如果提供了模式名称（例如CREATE DOMAIN *myschema.mydomain ...*），则将在指定的模式中创建域。 否则，它将在当前模式中创建。

域对于将字段上的公共约束抽象到单个位置进行维护很有用。 例如，几个表可能包含电子邮件地址列，所有这些列都需要相同的CHECK约束来验证地址语法。 定义域比为每个具有电子邮件列的表设置列约束要容易得多。

为了能够创建域，您必须对基础类型具有USAGE特权。

参数

name

要创建的域的名称（可以由模式指定）。

data_type

域的基础数据类型。这可能包括数组说明符。

DEFAULT expression

为域数据类型的列指定默认值。该值是任何无变量的表达式（但不允许子查询）。默认表达式的数据类型必须与域的数据



类型匹配。如果未指定默认值，则默认值为空值。默认表达式将在未为列指定值的任何插入操作中使用。如果为特定列定义了默认值，它将覆盖与该域关联的任何默认值。反过来，域默认值将覆盖与基础数据类型关联的任何默认值。

COLLATE *collation*

域的可选归类。如果未指定排序规则，则使用基础数据类型的默认排序规则。如果指定了COLLATE，则基础类型必须可整理。

CONSTRAINT *constraint_name*

约束的可选名称。如果未指定，系统将生成一个名称。

NOT NULL

通常可以防止该域的值为null。但是，如果为该域分配了已变为空的匹配域类型仍然是可行的，例如，通过左连接或者如`INSERT INTO tab (domcol) VALUES ((SELECT domcol FROM tab WHERE false))`之类的命令。

NULL

该域的值允许为空。这是默认值。此子句仅用于与非标准SQL数据库兼容。不建议在新应用中使用它。

CHECK (*expression*)

CHECK子句指定完整性约束或测试域必须满足的值。每个约束必须是一个产生布尔结果的表达式。它应该使用关键字VALUE来引用要测试的值。当前，CHECK表达式不能包含子查询，也不能引用VALUE以外的变量。

示例

创建us_zip_code数据类型。使用正则表达式测试来验证该值看起来像是有效的美国邮政编码。

```
CREATE DOMAIN us_zip_code AS TEXT CHECK
( VALUE ~ '^\d{5}$' OR VALUE ~ '^\d{5}-\d{4}$' );
```

兼容性

CREATE DOMAIN符合SQL标准。

另见

[ALTER DOMAIN](#) , [DROP DOMAIN](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

Registers an extension in a Greenplum database.

概要

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
    [ WITH ] [ SCHEMA schema_name ]
    [ VERSION version ]
    [ FROM old_version ]
    [ CASCADE ]
```

描述

`CREATE EXTENSION`将新的扩展名加载到当前数据库中。不可以已加载具有相同名称的扩展名。

加载扩展实质上等于运行扩展脚本文件。该脚本通常创建新的SQL对象，例如函数，数据类型，运算符和索引支持方法。`CREATE EXTENSION`命令还记录所有已创建对象的标识，因此，如果执行`DROP EXTENSION`，则可以再次删除它们。

加载扩展需要与创建组件扩展对象相同的特权。对于大多数扩展，这意味着需要超级用户或数据库所有者特权。运行`CREATE EXTENSION`的用户将成为扩展的所有者，以进行以后的权限检查，以及该扩展脚本创建的任何对象的所有者。

参数

IF NOT EXISTS

如果已经存在具有相同名称的扩展名，请不要报错。在这种情况下发出通知。无法保证现有扩展与安装的扩展类似。

extension_name

要安装的扩展名。该名称在数据库中必须唯一。根据扩展控制文件`SHAREDIR/extension/extension_name.control`中的详细信息创建扩展。



SHAREDIR是安装共享数据目录，例如`/usr/local/greenplum-db/share/postgresql`。命令`pg_config --sharedir`显示目录。

SCHEMA *schema_name*

在其中安装扩展对象的模式名称。假定扩展名允许其内容重定位。命名模式必须已经存在。如果未指定，并且扩展控制文件未指定模式，则使用当前的默认模式创建对象。

如果扩展名在其控制文件中指定了模式参数，则无法使用**SCHEMA**子句覆盖该模式。通常，如果给定了**SCHEMA**子句，并且与扩展模式参数冲突，则会引发错误。但是，如果还给出了**CASCADE**子句，则*schema_name*在冲突时将被忽略。给定的*schema_name*用于安装所有需要的扩展，这些扩展在其控制文件中未指定架构。

扩展本身不在任何模式内。扩展名具有非限定名称，这些名称在数据库中必须唯一。但是属于扩展名的对象可以在模式内。

VERSION *version*

要安装的扩展程序的版本。可以将其写为标识符或字符串文字。默认版本是扩展控制文件中指定的值。

FROM *old_version*

仅在尝试安装扩展来替换旧模块（该扩展是未打包到扩展中的对象集合）的扩展时，才指定**FROM old_version**。如果指定，则**CREATE EXTENSION**运行替代安装脚本，该脚本将现有对象吸收到扩展中，而不是创建新对象。确保**SCHEMA**子句指定了包含这些预先存在的对象的模式。

用于*old_version*的值由扩展作者确定，如果可以将多个旧版本的模块升级为扩展，则该值可能会有所不同。对于9.1之前的PostgreSQL提供的标准附加模块，在将模块更新为扩展样式时，请为*old_version*指定**unpackaged**。

CASCADE

尚未安装自动安装相关扩展。递归检查相关的扩展，并且这些从属扩展也将自动安装。如果指定了**SCHEMA**子句，则该模式适用于扩展和已安装的所有从属扩展。指定的其他选项不适用于自动安装的从属扩展。特别是，在安装从属扩展时始终选择默认版本。

注解

当前可用于加载的扩展名可以从[pg_available_extensions](#)或[pg_available_extension_versions](#)系统视图中识别。

在使用CREATE EXTENSION将扩展加载到数据库之前，必须安装支持的扩展文件，包括扩展控制文件和至少一个SQL脚本文件。支持文件必须安装在所有Greenplum数据库主机上的相同位置。有关创建新扩展的信息，请参见PostgreSQL有关[将相关对象打包到扩展中](#)的信息。

兼容性

CREATE EXTENSION是Greenplum数据库扩展。

另见

[ALTER EXTENSION, DROP EXTENSION](#)

Parent topic: [SQL Command Reference](#)

TABLE

Greenplum数据库® 6.0文档

定义一个新的外部表。

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

概要

```

CREATE [READABLE] EXTERNAL [TEMPORARY | TEMP] TABLE

  ( column_name data_type [, ...] | LIKE other_table )
  LOCATION ('file://seghost[:port]/path/file' [, ...])
  |
  ('gpfdist://filehost[:port]/file_pattern[#transform=trans_na
  [, ...]
  |
  ('gpfdists://filehost[:port]/file_pattern[#transform=trans_na
  [, ...])
  |
  ('pxf://path-to-data?
PROFILE=profile_name [&SERVER=server_name][&custom-
option=value[...]]')
  | ('s3://S3_endpoint[:port]/bucket_name/[S3_prefix]
[region=S3-region] [config=config_file]')
  [ON MASTER]
  FORMAT 'TEXT'
  [ ( [HEADER]
    [DELIMITER [AS] 'delimiter' | 'OFF']
    [NULL [AS] 'null string']
    [ESCAPE [AS] 'escape' | 'OFF']
    [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
    [FILL MISSING FIELDS] )]
  | 'CSV'
  [ ( [HEADER]
    [QUOTE [AS] 'quote']
    [DELIMITER [AS] 'delimiter']
    [NULL [AS] 'null string']
    [FORCE NOT NULL column [, ...]]
    [ESCAPE [AS] 'escape']
    [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
    [FILL MISSING FIELDS] )]
  | 'CUSTOM' (Formatter=<formatter_specifications>
  [ ENCODING 'encoding' ]
  [ [LOG ERRORS] SEGMENT REJECT LIMIT count
  [ROWS | PERCENT] ]
  |
  CREATE [READABLE] EXTERNAL WEB [TEMPORARY | TEMP] TABLE

  ( column_name data_type [, ...] | LIKE other_table )
  LOCATION ('http://webhost[:port]/path/file' [, ...])
  
```

```

| EXECUTE 'command' [ ON ALL
|                   | MASTER
|                   | number_of_segments
|                   | HOST ['segment_hostname']
|                   | SEGMENT segment_id ]
FORMAT 'TEXT'
[ ( [HEADER]
    [DELIMITER [AS] 'delimiter' | 'OFF']
    [NULL [AS] 'null string']
    [ESCAPE [AS] 'escape' | 'OFF']
    [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
    [FILL MISSING FIELDS] )
| 'CSV'
[ ( [HEADER]
    [QUOTE [AS] 'quote']
    [DELIMITER [AS] 'delimiter']
    [NULL [AS] 'null string']
    [FORCE NOT NULL column [, ...]]
    [ESCAPE [AS] 'escape']
    [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
    [FILL MISSING FIELDS] )
| 'CUSTOM' (Formatter=<formatter
specifications>)
[ ENCODING 'encoding' ]
[ [LOG ERRORS] SEGMENT REJECT LIMIT count
[ROWS | PERCENT] ]

CREATE WRITABLE EXTERNAL [ TEMPORARY | TEMP ] TABLE
table_name
( column_name data_type [, ...] | LIKE other_table )
LOCATION('gpfdist://outputhost[:port]/filename[#transfor
[ , ...])
|
('gpfdists://outputhost[:port]/file_pattern[#transform=trans_
[ , ...])
FORMAT 'TEXT'
[ ( [DELIMITER [AS] 'delimiter']
    [NULL [AS] 'null string']
    [ESCAPE [AS] 'escape' | 'OFF'] )
| 'CSV'
[ ([QUOTE [AS] 'quote']
    [DELIMITER [AS] 'delimiter']
    [NULL [AS] 'null string']
    [FORCE QUOTE column [, ...]] | * ]
    [ESCAPE [AS] 'escape'] )
| 'CUSTOM' (Formatter=<formatter
specifications>)
[ ENCODING 'write_encoding' ]
[ DISTRIBUTED BY ({column [opclass]}, [ ... ]) |
DISTRIBUTED RANDOMLY ]

CREATE WRITABLE EXTERNAL [ TEMPORARY | TEMP ] TABLE
table_name
( column_name data_type [, ...] | LIKE other_table )
LOCATION('s3://S3 endpoint[:port]/bucket name/[S3 prefix')

```

```

[region=S3-region] [config=config_file]')
[ON MASTER]
FORMAT 'TEXT'
[ ( [DELIMITER [AS] 'delimiter']
[NULL [AS] 'null string']
[ESCAPE [AS] 'escape' | 'OFF' ] ) ]
| 'CSV'
[ ([QUOTE [AS] 'quote']
[DELIMITER [AS] 'delimiter']
[NULL [AS] 'null string']
[FORCE QUOTE column [, ...]] | * ]
[ESCAPE [AS] 'escape' ] )]

CREATE WRITABLE EXTERNAL WEB [TEMPORARY | TEMP] TABLE

( column_name data_type [, ...] | LIKE other_table )
EXECUTE 'command' [ON ALL]
FORMAT 'TEXT'
[ ( [DELIMITER [AS] 'delimiter']
[NULL [AS] 'null string']
[ESCAPE [AS] 'escape' | 'OFF' ] ) ]
| 'CSV'
[ ([QUOTE [AS] 'quote']
[DELIMITER [AS] 'delimiter']
[NULL [AS] 'null string']
[FORCE QUOTE column [, ...]] | * ]
[ESCAPE [AS] 'escape' ] ) ]
| 'CUSTOM' (Formatter=<formatter
specifications>)
[ ENCODING 'write_encoding' ]
[ DISTRIBUTED BY ({column [opclass]}, [ ... ]) |
DISTRIBUTED RANDOMLY ]

```

描述

有关外部表的详细信息，请参阅Greenplum数据库管理员指南中的“使用外部表”。

CREATE EXTERNAL TABLE或CREATE EXTERNAL WEB TABLE在Greenplum数据库中创建一个新的可读外部表定义。 可读的外部表通常用于快速并行数据加载。 定义外部表后，您可以使用SQL命令直接（或并行）查询其数据。 例如，您可以选择，联接或排序外部表数据。 您也可以为外部表创建视图。 可读外部表上不允许DML操作（UPDATE, INSERT, DELETE或TRUNCATE），并且您不能在可读外部表上创建索引。

CREATE WRITABLE EXTERNAL TABLE或CREATE WRITABLE EXTERNAL WEB TABLE在Greenplum数据库中创建新的可写外部表定义。 可写的外部表通常用于将数据库中的数据卸载到一组文件或命名

管道中。 可写的外部Web表也可以用于将数据输出到可执行程序。 可写的外部表也可以用作Greenplum并行MapReduce计算的输出目标。 一旦定义了可写外部表，就可以从数据库表中选择数据并将其插入可写外部表中。 可写的外部表仅允许INSERT操作 – 不允许执行SELECT, UPDATE, DELETE或TRUNCATE。

常规外部表和外部Web表之间的主要区别在于它们的数据源。 常规可读的外部表访问静态平面文件，而外部Web表访问动态数据源 - 在Web服务器上或通过执行OS命令或脚本。

参数

READABLE | WRITABLE

指定外部表的类型，默认为可读。 可读的外部表用于将数据加载到Greenplum数据库中。 可写的外部表用于卸载数据。

WEB

在Greenplum数据库中创建一个可读或可写的外部Web表定义。 可读的外部Web表有两种形式 - 通过http://协议访问文件的表或通过执行OS命令访问数据的表。 可写的外部Web表将数据输出到可执行程序，该程序可以接受输入的数据流。 外部Web表在查询执行期间不可重新扫描。

s3协议不支持外部Web表。 但是，您可以创建一个外部Web表，该表执行第三方工具以从S3读取数据或直接向S3写入数据。

TEMPORARY | TEMP

如果指定，则在Greenplum数据库中创建一个临时的可读或可写外部表定义。 临时外部表存在于特殊模式中。 创建表时，无法指定模式名称。 在会话结束时，将自动删除临时外部表。

临时表存在时，当前会话将看不到具有相同名称的已知永久表，除非您引用具有其模式限定名称的永久表。

table_name

新外部表的名称。

column_name

要在外部表定义中创建的列的名称。 与常规表不同，外部表没有列约束或默认值，因此请不要指定它们。

LIKE *other_table*

LIKE子句指定一个表，新的外部表将从该表中自动复制所有列名称，数据类型和Greenplum分配策略。 如果原始表指定了任何列约束或默认列值，则不会将这些约束复制到新的外部表定义中。

data_type

列的数据类型。

LOCATION ('*protocol://[host[:port]]/path/file'* [, ...])

如果使用pxf协议访问外部数据源，请参阅[PXF创建外部表](#)文档，以获取有关pxf协议LOCATION子句语法的详细信息。

如果使用s3协议读取或写入S3，请参阅[关于S3协议URL](#)，以获取有关s3协议LOCATION子句语法的其他信息。

对于可读的外部表，指定用于填充外部表或Web表的外部数据源的URI。常规的可读外部表允许使用gpfdist或file协议。外部Web表允许使用http协议。如果省略port，则将http和gpfdist协议的端口假定为8080。如果使用gpfdist协议，则path是相对于gpfdist从中提供文件的目录（启动gpfdist程序时指定的目录）。另外，gpfdist可以使用通配符或其他C样式模式匹配（例如，空格字符为[[:space:]]) 来表示目录中的多个文件。例如：

```
'gpfdist://filehost:8081/*'  
'gpfdist://masterhost/my_load_file'  
'file://seghost1/dbfast1/external/myfile.txt'  
'http://intranet.example.com/finance/expenses.csv'
```

对于可写的外部表，指定gpfdist进程或S3协议的URI位置，该进程将收集Greenplum segment中的数据输出并将其写入一个或多个命名文件。对于gpfdist，该路径是相对于gpfdist从中提供文件的目录（启动gpfdist程序时指定的目录）的。如果列出了多个gpfdist位置，则发送数据的segment将在可用的输出位置之间平均分配。例如：

```
'gpfdist://outpuhost:8081/data1.out',  
'gpfdist://outpuhost:8081/data2.out'
```

在上面的示例中列出了两个gpfdist位置的情况下，一半的segment会将其输出数据发送到data1.out文件，另一半发送到data2.out文件。

使用#transform=trans_name选项，可以指定在加载或提取数据时要应用的转换。*trans_name*是您在运行gpfdist实用程序时指定的YAML配置文件中转换的名称。有关指定转换的信息，请参阅[Greenplum实用程序指南中的gpfdist](#)。

ON MASTER

将所有与表相关的操作限制为Greenplum master。仅允许使用s3或自定义协议创建的可读写外部表。

gpfdist, gpfdists, pxf和file协议不支持ON MASTER。Note: 在使用ON MASTER子句读取或写入创建的外部表时，请注意潜在的资源影响。当您仅将表操作限制为Greenplum master时，可能会遇到性能问题。

EXECUTE 'command' [ON ...]

只允许可读的外部Web表或可写的外部表。对于可读的外部Web表，指定要由segment实例执行的OS命令。

该*command*可以是单个OS命令或脚本。ON子句用于指定哪些segment实例将执行给定命令。

- 默认为ON ALL。该命令将由Greenplum数据库系统中所有segment主机上的每个活动primary实例执行。如果命令执行脚本，则该脚本必须位于所有segment主机上的相同位置，并且可由Greenplum超级用户（gpadmin）执行。
- ON MASTER仅在master主机上运行命令。
Note: 指定ON MASTER子句时，外部Web表不支持日志记录。
- ON *number*表示将按指定的segment数执行命令。特定的segment是在运行时由Greenplum数据库系统随机选择的。如果命令执行脚本，则该脚本必须位于所有segment主机上的相同位置，并且可由Greenplum超级用户（gpadmin）执行。
- HOST表示该命令将由每个segment主机上的一个segment执行（每个segment主机一次），而不管每个主机的活动segment实例数量如何。
- HOST *segment_hostname*表示命令将由指定的segment主机上的所有活动primary实例执行。
- SEGMENT *segment_id*表示命令只能由指定的segment执行一次。您可以通过查看系统catalog表[gp_segment_configuration](#) 中的content编号来确定segment实例的ID。Greenplum数据库主数据库的content ID始终为-1。

对于可写外部表，必须准备在EXECUTE子句中指定的*command*以将数据传递到其中。由于所有要发送数据的segment都会将其输出写入指定的命令或程序，因此ON子句的唯一可用选项是ON ALL。

FORMAT 'TEXT | CSV' (*options*)

当FORMAT子句标识定界文本（TEXT）或逗号分隔值（csv）格式时，格式设置选项与PostgreSQL [COPY](#)命令可用的格式设置选项相似。如果文件中的数据不使用默认的列定界符，转义符，空字符串等，则必须指定其他格式选项，以便Greenplum数据库可以正确读取外部文件中的数据。有关使用自定义格式的信息，请参阅Greenplum数据库管理员指南中的“加载和卸载数据”。

如果使用pxf协议访问外部数据源，请参阅[PXF创建外部表](#)文档，以获取有关pxf协议FORMAT子句语法的详细信息。

FORMAT 'CUSTOM' (*formatter= formatter_specification*)

指定自定义数据格式。*formatter_specification*指定用于格式化数据的函数，后跟格式化函数的逗号分隔参数。格式化程序规范的长度（包括Formatter=的字符串）最多可达到约50K字

节。

如果使用pxf协议访问外部数据源，请参阅[PXF创建外部表](#)文档，以获取有关pxf协议FORMAT子句语法的详细信息。

有关使用自定义格式的一般信息，请参阅[Greenplum数据库管理员指南](#)中的“加载和卸载数据”。

DELIMITER

指定一个ASCII字符，用于分隔数据的每一行（行）中的列。默认认为TEXT模式下的制表符，而csv模式下为逗号。在用于可读外部表的TEXT模式下，对于将非结构化数据加载到单列表的特殊用例，可以将定界符设置为OFF。

对于s3协议，定界符不能为换行符（\n）或回车符（\r）。

NULL

指定表示NULL值的字符串。在TEXT模式下，默认值为\N（反斜杠-N），在csv模式下，默认值为无引号的空值。对于不希望将NULL值与空字符串区分开的情况，即使在TEXT模式下，您也可能更喜欢空字符串。使用外部表和Web表时，与该字符串匹配的任何数据项都将被视为NULL值。

作为text格式的示例，此FORMAT子句可用于指定两个单引号（'）的字符串为NULL值。

```
FORMAT 'text' (delimiter ',' null ''''')
```

ESCAPE

指定用于C转义序列的单个字符（例如\n, \t, \100等）和转义可能用作行或列定界符的数据字符。确保选择一个在实际列数据中未使用的转义字符。对于文本格式的文件，默认转义字符是\（反斜杠），对于csv格式的文件，默认转义字符是"（双引号），但是可以指定另一个字符来表示转义。也可以通过将值'OFF'指定为转义值在文本格式化文件中禁用文本转义。这对于诸如文本格式的Web日志数据之类的数据非常有用，该数据具有许多嵌入式反斜杠，而这些反斜杠并非旨在转义。

NEWLINE

指定数据文件中使用的换行符 – LF（换行，0x0A），CR（回车，0x0D）或CRLF（回车加换行，0x0D 0x0A）。如果未指定，Greenplum数据库segment将通过查看接收到的第一行数据并使用遇到的第一个换行符类型来检测换行符类型。

HEADER

对于可读的外部表，指定数据文件中的第一行是标题行（包含表列的名称），并且不应包括表的数据在内。如果使用多个数据源文件，则所有文件都必须具有标题行。

对于s3协议，标题行中的列名称不能包含换行符（\n）或回车符（\r）。

pxf协议不支持HEADER格式化选项。

QUOTE

指定CSV模式的引号字符。默认值为双引号（"）。

FORCE NOT NULL

在CSV模式下，处理每个指定的列就好像被引用了一样，因此不是NULL值。对于CSV模式下的默认空字符串（两个定界符之间都不存在），这将导致缺失值被评估为零长度字符串。

FORCE QUOTE

在可写外部表的CSV模式下，强制将引号用于每个指定列中的所有非NULL值。如果指定*，则在所有列中都引用非NULL值。
NULL输出从不引用。

FILL MISSING FIELDS

对于可读外部表，在TEXT和CSV模式下，指定FILL MISSING FIELDS时，如果一行数据的行或行末尾缺少数据字段，则将缺少的尾随字段值设置为NULL（而不是报告错误）。空白行，具有NOT NULL约束的字段以及行尾的定界符仍然会报告错误。

ENCODING '*encoding*'

用于外部表的字符集编码。指定字符串常量（例如'SQL_ASCII'），整数编码数字或DEFAULT以使用默认的客户端编码。请参见[字符集支持](#)。

LOG ERRORS

这是一个可选的子句，可以在SEGMENT REJECT LIMIT子句之前记录有关具有格式错误的行的信息。错误日志信息存储在内部，并可以通过Greenplum数据库内置的SQL函数gp_read_error_log()进行访问。

请参阅[注解](#)以获取有关错误日志信息的信息以及用于查看和管理错误日志信息的内置函数。

SEGMENT REJECT LIMIT *count* [ROWS | PERCENT]

在单行错误隔离模式下运行COPY FROM操作。如果输入行存在格式错误，只要在加载操作期间未在任何Greenplum segment实例上达到拒绝限制计数，它们将被丢弃。拒绝限制计数可以指定为行数（默认）或总行数的百分比（1-100）。如果使用PERCENT，则只有在处理了参

数gp_reject_percent_threshold指定的行数之后，每个segment才开始计算错误行百分比。

gp_reject_percent_threshold的默认值为300行。约束错误（例如违反NOT NULL, CHECK或UNIQUE约束）仍将在“全有或全无”输入模式下处理。如果未达到限制，则将加载所有正确的行，并丢弃所有错误行。

Note: 读取外部表时，如果未首先触发SEGMENT REJECT LIMIT或未指定SEGMENT REJECT LIMIT，则Greenplum数据库会限制可能包含格式错误的初始行数。如果前1000行被拒绝，则COPY操作将停止并回滚。

可以使用Greenplum数据库服务器配置参数gp_initial_bad_row_limit更改初始拒绝行数的限制。

有关参数的信息，请参阅[服务器配置参数](#)。

DISTRIBUTED BY ({*column* [*opclass*]}, [...])
DISTRIBUTED RANDOMLY

用于声明可写外部表的Greenplum数据库分发策略。默认情况下，可写外部表是随机分布的。如果要从中导出数据的源表具有哈希分发策略，并且为可写外部表定义了相同的分发键列和运算符类*opclass*，则将消除在interconnect上移动行的需要，从而提高了卸载性能。当您发出诸如`INSERT INTO wex_table SELECT * FROM source_table`之类的卸载命令时，如果两个表具有相同的哈希分配策略，则可以将这些已卸载的行直接从segment发送到输出位置。

示例

在端口8081的后台启动gpfdist文件服务器程序，以服务目录/var/data/staging中的文件：

```
gpfdist -p 8081 -d /var/data/staging -l /home/gpadmin/log &
```

使用gpfdist协议和gpfdist目录中找到的所有文本格式文件 (*.txt) 创建一个名为ext_customer的可读外部表。使用竖线 (|) 作为列定界符，并使用空白作为NULL来格式化文件。还要以单行错误隔离模式访问外部表：

```
CREATE EXTERNAL TABLE ext_customer
(id int, name text, sponsor text)
LOCATION ( 'gpfdist://filehost:8081/*.txt' )
FORMAT 'TEXT' ( DELIMITER '|' NULL '' )
LOG ERRORS SEGMENT REJECT LIMIT 5;
```

创建与上述相同的可读外部表定义，但使用CSV格式的文件：

```
CREATE EXTERNAL TABLE ext_customer
(id int, name text, sponsor text)
LOCATION ( 'gpfdist://filehost:8081/*.csv' )
FORMAT 'CSV' ( DELIMITER ',' );
```

使用file协议和一些带有标题行的CSV格式的文件，创建一个名为ext_expenses的可读外部表：

```
CREATE EXTERNAL TABLE ext_expenses (name text, date date,
amount float4, category text, description text)
```

```

LOCATION (
  'file://segghost1/dbfast/external/expenses1.csv',
  'file://segghost1/dbfast/external/expenses2.csv',
  'file://segghost2/dbfast/external/expenses3.csv',
  'file://segghost2/dbfast/external/expenses4.csv',
  'file://segghost3/dbfast/external/expenses5.csv',
  'file://segghost3/dbfast/external/expenses6.csv'
)
FORMAT 'CSV' ( HEADER );

```

创建一个可读的外部Web表，每个表主机执行一次脚本：

```

CREATE EXTERNAL WEB TABLE log_output (linenum int, message
text) EXECUTE '/var/load_scripts/get_log_data.sh' ON HOST
FORMAT 'TEXT' (DELIMITER '|');

```

创建一个名为sales_out的可写外部表，该表使用gpfdist将输出数据写入名为sales.out的文件。使用竖线（|）作为列定界符，并使用空白作为NULL来格式化文件。

```

CREATE WRITABLE EXTERNAL TABLE sales_out (LIKE sales)
LOCATION ('gpfdist://etl1:8081/sales.out')
FORMAT 'TEXT' ( DELIMITER '|' NULL ' ')
DISTRIBUTED BY (txn_id);

```

创建一个可写的外部Web表，将segment接收的输出数据通过管道传输到名为to_adreport_etl.sh的可执行脚本：

```

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来卸载所选数据：

```

INSERT INTO campaign_out SELECT * FROM campaign WHERE
customer_id=123;

```

注解

当您指定LOG ERRORS子句时，Greenplum数据库将捕获读取外部表数据时发生的错误。您可以查看和管理捕获的错误日志数据。

- 使用内置的SQL函数`gp_read_error_log('table_name')`。

它要求对`table_name`具有SELECT特权。本示例使用COPY命令显示加载到表ext_expenses中的数据的错误日志信息：

```
SELECT * from gp_read_error_log('ext_expenses');
```

有关错误日志格式的信息，请参阅Greenplum数据库管理员指南中的[查看错误日志中的坏行](#)。

如果`table_name`不存在，则该函数返回FALSE。

- 如果指定表存在错误日志数据，则新的错误日志数据将附加到现有错误日志数据中。错误日志信息不会复制到mirror。

- 使用内置的SQL函

数`gp_truncate_error_log('table_name')`删

除`table_name`的错误日志数据。它需要表所有者特权。本示例删除将数据移入表ext_expenses时捕获的错误日志信息：

```
SELECT gp_truncate_error_log('ext_expenses');
```

如果`table_name`不存在，则该函数返回FALSE。

指定*通配符可删除当前数据库中现有表的错误日志信息。指定字符串*.*以删除所有数据库错误日志信息，包括由于先前的数据库问题而未被删除的错误日志信息。如果指定*，则需要数据库所有者特权。如果指定*.*，则需要操作系统超级用户特权。

当使用gpfdist, gpfdists或file协议定义了多个Greenplum数据库外部表并访问Linux系统中的同一命名管道时，Greenplum数据库会将对命名管道的访问限制为单个读取器。如果第二个读取器尝试访问命名管道，则返回错误。

兼容性

`CREATE EXTERNAL TABLE`是Greenplum数据库扩展。SQL标准没有为外部表做任何准备。

另见

[CREATE TABLE AS](#), [CREATE TABLE](#), [COPY](#), [SELECT INTO](#), [INSERT](#)

Parent topic: [SQL Command Reference](#)

WRAPPER

Greenplum数据库® 6.0文档

定义一个新的外部数据包装器。

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

概要

```
CREATE FOREIGN DATA WRAPPER name
    [ HANDLER handler_function | NO HANDLER ]
    [ VALIDATOR validator_function | NO VALIDATOR ]
    [ OPTIONS ( [ mpp_execute { 'master' | 'any' | 'all
segments' } [ , ] ] option 'value' [ , ... ] ) ]
```

描述

CREATE FOREIGN DATA WRAPPER在当前数据库中创建一个新的外部数据包装器。 定义外部数据包装器的用户将成为其所有者。

只有超级用户才能创建外部数据包装器。

参数

name

要创建的外部数据包装器的名称。 该名称在数据库中必须唯一。

HANDLER *handler_function*

Greenplum数据库调用以检索外部表的执行函数的先前注册函数的名称。*handler_function*必须不带任何参数，其返回类型必须是fdw_handler。

可以创建没有处理程序函数的外部数据包装器，但是只能使用此类包装器声明而不访问外部表。

VALIDATOR *validator_function*

Greenplum数据库调用以检查提供给外部数据包装程序的选项的



先前注册函数的名称。此函数还检查使用外部数据包装器的外部服务器，用户映射和外部表的选项。如果未指定验证器函数或指定NO VALIDATOR，则Greenplum数据库在创建时不会检查选项。（根据实现的不同，外部数据包装程序可能会在运行时忽略或拒绝无效的选项。）

*validator_function*必须带有两个参数：一个是text[]类型，它包含存储在系统catalog中的选项数组，另一个是oid类型，它标识包含选项的系统catalog的OID。

返回类型将被忽略；*validator_function*应该使用ereport(ERROR)函数报告无效的选项。

OPTIONS (*option 'value' [, ...]*)

新的外部数据包装程序的选项。选项名称必须唯一。选项名称和值是特定于外部数据包装器的，并使用外部数据包装器的*validator_function*进行了验证。

`mpp_execute { 'master' | 'any' | 'all segments' }`

一个选项，用于标识外部数据包装器从其请求数据的主机：

- master (默认) — 从master主机请求数据。
- any — 向master主机或任一segment请求数据，具体取决于哪条路径的成本更低。
- all segments — 从所有segment请求数据。要支持此选项值，外部数据包装器必须具有将segment与数据匹配的策略。

可以在多个命令中指定mpp_execute选项：CREATE FOREIGN TABLE, CREATE SERVER和CREATE FOREIGN DATA WRAPPER。外部表设置优先于外部服务器设置，然后是外部数据包装器设置。

注解

外部数据包装器功能仍在开发中。查询的优化是原始的（大部分留给包装器）。

示例

创建一个无用的外部数据包装器，命名为dummy：

```
CREATE FOREIGN DATA WRAPPER dummy;
```

使用名为file_fdw_handler的处理函数创建一个名为file的外部数据包装器：

```
CREATE FOREIGN DATA WRAPPER file HANDLER file_fdw_handler;
```

创建一个名为mywrapper的外部数据包装器，其中包括一个选项：

```
CREATE FOREIGN DATA WRAPPER mywrapper OPTIONS (debug  
'true');
```

兼容性

CREATE FOREIGN DATA WRAPPER符合ISO/IEC 9075-9 (SQL/MED)，但LIBRARY和VALIDATOR子句是扩展是例外，而Greenplum数据库中未实现标准子句LIBRARY和LANGUAGE。

但是请注意，整个SQL/MED函数尚未符合要求。

另见

[ALTER FOREIGN DATA WRAPPER](#), [DROP FOREIGN DATA WRAPPER](#), [CREATE SERVER](#), [CREATE USER MAPPING](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一个新的外部表。

概要

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name ( [
    column_name data_type [ OPTIONS ( option 'value' [ , ... ]
] ) ] [ COLLATE collation ] [ column_constraint [ ... ] ]
[ , ... ]
]
    SERVER server_name
[ OPTIONS ( [ mpp_execute { 'master' | 'any' | 'all
segments' } [ , ] ] option 'value' [ , ... ] ) ]
```

其中`column_constraint`为：

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  DEFAULT default_expr }
```

描述

`CREATE FOREIGN TABLE`在当前数据库中创建一个新的外部表。创建外部表的用户将成为其所有者。

如果您对表名称进行模式限定（例如，`CREATE FOREIGN TABLE myschema.mytable ...`），则Greenplum数据库将在指定模式中创建表。否则，将在当前模式中创建外部表。外部表的名称必须与同一模式中任何其他外部表，表，序列，索引或视图的名称不同。

因为`CREATE FOREIGN TABLE`自动创建一种数据类型，该数据类型表示与外部表的一行相对应的复合类型，所以外部表不能与同一模式中的任何现有数据类型具有相同的名称。

要创建外部表，您必须在外部服务器上具有`USAGE`特权，并且对表中使用的所有列类型都具有`USAGE`特权。



参数

IF NOT EXISTS

如果已经存在同名关系，则不要抛出错误。在这种情况下，Greenplum数据库会发出通知。请注意，不能保证现有关系类似于将要创建的关系。

table_name

要创建的外部表的名称（可以由模式指定）。

column_name

在新的外部表中创建的列的名称。

data_type

列的数据类型，包括数组说明符。

NOT NULL

该列不允许包含空值。

NULL

该列允许包含空值。这是默认值。

提供此子句仅是为了与非标准SQL数据库兼容。不建议在新应用中使用它。

DEFAULT *default_expr*

DEFAULT子句为其定义所在的列分配默认值。该值是任何无变量表达式；Greenplum数据库不允许子查询和对当前表中其他列的交叉引用。默认表达式的数据类型必须与列的数据类型匹配。

Greenplum数据库在未为列指定值的任何插入操作中使用默认表达式。如果列没有默认值，则默认值为null。

server_name

用于外部表的现有服务器的名称。有关定义服务器的详细信息，请参见[CREATE SERVER](#)。

OPTIONS (*option 'value'* [, ...])

新外部表或其列之一的选项。 虽然选项名称必须唯一，但表选项和列选项可能具有相同的名称。 选项名称和值是特定于外部数据包装程序的。 Greenplum数据库使用外部数据包装程序的*validator_function*验证选项和值。

`mpp_execute { 'master' | 'any' | 'all segments' }`

一个选项，用于标识外部数据包装器从其请求数据的主机：

- `master` (默认设置) - 从master主机请求数据。
- `any` — 向master主机或任一segment请求数据，具体取决于哪条路径的成本更低。
- `all segments` — 从所有segment中请求数据。要支持此选项值，外部数据包装器必须具有将segment与数据匹配的策略。

外部表`mpp_execute`选项和受支持的特定模式的使用是特定于外部数据包装器的。

可以在多个命令中指定`mpp_execute`选项：`CREATE FOREIGN TABLE`, `CREATE SERVER`和`CREATE FOREIGN DATA WRAPPER`。 外部表设置优先于外部服务器设置，然后是外部数据包装器设置。

示例

使用名为`film_server`的服务器创建一个名为`films`的外部表：

```
CREATE FOREIGN TABLE films (
    code          char(5) NOT NULL,
    title         varchar(40) NOT NULL,
    did           integer NOT NULL,
    date_prod     date,
    kind          varchar(10),
    len            interval hour to minute
)
SERVER film_server;
```

兼容性

`CREATE FOREIGN TABLE`基本上符合SQL标准；但是，与`CREATE TABLE`一样，Greenplum数据库允许NULL约束和零列外部表。指定默认值的功能是Greenplum数据库扩展，以及`mpp_execute`选项。

另见

[ALTER FOREIGN TABLE](#), [DROP FOREIGN TABLE](#), [CREATE TABLE](#), [CREATE SERVER](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT

PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一个新函数。

概要

```

CREATE [OR REPLACE] FUNCTION name
  ( [ argmode] [argname] argtype [ { DEFAULT | = }
default_expr ] [ , ... ] )
  [ RETURNS rettype
    | RETURNS TABLE ( column_name column_type [ , ... ] )
  ]
  { LANGUAGE langname
  | WINDOW
  | IMMUTABLE | STABLE | VOLATILE | [NOT] LEAKPROOF
  | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT |
  STRICT
  | [EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY
  DEFINER
  | EXECUTE ON { ANY | MASTER | ALL SEGMENTS }
  | COST execution_cost
  | SET configuration_parameter { TO value | = value |
  FROM CURRENT }
  | AS 'definition'
  | AS 'obj_file', 'link_symbol' } ...
  [ WITH ( { DESCRIBE = describe_function
            } [ , ... ] ) ]

```

描述

`CREATE FUNCTION` 定义一个新函数。`CREATE OR REPLACE FUNCTION` 要么创建新函数，要么替换现有定义。

新函数的名称不得与任何在同一模式中具有相同输入参数类型的现有函数匹配。但是，不同参数类型的函数可能共享一个名称（重载）。

要更新现有函数的定义，请使用`CREATE OR REPLACE FUNCTION`。不能以这种方式更改函数的名称或参数类型（这实际上会创建一个新的，不同的函数）。同样，`CREATE OR REPLACE FUNCTION` 将不允许您更改现有函数的返回类型。为此，必须删除并重新创建该函数。使用`OUT`参数时，这意味着您不能更改任何`OUT`参数的类。除非删除该函数。如果删除然后重新创建函数，则必须删除引用旧函数的现有对象（规则，视图，触发器等）。使用`CREATE OR REPLACE FUNCTION` 来更改函数定义，而不会破坏引用该函数的对象。

创建函数的用户将成为该函数的所有者。

为了能够创建函数，您必须对参数类型和返回类型具有USAGE特权。

有关创建函数的更多信息，请参见PostgreSQL文档的[用户定义函数](#)部分。

限制使用VOLATILE和STABLE函数

为了防止数据在Greenplum数据库的各个segment之间变得不同步，如果分类为STABLE或VOLATILE的任何函数包含SQL或以任何方式修改了数据库，则不能在segment级别执行该函数。例如，不允许在Greenplum数据库中的分布式数据上执行诸如random()或timeofday()之类的函数，因为它们可能会导致segment实例之间的数据不一致。

为了确保数据的一致性，可以在master数据库上评估并执行的语句中安全地使用VOLATILE和STABLE函数。例如，以下语句始终在master数据库上执行（没有FROM子句的语句）：

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

如果语句的FROM子句包含一个分布式表，并且FROM子句中使用的函数仅返回一组行，则可以在这些segment上执行：

```
SELECT * FROM foo();
```

此规则的一个例外是返回表引用 (rangeFuncs) 的函数或使用refCursor数据类型的函数。请注意，您无法从Greenplum数据库中的任何类型的函数返回refcursor。

函数易变性和EXECUTE ON属性

易变性属性 (IMMUTABLE, STABLE, VOLATILE) 和EXECUTE ON属性指定函数执行的两个不同方面。通常，易变性表示执行该函数的时间，EXECUTE ON表示执行该函数的位置。

例如，可以在查询计划时执行使用IMMUTABLE属性定义的函数，而必须对查询中的每一行执行带有VOLATILE属性的函数。具有EXECUTE ON MASTER属性的函数仅在master上执行，具有EXECUTE ON ALL SEGMENTS属性的函数仅在所有primary实例（而不是master）上执行。

请参阅Greenplum数据库管理员指南中的[使用函数和运算符](#)。

函数和复制表

在复制表上仅执行SELECT命令的用户定义函数可以在segment上运行。使用DISTRIBUTED REPLICATED子句创建的复制表将其所有行存储在每个segment上。函数在segment上读取它们是安全的，但是对复制表的更新必须在master实例上执行。

参数

name

要创建的函数的名称（可以由模式指定）。

argmode

参数的模式：IN, OUT, INOUT或VARIADIC。如果省略，则默认值为IN。只有OUT参数可以跟随声明为VARIADIC的参数。同样，OUT和INOUT参数不能与RETURNS TABLE表示法一起使用。

argname

参数的名称。某些语言（当前仅SQL和PL/pgSQL）允许您在函数主体中使用名称。对于其他语言，就函数本身而言，输入参数的名称只是额外的文档。但是您可以在调用函数时使用输入参数名称来提高可读性。在任何情况下，输出参数的名称都是有效的，因为它在结果行类型中定义了列名称。（如果省略输出参数的名称，则系统将选择默认列名称。）

argtype

函数参数的数据类型（可以由模式指定）（如果有）。参数类型可以是基本类型，复合类型或域类型，或者可以引用表列的类型。

根据实现语言的不同，还可以允许指定伪类型，例如cstring。

伪类型表示实际参数类型未完全指定，或者位于普通SQL数据类型集之外。

通过编写tablename.columnname%TYPE来引用列的类型。使用此函数有时可以使函数独立于表定义的更改。

default_expr

如果未指定参数，则用作默认值的表达式。该表达式必须对参数的参数类型具有强制性。只有IN和INOUT参数可以具有默认值。参数列表中紧随默认值的参数之后的每个输入参数也必须具有默认值。

rettype

返回数据类型（可以由模式指定）。返回类型可以是基本类型，复合类型或域类型，或者可以引用表列的类型。根据实现语言的不同，还可以允许指定伪类型，例如cstring。如果该函数不应该返回值，则将void指定为返回类型。

当有OUT或INOUT参数时，可以省略RETURNS子句。如果存在，

则必须与输出参数隐含的结果类型相符： RECORD，如果有多个输出参数，或者与单个输出参数具有相同的类型。

SETOF修饰符表示该函数将返回一组项目，而不是单个项目。

通过编写`tablename.columnname%TYPE`来引用列的类型。

column_name

RETURNS TABLE语法中的输出列的名称。这实际上是声明命名OUT参数的另一种方法，除了RETURNS TABLE还暗含RETURNS SETOF。

column_type

RETURNS TABLE语法中输出列的数据类型。

langname

函数所使用的语言的名称。可以是SQL, C, internal或用户定义的过程语言的名称。有关Greenplum数据库中支持的过程语言，请参见[CREATE LANGUAGE](#)。为了向后兼容，名称可以用单引号引起起来。

WINDOW

WINDOW表示该函数是窗口函数，而不是普通函数。当前这仅对用C编写的函数有用。替换现有函数定义时，不能更改WINDOW属性。

IMMUTABLE

STABLE

VOLATILE

LEAKPROOF

这些属性将有关函数的行为通知查询优化器。最多可以指定一种选择。如果这些都不出现，则默认为VOLATILE。由于Greenplum数据库当前对VOLATILE函数的使用受到限制，因此，如果一个函数确实是IMMUTABLE，则必须将其声明为可以不受限制地使用。

IMMUTABLE指示该函数无法修改数据库，并且在给定相同的参数值时始终返回相同的结果。它不进行数据库查找，或者使用其参数列表中不直接存在的信息。如果指定了此选项，则可以使用函数值立即替换具有全常数参数的任何函数调用。

STABLE表示该函数无法修改数据库，并且在单个表扫描中它将针对相同的参数值一致地返回相同的结果，但其结果可能会在SQL语句之间发生变化。对于结果取决于数据库查找，参数值（例如当前时区）等等的函数，这是适当的选择。还要注意，`current_timestamp`系列函数符合稳定条件，因为它们的值在事务中不会更改。

VOLATILE表示该函数值即使在一次表扫描中也可以更改，因此无法进行优化。从这个意义上说，相对来说很少有数据库函数是易变的。一些示例是`random()`, `timeofday()`。但是请注意，任何具有副作用的函数都必须归类为易失性，即使其结果是可以预测的，也可以防止调用被优化掉。一个示例是`setval()`。

`LEAKPROOF`表示该函数没有副作用。除了返回值以外，它不显示有关其参数的任何信息。例如，对于某些参数值而不是其他参数值抛出错误消息的函数，或者在任何错误消息中包含参数值的函数，都不是防漏的。查询优化器可以将防泄漏函数（但不能禁止其他功能）推入使用`security_barrier`选项创建的视图中。请参阅[CREATE VIEW](#)和[CREATE RULE](#)。该选项只能由超级用户设置。

`CALLED ON NULL INPUT`
`RETURNS NULL ON NULL INPUT`
`STRICT`

`CALLED ON NULL INPUT`（默认值）表示该函数的某些参数为`null`时将正常调用该函数。然后，函数作者有责任检查空值（如有必要）并做出适当响应。`RETURNS NULL ON NULL INPUT`或`STRICT`表示该函数在任何参数为`null`时始终返回`null`。如果指定了此参数，则在参数为空时不执行该函数；而是自动假定为空结果。

`[EXTERNAL] SECURITY INVOKER`
`[EXTERNAL] SECURITY DEFINER`

`SECURITY INVOKER`（默认值）指示该函数将以调用该函数的用户权限执行。`SECURITY DEFINER`指定要使用创建该函数的用户的特权来执行该函数。允许使用`EXTERNAL`关键字来实现SQL一致性，但它是可选的，因为与SQL不同，此功能不仅适用于外部函数，还适用于所有函数。

`EXECUTE ON ANY`
`EXECUTE ON MASTER`
`EXECUTE ON ALL SEGMENTS`

`EXECUTE ON`属性指定在查询执行过程中调用函数时在何处（`master`或`segment`实例）执行。

`EXECUTE ON ANY`（默认值）表示该函数可以在`master`或任何`segment`实例上执行，并且无论在何处执行，它均返回相同的结果。Greenplum数据库确定函数在哪里执行。

`EXECUTE ON MASTER`表示该函数只能在`master`实例上执行。

`EXECUTE ON ALL SEGMENTS`表明对于每个调用，该函数必须在所有`primary`实例上执行，但不能在`master`实例上执行。该函数的总结果是所有`segment`实例的结果的`UNION ALL`。

有关使用`EXECUTE ON`属性的信息，请参阅[注释](#)。

`COST execution_cost`

一个正数，标识函数的估计执行成本，以`cpu_operator_cost`为单位。如果函数返回一个集合，则`execution_cost`会标识每个返回行的成本。如果未指定成本，则C语言和内部函数默认为1个单位，而其他语言的函数默认为100个单位。当您指定较大的`execution_cost`值时，优化器将尝试较少地评估函数。

`configuration_parameter`
`value`

进入该函数时，`SET`子句将一个值应用于会话配置参数。函数退

出时，配置参数将恢复为其先前的值。SET FROM CURRENT会将执行CREATE FUNCTION时当前的参数值保存为进入该函数时要应用的值。

definition

定义函数的字符串常量；含义取决于语言。它可以是内部函数名称，目标文件的路径，SQL命令或过程语言的文本。

obj_file, link_symbol

当C语言源代码中的函数名称与SQL函数的名称不同时，这种形式的AS子句用于可动态加载的C语言函数。字符串*obj_file*是包含动态可加载对象的文件的名称，而*link_symbol*是C语言源代码中的函数的名称。如果省略链接符号，则假定它与所定义的SQL函数的名称相同。所有函数的C名称必须不同，因此必须为重载的SQL函数赋予不同的C名称（例如，将参数类型用作C名称的一部分）。建议相对于\$libdir（位于\$GPHOME/lib）或通过动态库路径（由dynamic_library_path服务器配置参数设置）定位共享库。如果新安装位于其他位置，则可以简化版本升级。

describe_function

解析调用此函数的查询时要执行的回调函数的名称。回调函数返回一个表示结果类型的元组描述符。

注解

用于自定义函数的所有已编译代码（共享库文件）必须放在Greenplum数据库阵列（master和所有segment）中每个主机的相同位置。此位置也必须位于LD_LIBRARY_PATH中，以便服务器可以找到文件。建议在Greenplum阵列中的所有master实例上，相对于\$libdir（位于\$GPHOME/lib）或通过动态库路径（由dynamic_library_path服务器配置参数设置）定位共享库。

输入参数和返回值允许使用完整的SQL类型语法。但是，类型规范的某些详细信息（例如，类型为*numeric*的精度字段）是基础函数实现的职责，而CREATE FUNCTION命令无法识别或强制执行。

Greenplum数据库允许函数重载。相同的名称可以用于多个不同的函数，只要它们具有不同的输入参数类型即可。但是，所有函数的C名称都必须不同，因此必须为重载的C函数赋予不同的C名称（例如，将参数类型用作C名称的一部分）。

如果两个函数具有相同的名称和输入参数类型，而忽略任何OUT参数，则认为它们是相同的。因此，例如，这些声明冲突：

```
CREATE FUNCTION foo(int) ...
CREATE FUNCTION foo(int, out text) ...
```

具有不同参数类型列表的函数在创建时不会被认为是冲突的，但是如果提供了参数默认值，则它们在使用中可能会发生冲突。例如，考虑：

```
CREATE FUNCTION foo(int) ...
CREATE FUNCTION foo(int, int default 42) ...
```

由于不清楚应调用哪个函数，因此调用`foo(10)`将失败。

当重复的`CREATE FUNCTION`调用引用相同的目标文件时，该文件仅被加载一次。要卸载并重新加载文件，请使用`LOAD`命令。

您必须对一种语言具有`USAGE`特权，才能使用该语言定义函数。

使用美元引号而不是常规的单引号语法编写函数定义字符串通常会很有帮助。如果不使用美元引号，则必须通过将它们加倍来转义函数定义中的任何单引号或反斜杠。用美元引用的字符串常量由一个美元符号（\$），一个零个或多个字符的可选标记，另一个美元符号，构成字符串内容的任意字符序列，一个美元符号以及与开始此美元引用相同的标记和一个美元符号组成。在用美元引用的字符串中，可以使用单引号，反斜杠或任何字符而无需转义。字符串内容始终按原义书写。例如，以下是两种使用美元引号指定字符串“Dianne's horse”的方法：

```
$$Dianne's horse$$
$SomeTag$Dianne's horse$SomeTag$
```

如果将`SET`子句附加到函数，则在函数内部针对相同变量执行的`SET LOCAL`命令的作用仅限于该函数；函数退出时，配置参数的先前值仍会恢复。但是，普通的`SET`命令（不包含`LOCAL`）会覆盖`CREATE FUNCTION SET`子句，就像以前的`SET LOCAL`命令所使用的一样。除非当前事务回滚，否则该命令的效果将在函数退出后继续存在。

如果将带有`VARIADIC`参数的函数声明为`STRICT`，则严格性检查将测试可变参数数组整体是否为非空。如果数组具有空元素，PL/pgSQL仍将调用该函数。

用`CREATE OR REPLACE FUNCTION`替换现有函数时，更改参数名称存在限制。您无法更改已分配给任何输入参数的名称（尽管您可以将名称添加到以前没有输入参数的参数中）。如果有多个输出参数，则不能更改输出参数的名称，因为这将更改描述函数结果的匿名复合类型的列名称。进行这些限制是为了确保函数的现有调用在被替换时不会停止工作。

将函数与查询一起用于分布式数据

在某些情况下，如在FROM子句中指定的表中的数据分布在Greenplum数据库segment上，Greenplum数据库不支持在查询中使用函数。例如，此SQL查询包含函数func()：

```
SELECT func(a) FROM table1;
```

如果满足以下所有条件，则该函数不支持在查询中使用：

- 表table1的数据分布在Greenplum数据库segment上。
- 函数func()从分布式表中读取或修改数据。
- 函数func()返回多个行或采用来自table1的参数(a)。

如果不满足任何条件，则支持该函数。具体来说，如果满足以下任一条件，则支持该函数：

- 函数func()不会访问分布式表中的数据，也不会访问仅在Greenplum数据库master上的数据。
- 表table1是仅在master上。
- 函数func()仅返回一行，并且仅接受常量值的输入参数。如果可以将其更改为不需要输入参数，则支持该函数。

使用EXECUTE ON属性

大多数执行查询以访问表的函数只能在master上执行。但是，仅对复制表执行SELECT查询的函数可以在segment上运行。如果函数访问哈希分布表或随机表，则应使用EXECUTE ON MASTER属性定义该函数。否则，在复杂的查询中使用该函数时，该函数可能会返回错误的结果。如果没有该属性，优化器优化可能会确定将函数调用推到segment实例将是有益的。

这些是使用EXECUTE ON MASTER或EXECUTE ON ALL SEGMENTS属性定义的函数的限制：

- 该函数必须是返回集合的函数。
- 该函数不能在查询的FROM子句中。
- 该函数不能在带有FROM子句的查询的SELECT列表中。
- 包含该函数的查询从GPORCA退回到Postgres查询计划程序。

示例

一个非常简单的加法函数：

```
CREATE FUNCTION add(integer, integer) RETURNS integer
AS 'select $1 + $2;'
LANGUAGE SQL
IMMUTABLE
RETURNS NULL ON NULL INPUT;
```

在PL/pgSQL中使用参数名称递增一个整数：

```
CREATE OR REPLACE FUNCTION increment(i integer) RETURNS
integer AS $$%
BEGIN
    RETURN i + 1;
END;
$$ LANGUAGE plpgsql;
```

为PL/pgSQL函数增加每个查询的默认segment主机内存：

```
CREATE OR REPLACE FUNCTION function_with_query() RETURNS
SETOF text AS $$%
BEGIN
    RETURN QUERY
        EXPLAIN ANALYZE SELECT * FROM large_table;
END;
$$ LANGUAGE plpgsql
SET statement_mem='256MB';
```

使用多态类型返回ENUM数组：

```
CREATE TYPE rainbow AS
ENUM('red','orange','yellow','green','blue','indigo','violet')

CREATE FUNCTION return_enum_as_array( anyenum, anyelement,
anyelement )
RETURNS TABLE (ae anyenum, aa anyarray) AS $$%
SELECT $1, array[$2, $3]
$$ LANGUAGE SQL STABLE;

SELECT * FROM return_enum_as_array('red'::rainbow,
'green'::rainbow, 'blue'::rainbow);
```

返回包含多个输出参数的记录：

```
CREATE FUNCTION dup(in int, out f1 int, out f2 text)
AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$%
LANGUAGE SQL;

SELECT * FROM dup(42);
```

您可以使用明确命名的复合类型来更详细地执行相同的操作：

```

CREATE TYPE dup_result AS (f1 int, f2 text);
CREATE FUNCTION dup(int) RETURNS dup_result
    AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$ 
    LANGUAGE SQL;

SELECT * FROM dup(42);

```

返回多列的另一种方法是使用TABLE函数：

```

CREATE FUNCTION dup(int) RETURNS TABLE(f1 int, f2 text)
    AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$ 
    LANGUAGE SQL;

SELECT * FROM dup(4);

```

该函数在EXECUTE ON ALL SEGMENTS中定义，以在所有primary实例上运行。SELECT命令执行返回在每个segment实例上运行时间的函数。

```

CREATE FUNCTION run_on_segs (text) returns setof text as $$ 
begin
    return next ($1 || ' - ' || now()::text );
end;
$$ language plpgsql VOLATILE EXECUTE ON ALL SEGMENTS;

SELECT run_on_segs('my test');

```

此功能在parts表中查找part名称。parts表是复制表，因此该函数可以在master或primary上执行。

```

CREATE OR REPLACE FUNCTION get_part_name(partno int) RETURNS
text AS
$$
DECLARE
    result text := '';
BEGIN
    SELECT part_name INTO result FROM parts WHERE part_id =
partno;
    RETURN result;
END;
$$ LANGUAGE plpgsql;

```

如果在master上执行SELECT get_part_name(100);，该函数在master上执行。（master实例将查询定向到单个primary。）如果orders是分布式表，并且您执行以下查询，则get_part_name()函数将在primary上执行。

```
SELECT order_id, get_part_name(orders.part_no) FROM orders;
```

兼容性

CREATE FUNCTION在SQL: 1999及更高版本中定义。Greenplum数据库版本相似，但不完全兼容。这些属性不是可移植的，不同的可用语言也不是。

为了与某些其他数据库系统兼容，可以在*argname*之前或之后写入*argmode*。但是只有第一种方法是符合标准的。

对于参数默认值，SQL标准仅使用DEFAULT关键字指定语法。在T-SQL和Firebird中使用=语法。

另见

[ALTER FUNCTION](#) , [DROP FUNCTION](#) , [LOAD](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0 文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一个新的数据库角色。

概要

```
CREATE GROUP name [[WITH] option [ ... ]]
```

其中 *option* 可以是：

```
SUPERUSER | NOSUPERUSER
| CREATEDB | NOCREATEDB
| CREATEROLE | NOCRAETEROLE
| CREATEUSER | NOCREATEUSER
| CREATEEXTTABLE | NOCREATEEXTTABLE
[ ( attribute='value'[, ...] ) ]
    where attributes and value are:
        type='readable' | 'writable'
        protocol='gpfdist' | 'http'
| INHERIT | NOINHERIT
| LOGIN | NOLOGIN
| CONNECTION LIMIT connlimit
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
| VALID UNTIL 'timestamp'
| IN ROLE rolename [, ...]
| ROLE rolename [, ...]
| ADMIN rolename [, ...]
| RESOURCE QUEUE queue_name
| RESOURCE GROUP group_name
| [ DENY deny_point ]
| [ DENY BETWEEN deny_point AND deny_point ]
```

描述

CREATE GROUP 是 [CREATE ROLE](#) 的别名。

兼容性

SQL 标准中没有 CREATE GROUP 语句。



另见

[CREATE ROLE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一个新索引。

概要

```
CREATE [UNIQUE] INDEX [name] ON table_name [USING method]
    ( {column_name | (expression)} [COLLATE parameter]
    [opclass] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...]
    )
    [ WITH ( storage_parameter = value [, ...] ) ]
    [ TABLESPACE tablespace ]
    [ WHERE predicate ]
```

描述

`CREATE INDEX`在指定表的指定列上构造索引。索引主要用于增强数据库性能（尽管使用不当会导致性能降低）。

索引的键字段指定为列名，或者指定为用括号括起来的表达式。如果`index`方法支持多列索引，则可以指定多个字段。

索引字段可以是根据表行的一个或多个列的值计算的表达式。该功能可用于基于基本数据的某种转换来快速访问数据。例如，在`upper(col)`上计算的索引将允许子句`WHERE upper(col) = 'JIM'`使用索引。

Greenplum数据库提供了索引方法B树，位图，GiST，SP-GiST和GIN。用户还可以定义自己的索引方法，但这相当复杂。

存在`WHERE`子句时，将创建部分索引。部分索引是仅包含表一部分的条目的索引，通常索引的部分比表的其余部分更有用。例如，如果您有一个既包含开票订单又包含未开票订单的表，其中未开票订单仅占总表的一小部分，但最常被选择，则可以通过仅在该部分上创建索引来提高性能。

`WHERE`子句中使用的表达式可能仅引用基础表的列，但它可以使用所有列，而不仅仅是被索引的列。`WHERE`中也禁止子查询。相同的限制适用于作为表达式的索引字段。

索引定义中使用的所有函数和运算符必须是不可变的。它们的结果必

须仅取决于其参数，而不得取决于任何外部影响（例如另一个表的内容或参数值）。此限制可确保索引的行为得到良好定义。要在索引表达式或WHERE子句中使用用户定义的函数，请记住在创建函数时将其标记为IMMUTABLE。

参数

UNIQUE

创建索引并每次添加数据时，检查表中是否有重复值。重复的条目将产生错误。唯一索引仅适用于B树索引。在Greenplum数据库中，仅当索引键的列与Greenplum分布键相同（或超集）时，才允许唯一索引。在分区表上，唯一索引仅在单个分区中受支持 - 不可以跨所有分区。

name

要创建的索引的名称。索引始终与其父表在相同的模式中创建。如果省略该名称，则Greenplum数据库将基于父表的名称和索引列的名称选择一个合适的名称。

table_name

要建立索引的表的名称（可以由模式指定）。

method

要使用的索引方法的名称。选择为btree, bitmap, gist, spgist和gin。默认方法是btree。

当前，仅B树，GiST和GIN索引方法支持多列索引。默认情况下，最多可以指定32个字段。当前只有B树支持唯一索引。

GPORCAS仅支持B树，位图和GiST索引。GPORCAS会忽略使用不受支持的索引方法创建的索引。

column_name

在其上创建索引的表的列的名称。仅B树，位图，GiST和GIN索引方法支持多列索引。

expression

一种基于表的一个或多个列的表达式。表达式通常必须用括号括起来，如语法所示。但是，如果表达式具有函数调用的形式，则可以省略括号。

collation

索引使用的归类名称。默认情况下，索引使用为要建立索引的列声明的归类或要建立索引的表达式的结果归类。具有非默认归类的索引对于涉及使用非默认归类的表达式的查询很有用。

opclass

运算符类的名称。operator类标识该列的索引要使用的运算符。例如，在四字节整数上的B树索引将使用int4_ops类（此运算符类包含四字节整数的比较函数）。实际上，列数据类型的默认运算符类通常就足够了。拥有运算符类的要点是，对于某些数据类型，可能会有不止一种有意义的排序。例如，复数数据

类型可以按绝对值或实数进行排序。为此，我们可以为数据类型定义两个运算符类，然后在创建索引时选择适当的类。

ASC

指定升序排序（默认）。

DESC

指定降序排序。

NULS FIRST

指定null排在非null之前。当指定DESC时，这是默认设置。

NULS LAST

指定空值在非空值之后排序。如果未指定DESC，这是默认设置。

storage_parameter

特定于索引方法的存储参数的名称。每个索引方法都有其自己的一组允许的存储参数。

FILLFACTOR-B树，位图，GiST和SP-GiST索引方法都接受此参数。索引的FILLFACTOR是一个百分比，它确定索引方法将尝试打包索引页面的程度。对于B树，在初始索引构建期间以及在向右扩展索引时（添加新的最大键值），叶子页将填充到该百分比。如果页面随后完全填满，它们将被拆分，从而导致索引效率逐渐下降。B树使用默认填充因子90，但可以选择10到100之间的任何整数值。如果表是静态的，则fillfactor 100最好最大程度地减小索引的物理大小，但是对于大量更新的表，较小的fillfactor更好地最小化对页面拆分的需求。其他索引方法以不同但大致相似的方式使用fillfactor；默认的填充因子因方法而异。

BUFFERING - 除FILLFACTOR之外，GiST索引还接受BUFFERING参数。BUFFERING确定Greenplum数据库是否使用PostgreSQL文档中[GiST缓冲构建](#)中描述的缓冲构建技术构建索引。如果设置为OFF，则禁用它；如果设置为ON，则启用；将其设置为AUTO时，最初将其禁用；但是，一旦索引大小达到[有效缓存大小](#)，它就会即时打开。默认为AUTO。

FASTUPDATE - GIN索引方法接受FASTUPDATE存储参数。

FASTUPDATE是一个布尔参数，用于禁用或启用GIN索引快速更新技术。值ON启用快速更新（默认值），而值OFF禁用它。有关更多信息，请参见PostgreSQL文档中的[GIN快速更新技术](#)。

Note: 通过ALTER INDEX关闭FASTUPDATE可以防止将来的插入进入挂起的索引条目列表，但本身不会刷新以前的条目。您可能需要稍后对表进行VACUUM处理，以确保清空待处理列表。

tablespace_name

在其中创建索引的表空间。如果未指定，则使用默认表空间。

predicate

部分索引的约束表达式。

注解

可以为索引的每一列指定一个运算符类。operator类标识该列的索引要使用的运算符。例如，在四字节整数上的B树索引将使用int4_ops类。此运算符类包含用于四字节整数的比较函数。实际上，列数据类型的默认运算符类通常就足够了。拥有运算符类的要点是，对于某些数据类型，可能会有不止一种有意义的排序。例如，我们可能想按绝对值或实数对复数数据类型进行排序。为此，我们可以为数据类型定义两个运算符类，然后在创建索引时选择适当的类。

对于支持排序扫描的索引方法（当前仅支持B树），可以指定可选的子句ASC, DESC, NULLS FIRST和/或NULLS LAST来修改索引的排序顺序。由于可以向前或向后扫描有序索引，因此创建单列DESC索引通常没有用 - 常规索引已经可以使用排序顺序。这些选项的值在于可以创建与混合排序查询所请求的排序顺序匹配的多列索引，例如SELECT ... ORDER BY x ASC, y DESC。如果您需要在依赖索引以避免排序步骤的查询中支持“nulls sort low”行为，而不是默认的“nulls sort high”行为，则NULLS选项很有用。

对于大多数索引方法，创建索引的速度取决于maintenance_work_mem的设置。较大的值将减少索引创建所需的时间，只要您不使其大于实际可用的内存量即可，这会促使计算机进行交换。

在分区表上创建索引后，该索引将传播到Greenplum数据库创建的所有子表。不支持在由Greenplum数据库创建的表上创建索引以供分区表使用。

仅当索引列与Greenplum分布键列相同（或超集）时，才允许使用UNIQUE索引。

附加优化表上不允许使用UNIQUE索引。

可以在分区表上创建UNIQUE索引。但是，唯一性仅在分区内有效。分区之间不强制唯一性。例如，对于具有基于年份的分区和基于季度的子分区的分区表，仅在每个单独的季度分区上实施唯一性。季度分区之间不强制唯一性

默认情况下，索引不用于IS NULL子句。在这种情况下，使用索引的最佳方法是使用IS NULL谓词创建部分索引。

对于具有100到100,000个不同值的列，位图索引的性能最佳。对于具有超过100,000个不同值的列，位图索引的性能和空间效率会下降。位图索引的大小与表中的行数乘以索引列中不同值的数量成正比。

少于100个不同值的列通常不会从任何类型的索引中受益太多。例如，对于男性和女性仅具有两个不同值的性别列将不是索引的理想选择。

Greenplum数据库的先前版本也具有R树索引方法。该方法已被删除，因为它与GiST方法相比没有明显的优势。如果指定了USING rtree，则CREATE INDEX会将其解释为USING gist。

有关GiST索引类型的更多信息，请参见[PostgreSQL文档](#)。

哈希索引的使用已在Greenplum数据库中禁用。

示例

要在films表的title列上创建B树索引：

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

要在表employee中的gender列上创建位图索引：

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

要在表达式lower(title)上创建索引，以允许有效的不区分大小写的搜索：

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

(在此示例中，我们选择省略索引名称，因此系统将选择一个名称，通常为films_lower_idx。)

要使用非默认排序规则创建索引：

```
CREATE INDEX title_idx_german ON films (title COLLATE "de_DE");
```

要创建具有非默认填充因子的索引：

```
CREATE UNIQUE INDEX title_idx ON films (title) WITH (fillfactor = 70);
```

要创建禁用了快速更新的GIN索引：

```
CREATE INDEX gin_idx ON documents_table USING gin
(locations) WITH (fastupdate = off);
```

要在表films的列code上创建索引并使索引驻留在表空间indexspace中：

```
CREATE INDEX code_idx ON films(code) TABLESPACE indexspace;
```

要在point属性上创建GiST索引，以便我们可以对转换函数的结果有效地使用box运算符：

```
CREATE INDEX pointloc ON points USING gist
(box(location,location));
SELECT * FROM points WHERE box(location,location) &&
'(0,0),(1,1)::box;
```

兼容性

CREATE INDEX是Greenplum数据库语言的扩展。SQL标准中没有索引的规定。

Greenplum数据库不支持并发创建索引（不支持CONCURRENTLY关键字）。

另见

[ALTER INDEX](#), [DROP INDEX](#), [CREATE TABLE](#), [CREATE OPERATOR CLASS](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一种新的过程语言。

概要

```
CREATE [ OR REPLACE ] [ PROCEDURAL ] LANGUAGE name
```

```
CREATE [ OR REPLACE ] [ TRUSTED ] [ PROCEDURAL ] LANGUAGE name
```

```
HANDLER call_handler [ INLINE inline_handler ]
[ VALIDATOR valfunction ]
```

描述

CREATE LANGUAGE在Greenplum数据库中注册了新的过程语言。随后，可以用这种新语言定义函数和触发过程。

Note: Greenplum数据库的过程语言已被制成“扩展名”，因此应与[CREATE EXTENSION](#)一起安装，而不是CREATE LANGUAGE。直接使用CREATE LANGUAGE应该仅限于扩展安装脚本。如果数据库中存在“裸”语言（可能是升级的结果），则可以使用CREATE EXTENSION *langname* FROM unpackaged将其转换为扩展名。

超级用户可以在Greenplum数据库中注册新语言。数据库所有者还可以在该数据库中注册pg_pltemplate catalog中列出的tmp1dbacreate字段为true的任何语言。默认配置仅允许数据库所有者注册受信任的语言。语言的创建者成为其所有者，以后可以将其删除，重命名或将所有权分配给新所有者。

CREATE OR REPLACE LANGUAGE将创建新语言或替换现有定义。如果该语言已经存在，则其参数将根据pg_pltemplate中指定的值或从pg_pltemplate中获取的值进行更新，但该语言的所有权和权限设置不会更改，并且假定使用该语言编写的任何现有函数仍然有效。除了创建语言的正常特权要求外，用户还必须是现有语言的超级用户或所有者。REPLACE情况主要用于确保该语言存在。如果该语言具有pg_pltemplate条目，则REPLACE实际上不会更改有关于该语言的任何内容，除非在异常情况下自创建该语言以来已对pg_pltemplate条目进行了修改。

`CREATE LANGUAGE`有效地将语言名称与处理程序函数关联，该处理函数负责执行以该语言编写的函数。对于以过程语言（C或SQL以外的语言）编写的函数，数据库服务器没有有关如何解释函数源代码的内置知识。该任务将传递给知道该语言详细信息的特殊处理程序。处理程序可以执行解析，语法分析，执行等所有工作，也可以充当Greenplum数据库与编程语言的现有实现之间的桥梁。处理程序本身是一种C语言函数，已编译为共享对象并按需加载，就像其他任何C函数一样。这些过程语言程序包包含在标准的Greenplum数据库发行版中：PL/pgSQL，PL/Perl和PL/Python。还为PL/Java和PL/R添加了语言处理程序，但是Greenplum数据库未预安装这些语言。有关使用这些过程语言开发函数的更多信息，请参见PostgreSQL文档中有关[过程语言](#)的主题。

PL/Perl，PL/Java和PL/R库分别需要安装正确版本的Perl，Java和R。

`CREATE LANGUAGE`命令有两种形式。在第一种形式中，用户指定所需语言的名称，Greenplum数据库服务器使用`pg_pltemplate`系统catalog来确定正确的参数。在第二种形式中，用户指定语言参数以及语言名称。您可以使用第二种形式来创建`pg_pltemplate`中未定义的语言。

当服务器在`pg_pltemplate` catalog中找到给定语言名称的条目时，即使命令包含语言参数，它也会使用catalog数据。此行为简化了旧转储文件的加载，这些文件可能包含有关语言支持函数的过时信息。

参数

TRUSTED

`TRUSTED`指定该语言不会授予用户原本就不会访问的数据的访问权限。如果在注册语言时省略此关键字，则只有具有Greenplum数据库超级用户特权的用户才能使用该语言创建新函数。

PROCEDURAL

这是一个干扰词。

name

新过程语言的名称。该名称在数据库中的语言之间必须唯一。包含对`plpgsql`，`plperl`和`plpythonu`的内置支持。默认情况下，在Greenplum数据库中已默认安装了语言`plpgsql`（PL/pgSQL）和`plpythonu`（PL/Python）。

HANDLER *call_handler*

如果服务器在`pg_pltemplate`中有用于指定语言名称的条目则忽略。先前注册的函数的名称，将调用该函数来执行过程语言函数。程序语言的调用处理程序必须使用编译后的语言编写，

例如使用版本1调用约定的C语言，并在Greenplum数据库中注册为不带任何参数的函数，并返回language_handler类型，该仅是用于标识函数的占位符类型作为呼叫处理程序。

INLINE *inline_handler*

以前注册的函数的名称，该函数被调用以执行用DO命令创建的该语言的匿名代码块。如果未指定inline_handler函数，则该语言不支持匿名代码块。处理函数必须采用类型为internal的一个参数，即DO命令的内部表示形式。该函数通常返回void。处理程序的返回值将被忽略。

VALIDATOR *valfunction*

如果服务器在pg_pltemplate中有用于指定语言名称的条目则忽略。先前注册的函数的名称，将调用该函数来执行过程语言函数。程序语言的调用处理程序必须使用编译式语言编写，例如使用版本1调用约定的C语言，并在Greenplum数据库中注册为不带任何参数的函数，并返回language_handler类型，该仅是用于标识函数的占位符类型作为呼叫处理程序。

注解

默认情况下，PL/pgSQL语言已在所有数据库中注册。PL/Python语言扩展已安装但未注册。

系统catalog pg_language记录有关当前安装的语言的信息。

要使用过程语言创建函数，用户必须具有该语言的USAGE特权。默认情况下，USAGE被授予PUBLIC（每个人）用于受信任的语言。如果需要，可以将其撤消。

过程语言是各个数据库的本地语言。您为单个数据库创建和删除语言。

如果服务器在pg_pltemplate中没有该语言的条目，则调用处理程序函数和验证器函数（如果有）必须已经存在。但是，当有一个条目时，函数就不必已经存在了。如果数据库中不存在它们，它们将被自动定义。

任何实现一种语言的共享库都必须位于Greenplum数据库阵列中所有segment主机上的LD_LIBRARY_PATH位置中。

示例

创建任何标准过程语言的首选方法是使用CREATE EXTENSION而不是CREATE LANGUAGE。例如：

```
CREATE EXTENSION plperl;
```

对于pg_pltemplate catalog中未知的语言：

```
CREATE FUNCTION plsample_call_handler() RETURNS
language_handler
AS '$libdir/plsample'
LANGUAGE C;
CREATE LANGUAGE plsample
HANDLER plsample_call_handler;
```

兼容性

CREATE LANGUAGE是Greenplum数据库扩展。

另见

[ALTER LANGUAGE](#), [CREATE EXTENSION](#), [CREATE FUNCTION](#),
[DROP EXTENSION](#), [DROP LANGUAGE](#), [GRANT DO](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一个新的运算符。

概要

```
CREATE OPERATOR name (
    PROCEDURE = funcname
    [, LEFTARG = lefttype] [, RIGHTARG = righttype]
    [, COMMUTATOR = com_op] [, NEGATOR = neg_op]
    [, RESTRICT = res_proc] [, JOIN = join_proc]
    [, HASHES] [, MERGES] )
```

描述

CREATE OPERATOR 定义一个新的运算符。 定义运算符的用户将成为其所有者。

运算符名称是以下列表中最多NAMEDATALEN-1（默认为63）个字符的序列： + - * / < > = ~ ! @ # % ^ & | ` ?

名称的选择有一些限制：

- --和/*不能出现在运算符名称的任何位置，因为它们将被用作注释的开头。
- 多字符运算符名称不能以+或-结尾，除非该名称也包含以下至少一个字符： ~ ! @ # % ^ & | ` ?

例如，@-是允许的运算符名称，但*-不是。此限制使Greenplum数据库可以解析与SQL兼容的命令，而无需在标记之间使用空格。

不建议将=>用作运算符。在将来的版本中可能完全不允许使用它。

运算符!=在输入时映射为<>，因此这两个名称始终相等。

必须至少定义LEFTARG和RIGHTARG之一。对于二元运算符，必须同时定义两者。对于右一元运算符，仅应定义LEFTARG，而对于左一元运算符，仅应定义RIGHTARG。

*funcname*过程必须事先使用CREATE FUNCTION定义，必须是IMMUTABLE，并且必须定义为接受所指示类型的正确数量的参数



(一个或两个)。

其他子句指定可选的运算符优化子句。适当时应提供这些子句，以加快使用运算符的查询的速度。但是，如果提供它们，则必须确保它们是正确的。错误地使用优化子句可能导致服务器进程崩溃，错误的输出或其他意外结果。如果您不确定优化子句，则可以始终忽略该子句。

为了能够创建运算符，您必须对参数类型和返回类型具有USAGE特权，并且对基础函数具有EXECUTE特权。如果指定了换向或负运算符，则您必须拥有这些运算符。

参数

name

要定义的运算符的名称（可以由模式指定）。如果同一模式中的两个运算符对不同的数据类型进行运算，则它们可以具有相同的名称。

funcname

用于实现此运算符的函数（必须为IMMUTABLE函数）。

lefttype

运算符的左操作数的数据类型（如果有）。左一元运算符将忽略此选项。

righttype

运算符的右操作数的数据类型（如果有）。右一元运算符将忽略此选项。

com_op

可选的COMMUTATOR子句命名一个运算符，该运算符是要定义的运算符的交换器。我们说如果对于所有可能的输入值 x , y , $(x A y)$ 等于 $(y B x)$ ，则算符A是算符B的交换子。请注意，B也是A的交换子。例如，特定数据类型的运算符<和>通常是彼此的交换子，而运算符+通常是与自身交换的。但是运算符-通常不与任何东西互换。可交换运算符的左操作数类型与其交换器的右操作数类型相同，反之亦然。因此，只需在COMMUTATOR子句中提供交换器运算符的名称即可。

neg_op

可选的NEGATOR子句命名一个运算符，该运算符是要定义的运算符的否定符。我们说如果两个操作符都返回布尔结果，并且对于所有可能的输入 x , y , $(x A y)$ 等于NOT $(x B y)$ ，则运算符A是运算符B的否定符。请注意，B也是A的否定符。例如，<和>=是大多数数据类型的否定符对。运算符的否定符必须具有与要定义的运算符相同的左右操作数类型，因此，在NEGATOR子句中仅需要指定运算符名称。

res_proc

可选的RESTRICKT为运算符命名了一个限制选择性估计函数。

请注意，这是一个函数名称，而不是运算符名称。

RESTRICKT子句仅对返回布尔值的二元运算符有意义。限制选择性估计器背后的想法是猜测表中哪几部分行将满足以下形式的WHERE子句条件：

```
column OP constant
```

用于当前运算符和特定的常量值。这有助于优化器了解具有这种形式的WHERE子句将消除多少行。

You can usually just use one of the following system standard estimator functions for many of your own operators:

`eqsel` 为 =

`neqsel` 为 <>

`scalarltsel` 为 < 或 <=

`scalargtsel` 为 > 或 >=

join_proc

可选的JOIN子句为运算符命名联接选择性估计函数。请注意，这是一个函数名称，而不是运算符名称。JOIN子句仅对返回布尔值的二元运算符有意义。连接选择性估计器背后的想法是猜测一对表中的哪几部分行将满足以下WHERE子句条件

```
table1.column1 OP table2.column2
```

对于当前的运算符。通过让优化器找出几个可能的连接序列中哪一个可能花费最少的工作，可以帮助优化器。

通常，您可以对许多自己的运算符使用以下系统标准联接选择性估计器函数之一：

`eqjoinsel` 为 =

`neqjoinsel` 为 <>

`scalarltjoinsel` 为 < 或 <=

`scalargtjoinsel` 为 > 或 >=

`areajoinsel` 为基于2D区域的比较

`positionjoinsel` 为基于2D位置的比较

cont join sel 为基于2D包含的比较

HASHES

可选的HASHES子句告诉系统允许使用哈希联接方法进行基于此运算符的联接。 HASHES仅对返回布尔值的二元运算符有意义。

哈希联接运算符只能对哈希到同一哈希代码的一对左右值返回true。如果将两个值放在不同的哈希存储桶中，则联接将永远不会比较它们，隐式地假设联接运算符的结果必须为false。因此，为不表示相等性的运算符指定HASHES永远没有意义。

在大多数情况下，仅在两侧采用相同数据类型的运算符支持散列。但是，您可以为两个或多个数据类型设计兼容的哈希函数，即使这些值的表示方式不同，这些函数也会为“相等”值生成相同的哈希码。

要标记为HASHES，联接运算符必须出现在哈希索引运算符类中。如果不存在此类运算符类，则尝试在哈希联接中使用运算符将在运行时失败。系统需要运算符类为运算符的输入数据类型查找特定于数据类型的哈希函数。在创建运算符类之前，还必须提供合适的哈希函数。在准备散列函数时，请格外小心，因为存在与机器相关的方法，散列函数可能无法正确运行。例如，在满足IEEE浮点标准的计算机上，负零和正零是不同的值（不同的位模式），但被定义为相等。如果浮点值可以包含负零，则将其定义为生成与正零相同的哈希值。

可哈希连接的运算符必须具有出现在同一运算符族中的换向符（如果两个操作数数据类型相同，则为自身，或者如果两个运算符数据类型不同，则为相关的相等运算符）。否则，使用运算符时可能会发生优化器错误。为了更好的优化，支持多种数据类型的哈希运算符族应为数据类型的每种组合提供相等运算符。
Note: 哈希可连接运算符的基础函数必须标记为不可变或稳定的；标记为volatile的运算符将不会使用。如果可哈希连接的运算符具有标记为严格的基础函数，则该函数也必须是完整的，对于任何两个非空输入都返回true或false，并且不返回null。

MERGES

MERGES子句（如果存在）告诉系统允许使用merge-join方法基于该运算符进行联接。 MERGES仅对返回布尔值的二元运算符有意义，并且在实践中，该运算符必须表示某些数据类型或一对数据类型的相等性。

合并联接基于的想法是将左表和右表排序，然后并行扫描它们。这意味着这两种数据类型都必须能够完全排序，并且联接运算符必须是仅对按排序顺序位于相等位置的值对成功的运算符。实际上，这意味着联接运算符必须表现得像相等运算符。但是，您可以合并联接两种不同的数据类型，只要它们在逻辑上是兼容的。例如，smallint-versus-integer等价运算符是可合并连接的。仅需要将两种数据类型都放入逻辑兼容序列的排序运算符。

要标记为MERGES，联接运算符必须显示为btree索引运算符族的相等成员。创建运算符时不会强制执行此操作，因为引用运算符族直到稍后才存在。但是，除非可以找到匹配的运算符族，否则该运算符实际上不会用于合并联接。因此，MERGE标记可作为对优化器的建议，以寻找匹配的运算符族。

可合并连接的运算符必须具有出现在同一运算符族中的换向器。如果两个操作数数据类型相同，则其为数据本身；如果数据类型不同，则为相关的相等运算符。如果没有合适的换向器，则在使用运算符时会发生优化器错误。同样，尽管不是严格要求，但是支持多种数据类型的btree运算符族应该能够为数据类型的每种组合提供相等运算符；这样可以实现更好的优化。

Note: SORT1, SORT2, LTCMP和GTCMP以前用于指定与可合并连接运算符关联的排序运算符的名称。现在，通过查看B树运算符族可以找到有关关联运算符的信息。指定这些运算符中的任何一个都将被忽略，除非它将隐式地将MERGES设置为true。

注解

用于实现运算符的任何函数都必须定义为IMMUTABLE。

在CREATE OPERATOR中不能指定运算符的词法优先级，因为解析器的优先级行为是固定的。有关优先级的详细信息，请参见PostgreSQL文档中的[运算符优先级](#)。

使用[DROP OPERATOR](#)从数据库中删除用户定义的运算符。使用[ALTER OPERATOR](#)修改数据库中的运算符。

示例

假设我们已经创建了complex类型的定义，这是创建一个用于添加两个复数的运算符的示例。首先定义完成工作的函数，然后定义运算符：

```
CREATE FUNCTION complex_add(complex, complex)
RETURNS complex
AS 'filename', 'complex_add'
LANGUAGE C IMMUTABLE STRICT;
CREATE OPERATOR +
(leftarg = complex,
rightarg = complex,
procedure = complex_add,
commutator = +
);
```

要在查询中使用此运算符：

```
SELECT (a + b) AS c FROM test_complex;
```

兼容性

CREATE OPERATOR是一个Greenplum数据语言的扩展。 SQL标准不提供用户定义的运算符。

另见

[CREATE FUNCTION](#) , [CREATE TYPE](#) , [ALTER OPERATOR](#) , [DROP OPERATOR](#)

Parent topic: [SQL Command Reference](#)

CLASS

Greenplum数据库® 6.0文档

定义一个新的运算符类。

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLE

ALTER FOREIGN
DATA WRAPPER

ALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

概要

```
CREATE OPERATOR CLASS name [DEFAULT] FOR TYPE data_type
    USING index_method [ FAMILY family_name ] AS
        { OPERATOR strategy_number operator_name [ ( op_type,
op_type ) ] [ FOR SEARCH | FOR ORDER BY sort_family_name ]
        | FUNCTION support_number funcname (argument_type [ , ... ]
        )
        | STORAGE storage_type
        } [ , ... ]
```

描述

CREATE OPERATOR CLASS创建一个新的运算符类。 运算符类定义如何将特定数据类型与索引一起使用。 运算符类指定某些运算符将为此数据类型和此索引方法填充特定角色或策略。 当为索引列选择了运算符类时，运算符类还指定了索引方法要使用的支持过程。 在创建运算符类之前，必须定义运算符类使用的所有运算符和函数。 用于实现运算符类的任何函数都必须定义为IMMUTABLE。

CREATE OPERATOR CLASS目前不检查运算符类定义是否包括所有由索引方法所需的运算符和函数，也不检查运算符和函数是否构成一个一致的集合。 定义有效的运算符类是用户的责任。

您必须是超级用户才能创建运算符类。

参数

name

要定义的运算符类的名称（可以由模式指定）。  模式中的两个运算符类仅在用于不同的索引方法时才可以使用相同的名称。

DEFAULT

使运算符类成为其数据类型的默认运算符类。对于特定的数据类型和索引方法，最多可以使用一个运算符类作为默认值。

data_type

此运算符类的列数据类型。

index_method

此运算符类用于的索引方法的名称。选项为btree, bitmap和gist。

family_name

要添加此运算符类的现有运算符族的名称。如果未指定，则使用与运算符类相同的名称族（如果尚不存在，则创建它）。

strategy_number

与运算符类关联的运算符由策略编号标识，该策略编号用于标识每个运算符在其运算符类的上下文中的语义。例如，B树对键施加严格的排序，从小到大，因此对于B树来说，类似小于和大于或等于的运算符更有吸引力。可以将这些策略视为广义运算符。每个运算符类指定针对特定数据类型和索引语义解释的每个策略对应的实际运算符。每种索引方法的相应策略编号如下：

Table 1. B树和位图策略

操作	策略编号
小于	1
小于等于	2
等于	3
大于等于	4
大于	5

Table 2. GiST二维策略 (R树)

操作	策略编号
严格左边	1
不延伸到右边	2
覆盖	3
不延伸到左边	4
严格右边	5
相同	6
包含	7
被包含	8
不延伸到上边	9

严格下边	10
严格上边	11
不延伸到下边	12

sort_family_name

现有btree运算符族的名称（可以由模式指定），用于描述与排序运算符关联的排序顺序。

如果未指定FOR SEARCH或FOR ORDER BY，则默认为FOR SEARCH。

operator_name

与运算符类关联的运算符的名称（可以由模式指定）。

op_type

在OPERATOR子句中，运算符的操作数数据类型，或NONE表示左一元或右一元运算符。在正常情况下，与运算符类的数据类型相同时，可以省略操作数数据类型。

在FUNCTION子句中，如果与函数的输入数据类型（用于B树比较函数和哈希函数）或类的数据类型（用于B树排序支持函数以及GiST, SP-GiST和GIN运算符类中的所有函数）不同，则函数要支持的操作数数据类型。这些默认值是正确的，因此，除了B树排序支持函数旨在支持跨数据类型比较的情况下，无需在FUNCTION子句中指定*op_type*。

support_number

索引方法为了工作需要额外的支持例程。这些操作是索引方法在内部使用的管理例程。与策略一样，运算符类标识对于给定的数据类型和语义解释，哪些特定函数应扮演这些角色中的每一个。索引方法定义了所需的函数集，运算符类通过将它们分配给支持函数编号来标识要使用的正确函数，如下所示：

Table 3. B树和位图支持函数

函数	支持编号
比较两个键并返回小于零，零或大于零的整数，指示第一个键是否小于，等于或大于第二个键。	1

Table 4. GiST Support Functions

函数	支持编号
consistent - 确定键是否满足查询限定符。	1
union - 计算一组键的并集。	2
compress - 计算要索引的键或值的压缩表示形式。	3
decompress -	4

	计算压缩键的解压缩表示。	
	penalty - 计算使用给定子树的键将新键插入子树的代价。	5
	picksplit - 确定一个页面的条目将被移动到新的页面，并计算联合键结果页面。	6
	equal - 比较两个键，如果相等则返回true。	7

funcname

函数的名称（可以由模式指定），该函数是运算符类的索引方法支持过程。

argument_types

函数的参数数据类型。

storage_type

实际存储在索引中的数据类型。通常，这与列数据类型相同，但是某些索引方法（当前为GiST和GIN）允许其不同。除非索引方法允许使用其他类型，否则必须省略STORAGE子句。

注解

因为索引机制在使用函数之前不会检查对函数的访问权限，所以在运算符类中包括函数或运算符就等同于对其授予公共执行权限。对于在运算符类中有用的各种函数，这通常不是问题。

运算符不应由SQL函数定义。SQL函数很可能内联到调用查询中，这将阻止优化器识别查询与索引匹配。

用于实现运算符类的任何函数都必须定义为IMMUTABLE。

在Greenplum数据库6.0之前，OPERATOR子句可以包含RECHECK选项。此选项不再支持。Greenplum数据库现在可以确定索引运算符是否在运行时是“有损的”。这允许在一个运算符可能是或者可能不是有损的情况下有效地处理。

示例

以下示例命令为数据类型_int4（int4的数组）定义了GiST索引运算符类。有关完整的示例，请参见intarray contrib模块。

```
CREATE OPERATOR CLASS gist_int_ops
  DEFAULT FOR TYPE _int4 USING gist AS
    OPERATOR 3 &&,
```

```
OPERATOR 6 = (anyarray, anyarray),
OPERATOR 7 @>,
OPERATOR 8 <@,
OPERATOR 20 @@ (_int4, query_int),
FUNCTION 1 g_int_consistent (internal, _int4, int,
oid, internal),
FUNCTION 2 g_int_union (internal, internal),
FUNCTION 3 g_int_compress (internal),
FUNCTION 4 g_int_decompress (internal),
FUNCTION 5 g_int_penalty (internal, internal,
internal),
FUNCTION 6 g_int_picksplit (internal, internal),
FUNCTION 7 g_int_same (_int4, _int4, internal);
```

兼容性

CREATE OPERATOR CLASS是Greenplum数据库扩展。 SQL标准中没有CREATE OPERATOR CLASS语句。

另见

[ALTER OPERATOR CLASS](#) , [DROP OPERATOR CLASS](#) , [CREATE FUNCTION](#)

Parent topic: [SQL Command Reference](#)

FAMILY

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLE

ALTER FOREIGN
DATA WRAPPER

ALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一个新的运算符族。

概要

```
CREATE OPERATOR FAMILY name USING index_method
```

描述

CREATE OPERATOR FAMILY创建一个新的运算符族。 运算符族定义了一组相关的运算符类，以及可能与这些运算符类兼容但对于任何单个索引的函数而言并非必不可少的一些其他运算符和支持函数。（对于索引必不可少的运算符和函数应该在相关的运算符类中分组，而不是“松散地”在运算符族中。通常，单数据类型的运算符绑定到运算符类，而跨数据类型的运算符在包含两种数据类型的运算符类的运算符族中可能会比较松散。）

新的运算符族最初是空的。 应该通过发出后续的CREATE OPERATOR CLASS命令来添加所包含的运算符类，并可选地通过ALTER OPERATOR FAMILY命令来添加“松散”运算符及其相应的支持函数来填充该表。

如果指定了模式名称，则会在指定的模式中创建运算符族。否则，它将在当前模式中创建。同一模式中的两个运算符族只有用于不同索引方法的名称才能相同。

定义运算符族的用户将成为其所有者。当前，创建用户必须是超级用户。（之所以做出此限制，是因为错误的运算符族定义可能会使服务器混乱甚至崩溃。）

参数

name

要定义的运算符族的名称（可以由模式指定）。
index_method



该运算符族所针对的索引方法的名称。

兼容性

CREATE OPERATOR FAMILY是Greenplum数据库扩展。 SQL标准中没有CREATE OPERATOR FAMILY语句。

另见

[ALTER OPERATOR FAMILY](#) , [DROP OPERATOR FAMILY](#) , [CREATE FUNCTION](#) , [ALTER OPERATOR CLASS](#) , [CREATE OPERATOR CLASS](#) , [DROP OPERATOR CLASS](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

注册自定义数据访问协议，该协议可以在定义Greenplum数据库外部表时指定。

概要

```
CREATE [TRUSTED] PROTOCOL name (
    [readfunc='read_call_handler'] [, 
    writefunc='write_call_handler'] [, 
    validatorfunc='validate_handler' ])
```

描述

CREATE PROTOCOL将数据访问协议名称与负责从外部数据源读取数据和将数据写入外部数据源的调用处理程序相关联。您必须是超级用户才能创建协议。

CREATE PROTOCOL命令必须指定读调用处理程序或写调用处理程序。必须在数据库中定义CREATE PROTOCOL命令中指定的调用处理程序。

可以在CREATE EXTERNAL TABLE命令中指定协议名称。

有关创建和启用自定义数据访问协议的信息，请参阅Greenplum数据库管理员指南中的“示例自定义数据访问协议”。

参数

TRUSTED

如果未指定，则只有超级用户和协议所有者才能使用协议创建外部表。如果指定，超级用户和协议所有者可以将协议的许可权限授予其他数据库角色。

name

数据访问协议的名称。协议名称区分大小写。该名
 数据库中的协议之间必须唯一。

readfunc='read_call_handler'

Greenplum数据库调用以从外部数据源读取数据的先前注册函数

的名称。该命令必须指定读调用处理程序或写调用处理程序。
writefunc= '*write_call_handler*'

Greenplum数据库调用以将数据写入外部数据源时调用的先前注册函数的名称。该命令必须指定读调用处理程序或写调用处理程序。

validatorfunc=' *validate_handler*'

可选的验证器函数，用于验证CREATE EXTERNAL TABLE命令中指定的URL。

注解

Greenplum数据库在内部处理类型为file, gpfdist和gpfdists的外部表。有关启用s3协议的信息，请参见[配置和使用S3外部表](#)。有关配置PXF扩展和使用pxf协议的信息，请参阅[Greenplum平台扩展框架 \(PXF\)](#) 文档。

任何实现数据访问协议的共享库都必须位于所有Greenplum数据库segment主机上的同一位置。例如，共享库可以位于所有主机上的操作系统环境变量LD_LIBRARY_PATH指定的位置。定义处理程序函数时，也可以指定位置。例如，当您在CREATE PROTOCOL命令中定义s3协议时，您将\$libdir/gps3ext.so指定为共享库的位置，其中\$libdir位于\$GPHOME/lib。

兼容性

CREATE PROTOCOL是Greenplum数据库扩展。

另见

[ALTER PROTOCOL](#), [CREATE EXTERNAL TABLE](#), [DROP PROTOCOL](#),
[GRANT](#)

Parent topic: [SQL Command Reference](#)

GROUP

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一个新的资源组。

概要

```
CREATE RESOURCE GROUP name WITH (group_attribute=value [ , ... ])
```

其中*group_attribute*是：

```
CPU_RATE_LIMIT=integer | CPUSET=tuple  
MEMORY_LIMIT=integer  
[ CONCURRENCY=integer ]  
[ MEMORY_SHARED_QUOTA=integer ]  
[ MEMORY_SPILL_RATIO=integer ]  
[ MEMORY_AUDITOR= {vmtracker | cgroup} ]
```

描述

为Greenplum数据库资源管理创建一个新的资源组。 您可以创建资源组来管理角色资源或管理Greenplum数据库外部组件（例如PL/Container）的资源。

启用资源组后，您创建的用于管理用户角色的资源组将标识该角色的并发事务，内存和CPU限制。 您可以将此类资源组分配给一个或多个角色。

您创建的用于管理Greenplum数据库外部组件（例如PL/Container）的资源的资源组会在启用资源组时标识该组件的内存和CPU限制。 这些资源组使用cgroup进行CPU和内存管理。 将资源组分配给外部组件是特定于组件的。 例如，当您配置PL/Container运行时时，您将分配一个PL/Container资源组。 您不能将为外部组件创建的资源组分配给角色，也不能将为角色创建的资源组分配给外部组件。

您必须具有SUPERUSER特权才能创建资源组。 Greenplum数据库群集中允许的最大资源组数为100。

Greenplum数据库预定义了两个默认资源



居库群集

组: `admin_group`和`default_group`。这些组名以及组名`none`均保留。

要为资源组设置适当的限制, Greenplum数据库管理员必须熟悉通常在系统上执行的查询, 以及执行这些查询的用户/角色以及他们可能使用的外部组件, 例如PL/Containers。

创建资源组的角色后, 组分配给使用`ALTER ROLE`或`CREATE ROLE`命令一个或多个角色。

创建资源组以管理外部组件的CPU和内存资源后, 将外部组件配置为使用该资源组。例如, 配置PL/Container运行时`resource_group_id`。

参数

name

资源组的名称。

`CONCURRENCY integer`

此资源组允许的最大并发事务数, 包括活动和空闲事务。

`CONCURRENCY`值必须是[0 .. `max_connections`]范围内的整数。为角色定义的资源组的默认`CONCURRENCY`值为20。

对于为外部组件创建的资源组, 必须将`CONCURRENCY`设置为零(0)。

Note: 您不能将`admin_group`的`CONCURRENCY`值设置为零(0)。

`CPU_RATE_LIMIT integer`

`CPUSSET tuple`

需要。创建资源组时, 只能指定`CPU_RATE_LIMIT`或`CPUSSET`之一。

`CPU_RATE_LIMIT`是分配给该资源组的CPU资源的百分比。您可以为资源组指定的最小CPU百分比为1。最大为100。

为Greenplum数据库集群中定义的所有资源组指定的`CPU_RATE_LIMIT`值的总和必须小于或等于100。

`CPUSSET`标识要为此资源组保留的CPU核心。您在`tuple`中指定的CPU核心必须在系统中可用, 并且不能与您为其他资源组指定的任何CPU核心重叠。

`tuple`是用逗号分隔的单个核心号或核心号间隔的列表。您必须将元组用单引号引起来, 例如'1,3-4'。

Note: 仅在为Greenplum数据库集群启用基于资源组的资源管理之后, 才能为资源组配置`CPUSSET`。

`MEMORY_LIMIT integer`

需要。分配给该资源组的Greenplum数据库内存资源的总百分

比。 您可以为资源组指定的最小内存百分比为1。 最大为100。 为Greenplum数据库集群中定义的所有资源组指定的MEMORY_LIMIT值的总和必须小于或等于100。

MEMORY_SHARED_QUOTA *integer*

资源组中共享内存的配额。 具有MEMORY_SHARED_QUOTA阈值的资源组预留了分配给资源组以在事务之间共享的内存的百分比。 共享内存将按先到先得的原则分配。 事务可能不使用任何，部分或全部内存。 您可以为资源组指定的最小内存共享配额百分比为0。 最大为100。 默认的MEMORY_SHARED_QUOTA值为20。

MEMORY_SPILL_RATIO *integer*

事务中内存密集型运算的内存使用量阈值。 当达到此阈值时，事务会溢出到磁盘。 您可以为资源组指定的最小内存溢出率百分比是0。 最大值是100。 默认的MEMORY_SPILL_RATIO值为20。

MEMORY_AUDITOR {vmtracker | cgroup}

资源组的内存审核员。 Greenplum数据库对角色资源使用虚拟内存跟踪，对外部组件使用的资源使用cgroup内存跟踪。 默认的MEMORY_AUDITOR为vmtracker。 使用vmtracker内存审核创建资源组时，Greenplum数据库在内部跟踪该资源组的内存。

当您创建指定cgroup MEMORY_AUDITOR的资源组时，Greenplum数据库会将该资源组使用的内存记账推迟到cgroup。 对于您为外部组件（例如PL/Container）创建的资源组，CONCURRENCY必须为零（0）。 您不能将为外部组件创建的资源组分配给Greenplum数据库角色。

注解

您不能在显式事务或子事务中提交CREATE RESOURCE GROUP命令。

使用gp_toolkit.gp_resgroup_config系统视图显示所有资源组的限制设置：

```
SELECT * FROM gp_toolkit.gp_resgroup_config;
```

示例

创建一个CPU和内存限制百分比为35的资源组：

```
CREATE RESOURCE GROUP rgroup1 WITH (CPU_RATE_LIMIT=35,  
MEMORY_LIMIT=35);
```

创建一个资源组，其并发事务限制为20，内存限制为15，CPU限制为25：

```
CREATE RESOURCE GROUP rgroup2 WITH (CONCURRENCY=20,  
MEMORY_LIMIT=15, CPU_RATE_LIMIT=25);
```

创建一个资源组来管理PL/Container资源，指定内存限制为10，CPU限制为10：

```
CREATE RESOURCE GROUP plc_run1 WITH (MEMORY_LIMIT=10,  
CPU_RATE_LIMIT=10,  
CONCURRENCY=0, MEMORY_AUDITOR=cgroup);
```

创建一个内存限制百分比为11的资源组，并为其分配CPU核心1至3：

```
CREATE RESOURCE GROUP rgroup3 WITH (CPUSET='1-3',  
MEMORY_LIMIT=11);
```

兼容性

CREATE RESOURCE GROUP是Greenplum数据扩展。在SQL标准中没有资源组或资源管理。

另见

[ALTER ROLE](#), [CREATE ROLE](#), [ALTER RESOURCE GROUP](#), [DROP RESOURCE GROUP](#)

Parent topic: [SQL Command Reference](#)

QUEUE

Greenplum数据库® 6.0文档

定义一个新的资源队列。

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT

PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

概要

```
CREATE RESOURCE QUEUE name WITH (queue_attribute=value [, ...])
```

其中*queue_attribute*为：

```
ACTIVE_STATEMENTS=integer
  [ MAX_COST=float [ COST_OVERCOMMIT={TRUE|FALSE} ] ]
  [ MIN_COST=float ]
  [ PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX} ]
  [ MEMORY_LIMIT='memory_units' ]

  | MAX_COST=float [ COST_OVERCOMMIT={TRUE|FALSE} ]
    [ ACTIVE_STATEMENTS=integer ]
    [ MIN_COST=float ]
    [ PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX} ]
    [ MEMORY_LIMIT='memory_units' ]
```

描述

为Greenplum数据库资源管理创建一个新的资源队列。 资源队列必须具有ACTIVE_STATEMENTS或MAX_COS值（或者可以同时具有两者）。 只有超级用户才能创建资源队列。

具有ACTIVE_STATEMENT\$阈值的资源队列为分配给该队列的角色可以执行的查询数量设置了最大限制。 它控制允许同时运行的活动查询的数量。 ACTIVE_STATEMENT\$的值应为大于0的整数。

具有MAX_COS阈值的资源队列对可以由分配给该队列的角色执行的查询的总成本设置最大限制。 成本以Greenplum数据库查询优化器确定的查询的估计总成本来衡量（如查询的EXPLAIN输出中所示）。 管理员必须熟悉通常在系统上执行的查询，以便为队列设置适当的MAX_COS阈值。 成本以获取磁盘页面为单位进行衡量；1.0等于读取一个连续的磁盘页面。 MAX_COS的值指定为浮点数（例如100.0），也可以指定为指数（例如1e+2）。 如果基于成本阈值限制资源队列，则管理员可以允许COST_OVERCOMMIT=TRUE（默认值）。 这意味着将允许运行超过允

许的成本阈值的查询，但仅在系统空闲时才允许运行。如果指定COST_OVERCOMMIT=FALSE则超出成本限制的查询将始终被拒绝，并且永远不允许运行。通过为MIN_COST指定一个值，管理员可以为小型查询定义费用，这些费用将免于资源排队。

Note: GPORCA和Postgres查询优化器利用不同的查询成本模型，并且可以为同一查询计算不同的成本。Greenplum数据库资源队列资源管理方案既不区分GPORCA和Postgres优化器之间的成本，也没有对其进行调整。它使用从优化程序返回的文字成本值来限制查询。

当基于资源队列的资源管理处于活动状态时，请对资源队列使用MEMORY_LIMIT和ACTIVE_STATEMENTS限制，而不是配置基于成本的限制。即使使用GPORCA，Greenplum数据库也可能会退回到对某些查询使用Postgres查询优化器，因此使用基于成本的限制可能会导致意外结果。

如果未为ACTIVE_STATEMENTS或MAX_COST定义值，则默认情况下将其设置为-1（表示无限制）。定义资源队列后，必须使用ALTER ROLE或CREATE ROLE命令将角色分配给队列。

您可以选择将PRIORITY分配给资源队列，以控制与该队列相关联的查询相对于其他资源队列使用的可用CPU资源的相对份额。如果未为PRIORITY定义值，则与队列关联的查询的默认优先级为MEDIUM。

具有可选MEMORY_LIMIT阈值的资源队列设置了对通过资源队列提交的所有查询可以在segment主机上使用的最大内存量的限制。这就决定了一个查询的所有工作进程可以在查询执行期间的segment主机上消耗的存储器的总量。Greenplum建议将MEMORY_LIMIT与ACTIVE_STATEMENTS结合使用，而不要与MAX_COST结合使用。在基于语句的队列上，每个查询分配的默认内存量为：MEMORY_LIMIT / ACTIVE_STATEMENTS。在基于成本的队列上，每个查询分配的默认内存量为：MEMORY_LIMIT * (query_cost / MAX_COST)。

如果不超过MEMORY_LIMIT或statement_mem，则可以用statement_mem服务器配置参数在每个查询的基础上覆盖默认内存分配。例如，为一个特定的查询分配更多的内存：

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

您所有资源队列的MEMORY_LIMIT值不应超过segment主机的物理内存量。如果工作负载分散在多个队列中，则内存分配可能会超额预定。但是，如果超出了gp_vmem_protect_limit中指定的segment主机内存限制，则可以在执行期间取消查询。

有关statement_mem，max_statement 和gp_vmem_protect_limit 的信息，请参阅[服务器配置参数](#)。

参数

name

资源队列的名称。

ACTIVE_STATEMENTS *integer*

具有ACTIVE_STATEMENTS阈值的资源队列限制了分配给该队列的角色可以执行的查询数量。它控制允许同时运行的活动查询的数量。ACTIVE_STATEMENTS的值应为大于0的整数。

MEMORY_LIMIT '*memory_units*'

设置此资源队列中用户提交的所有语句的总内存配额。可以用kB, MB或GB指定存储单元。资源队列的最小内存配额为10MB。没有最大值，但是查询执行时的上限受segment主机的物理内存限制。默认值为无限制 (-1)。

MAX_COST *float*

具有MAX_COST阈值的资源队列对可以由分配给该队列的角色执行的查询的总成本设置了最大限制。成本以Greenplum数据库查询优化器确定的查询的估算总成本来衡量（如查询的EXPLAIN输出中所示）。因此，管理员必须熟悉通常在系统上执行的查询，以便为队列设置适当的成本阈值。成本以获取磁盘页面为单位进行衡量；1.0等于读取一个连续的磁盘页面。MAX_COST的值指定为浮点数（例如100.0），也可以指定为指数（例如1e+2）。

COST_OVERCOMMIT *boolean*

如果基于MAX_COST限制资源队列，则管理员可以允许COST_OVERCOMMIT（默认值）。这意味着将允许运行超过允许的成本阈值的查询，但仅在系统空闲时才允许运行。如果指定COST_OVERCOMMIT=False则超出成本限制的查询将始终被拒绝，并且永远不允许运行。

MIN_COST *float*

小型查询的最低查询成本限制。费用低于此限制的查询将不会排队并立即运行。成本以Greenplum数据库查询优化器确定的查询的估计总成本来衡量（如查询的EXPLAIN输出中所示）。因此，管理员必须熟悉通常在系统上执行的查询，以便为被认为是小的查询设置适当的成本。成本以获取磁盘页面为单位进行衡量；1.0等于读取一个连续的磁盘页面。MIN_COST的值指定为浮点数（例如100.0），也可以指定为指数（例如1e+2）。

PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX}

设置与资源队列关联的查询的优先级。发生竞争时，具有较高优先级的队列中的查询或语句将获得更多的可用CPU资源份额。在执行较高优先级的查询时，低优先级队列中的查询可能会延迟。如果未指定优先级，则与队列关联的查询的优先级为MEDIUM。

注解

使用gp_toolkit.gp_resqueue_status
限制设置和当前状态：

系统视图查看资源队列的限

```
SELECT * from gp_toolkit.gp_resqueue_status WHERE
rsqname='queue_name';
```

还有另一个名为pg_stat_resqueues 的系统视图，该视图显示了一段时间内资源队列的统计指标。但是，要使用此视图，必须启用stats_queue_level 服务器配置参数。有关使用资源队列的更多信息，请参见Greenplum数据库管理员指南中的“管理工作负载和资源”。

CREATE RESOURCE QUEUE不能在事务中运行。

此外，在执行EXPLAIN ANALYZE命令的过程中运行的SQL语句将从资源队列中排除。

示例

创建一个活动查询限制为20的资源队列：

```
CREATE RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=20);
```

创建一个资源队列，其活动查询限制为20，总内存限制为2000MB（每个查询在执行时将分配100MB的segment主机内存）：

```
CREATE RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=20,
MEMORY_LIMIT='2000MB');
```

创建一个查询费用限制为3000.0的资源队列：

```
CREATE RESOURCE QUEUE myqueue WITH (MAX_COST=3000.0);
```

创建一个查询成本限制为 3^{10} （或30000000000.0）的资源队列，并且不允许过量使用。允许成本低于500的小型查询立即运行：

```
CREATE RESOURCE QUEUE myqueue WITH (MAX_COST=3e+10,
COST_OVERCOMMIT=FALSE, MIN_COST=500.0);
```

创建同时具有活动查询限制和查询成本限制的资源队列：

```
CREATE RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=30,  
MAX_COST=5000.00);
```

创建一个活动查询限制为5且最大优先级设置的资源队列：

```
CREATE RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=5,  
PRIORITY=MAX);
```

兼容性

CREATE RESOURCE QUEUE是Greenplum数据库扩展。 SQL标准中没有提供资源队列或资源管理的规定。

另见

[ALTER ROLE](#), [CREATE ROLE](#), [ALTER RESOURCE QUEUE](#), [DROP RESOURCE QUEUE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一个新的数据库角色（用户或组）。

概要

```
CREATE ROLE name [[WITH] option [ ... ]]
```

其中*option*可以是：

```
SUPERUSER | NOSUPERUSER
| CREATEDB | NOCREATEDB
| CREATEROLE | NOCREATEROLE
| CREATEUSER | NOCREATEUSER
| CREATEEXTTABLE | NOCREATEEXTTABLE
[ ( attribute='value'[, ...] ) ]
    where attributes and value are:
        type='readable' | 'writable'
        protocol='gpfdist' | 'http'
| INHERIT | NOINHERIT
| LOGIN | NOLOGIN
| REPLICATION | NOREPLICATION
| CONNECTION LIMIT connlimit
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
| VALID UNTIL 'timestamp'
| IN ROLE rolename [, ...]
| ROLE rolename [, ...]
| ADMIN rolename [, ...]
| USER rolename [, ...]
| SYSID uid [, ...]
| RESOURCE QUEUE queue_name
| RESOURCE GROUP group_name
| [ DENY deny_point ]
| [ DENY BETWEEN deny_point AND deny_point ]
```

描述

CREATE ROLE在Greenplum数据库系统中添加了新角色。角色是可以拥有数据库对象并具有数据库特权的实体。根据角色的使用方式，可以将角色视为用户，组或两者。您必须具有CREATEROLE特权或是数据库超级用户才能使用此命令。

请注意，角色是在系统级别定义的，并且对于Greenplum数据库系统中的所有数据库均有效。



参数

name

新角色的名称。

SUPERUSER

NOSUPERUSER

如果指定了SUPERUSER，则定义的角色将是超级用户，该超级用户可以覆盖数据库中的所有访问限制。超级用户状态很危险，应仅在真正需要时使用。您必须自己是超级用户才能创建新的超级用户。默认值为NOSUPERUSER。

CREATEDB

NOCREATEDB

如果指定了CREATEDB，将允许所定义的角色创建新数据库。

NOCREATEDB（默认值）将使角色无法创建数据库。

CREATEROLE

NOCREATEROLE

如果指定了CREATEROLE，则允许定义的角色创建新角色，更改其他角色和删除其他角色。NOCREATEROLE（默认值）将拒绝角色创建新角色或修改其他角色。

CREATEUSER

NOCREATEUSER

这些子句已经过时，但仍然被SUPERUSER和NOSUPERUSER的拼写接受。请注意，它们不等同于CREATEROLE和NOCREATEROLE子句。

CREATEEXTTABLE

NOCREATEEXTTABLE

如果指定了CREATEEXTTABLE，则允许定义的角色创建外部表。如果未指定，则默认type为readable，默认protocol为gpfdist。有效类型为gpfdist, gpfdists, http和https。

NOCREATEEXTTABLE（默认类型）拒绝该角色创建外部表。请注意，使用file或execute协议的外部表只能由超级用户创建。

使用GRANT...ON PROTOCOL命令允许用户创建和使用具有自定义协议类型的外部表，包括Greenplum数据库附带的s3和pxf协议。

INHERIT

NOINHERIT

如果指定了该属性，则INHERIT（默认设置）允许该角色使用为其直接或间接所属的所有角色授予的任何数据库特权。使用NOINHERIT时，另一个角色的成员资格仅授予SET ROLE权

限给该另一个角色。

LOGIN

NOLOGIN

如果指定，则LOGIN允许角色登录数据库。可以将具有LOGIN属性的角色视为用户。具有NOLOGIN的角色对于管理数据库特权很有用，并且可以视为组。如果未指定，则NOLOGIN为默认值，除非CREATE ROLE通过其替代拼写CREATE USER被调用时。

REPLICATION

NOREPLICATION

这些子句确定是允许角色启动流复制还是使系统进入和退出备份模式。具有REPLICATION属性的角色是具有很高特权的角色，并且仅应在实际用于复制的角色上使用。如果未指定，则NOREPLICATION是默认值。

CONNECTION LIMIT *connlimit*

此角色可以建立的并发连接的最大数量。默认值-1表示没有限制。

PASSWORD *password*

使用LOGIN属性设置角色的用户密码。如果您不打算使用密码身份验证，则可以忽略此选项。如果未指定密码，则密码将设置为null，并且该用户的密码身份验证将始终失败。空密码可以选择地显式写为PASSWORD NULL。

ENCRYPTED

UNENCRYPTED

这些关键字控制密码是否以加密方式存储在系统catalog中。

(如果未指定，则默认行为由配置参数`password_encryption`决定。) 如果显示的密码字符串已经采用MD5加密格式，则将按原样存储加密，而不管是否指定了ENCRYPTED或UNENCRYPTED (因为系统无法解密指定的加密密码字符串)。这允许在转储/还原期间重新加载加密的密码。

VALID UNTIL '*timestamp*'

VALID UNTIL子句设置日期和时间，之后该角色的密码将不再有效。如果省略此子句，密码将永不过期。

IN ROLE *rolename*

将新角色添加为命名角色的成员。请注意，没有任何选项可以将新角色添加为管理员。使用单独的GRANT命令来执行此操作。

ROLE *rolename*

将命名角色添加为该角色的成员，从而使该新角色成为一个组。

ADMIN *rolename*

ADMIN子句类似于ROLE，但是被提及的角色被使用WITH ADMIN OPTION加入到新角色中，从而赋予他们将这个角色的成员资格授予其他人的权利。

RESOURCE GROUP *group_name*

要分配给新角色的资源组的名称。该角色将受限于资源组配置的并发事务，内存和CPU限制。您可以将一个资源组分配给一个或多个角色。

如果未为新角色指定资源组，则会自动为该角色分配角色的默认资源组，为SUPERUSER角色分配admin_group，为非管理员角色分配default_group。

您可以将admin_group资源组分配给具有SUPERUSER属性的任何角色。

您可以将default_group资源组分配给任何角色。

您不能将为外部组件创建的资源组分配给角色。

RESOURCE QUEUE *queue_name*

新用户级别角色将分配到的资源队列的名称。只能将具有LOGIN特权的角色分配给资源队列。特殊关键字NONE表示将角色分配给默认资源队列。一个角色只能属于一个资源队列。

具有SUPERUSER属性的角色不受资源队列限制。对于超级用户角色，无论分配的资源队列施加什么限制，查询总是立即运行。

DENY *deny_point***DENY BETWEEN *deny_point* AND *deny_point***

DENY和DENY BETWEEN关键字设置在登录时强制执行的基于时间的约束。DENY设置拒绝访问的日期或日期和时间。DENY BETWEEN设置一个拒绝访问的时间间隔。两者都使用具有以下格式的参数*deny_point*:

```
DAY day [ TIME 'time' ]
```

*deny_point*参数的两个部分使用以下格式:

对于day:

```
{ 'Sunday' | 'Monday' | 'Tuesday' | 'Wednesday' |
  'Thursday' | 'Friday' |
  'Saturday' | 0-6 }
```

对于time:

```
{ 00-23 : 00-59 | 01-12 : 00-59 { AM | PM } }
```

DENY BETWEEN子句使用两个*deny_point*参数:

```
DENY BETWEEN deny_point AND deny_point
```

有关基于时间的约束的更多信息和示例，请参阅Greenplum数据库管理员指南中的“管理角色和特权”。

注解

添加和删除角色成员（管理组）的首选方法是使用[GRANT](#)和[REVOKE](#)。

VALID UNTIL子句仅为密码而不是角色定义过期时间。 使用非基于密码的身份验证方法登录时，不会强制使用到期时间。

`INHERIT`属性控制可授予特权（数据库对象和角色成员的访问特权）的继承。 它不适用于由`CREATE ROLE`和`ALTER ROLE`设置的特殊角色属性。 例如，即使设置了`INHERIT`，具有`CREATEDB`特权的角色成员也不会立即授予创建数据库的能力。 这些特权/属性永远不会被继承：`SUPERUSER`, `CREATEDB`, `CREATEROLE`, `CREATEEXTTABLE`, `LOGIN`, `RESOURCE GROUP`和`RESOURCE QUEUE`。 必须在每个用户级角色上设置属性。

由于向后兼容，`INHERIT`属性是默认属性。 在以前的Greenplum数据库版本中，用户始终可以访问其所属组的所有特权。 但是，`NOINHERIT`提供与SQL标准中指定的语义更接近的匹配。

使用`CREATEROLE`特权时要小心。 对于`CREATEROLE-role`的特权，没有继承的概念。 这意味着，即使一个角色没有特定的特权，但被允许创建其他角色，它也可以轻松地创建另一个角色，而该角色的特权不同于其自己的角色（创建具有超级用户特权的角色除外）。 例如，如果角色具有`CREATEROLE`特权，但没有`CREATEDB`特权，则它可以用`CREATEDB`特权创建新角色。 因此，将具有`CREATEROLE`特权的角色视为几乎超级用户角色。

超级用户绝不执行`CONNECTION LIMIT`选项。

使用此命令指定未加密的密码时必须小心。 密码将以明文形式传输到服务器，并且也可能会记录在客户端的命令历史记录或服务器日志中。 但是，客户端程序`createuser`传输加密的密码。 另外，`psql`包含命令`\password`，可用于稍后安全地更改密码。

示例

创建一个可以登录但不提供密码的角色：

```
CREATE ROLE jonathan LOGIN;
```

创建一个属于资源队列的角色：

```
CREATE ROLE jonathan LOGIN RESOURCE QUEUE poweruser;
```

使用有效期至2016年底的密码创建角色（CREATE USER与CREATE ROLE相同，只不过它暗含了LOGIN）：

```
CREATE USER joelle WITH PASSWORD 'jw8s0F4' VALID UNTIL
'2017-01-01';
```

创建一个可以创建数据库并管理其他角色的角色：

```
CREATE ROLE admin WITH CREATEDB CREATEROLE;
```

创建一个角色，该角色在星期日不允许登录访问：

```
CREATE ROLE user3 DENY DAY 'Sunday';
```

创建一个可以创建类型为'gpfdist'的可读可写外部表的角色：

```
CREATE ROLE jan WITH CREATEEXTTABLE(type='readable',
protocol='gpfdist')
CREATEEXTTABLE(type='writable', protocol='gpfdist');
```

创建一个角色，分配一个资源组：

```
CREATE ROLE bill RESOURCE GROUP rg_light;
```

兼容性

SQL标准定义了用户和角色的概念，但是将它们视为不同的概念，并将所有定义用户的命令留给数据库实现指定。在Greenplum数据库中，用户和角色被统一为单一类型的对象。因此，角色具有比标准中更多的可选属性。

CREATE ROLE在SQL标准中，但是该标准仅需要以下语法：

```
CREATE ROLE name [WITH ADMIN rolename]
```

数据库扩展是允许多个初始管理员，以及
所有其他选项。

通过为用户提供NOINHERIT属性，而为角色赋予INHERIT属性，可
以最接近地逼近SQL标准指定的行为。

另见

[SET ROLE](#), [ALTER ROLE](#), [DROP ROLE](#), [GRANT](#), [REVOKE](#), [CREATE RESOURCE QUEUE](#) [CREATE RESOURCE GROUP](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义新的重写规则。

概要

```
CREATE [OR REPLACE] RULE name AS ON event
    TO table_name [WHERE condition]
    DO [ALSO | INSTEAD] { NOTHING | command | (command;
command
    ...) }
```

描述

CREATE RULE定义了应用于指定表或视图的新规则。CREATE OR REPLACE RULE将创建一个新规则，或为同一表替换一个具有相同名称的现有规则。

Greenplum数据库规则系统允许您定义对数据库表中的插入，更新或删除执行的另一种操作。当执行给定表上的给定命令时，规则将导致执行附加或替代命令。INSTEAD规则可以用另一个命令替换给定命令，或导致根本不执行命令。规则也可以用于实现SQL视图。重要的是要认识到规则实际上是命令转换机制或命令宏。转换发生在命令执行开始之前。它不会像触发器那样针对每个物理行独立运行。

ON SELECT规则必须是无条件的INSTEAD规则，并且必须具有由单个SELECT命令组成的操作。因此，ON SELECT规则有效地将表变成视图，其可见内容是规则的SELECT命令返回的行，而不是表中存储的内容（如果有的话）。与创建真实表并为其定义ON SELECT规则相比，写CREATE VIEW命令被认为是更好的样式。

您可以通过定义ON INSERT，ON UPDATE和ON DELETE规则将视图上的更新操作替换为其他表上的适当更新，来创建可更新视图的错觉。如果要支持INSERT RETURNING等，请确保在每个规则中放入合适的RETURNING子句。

如果尝试使用条件规则进行视图更新，则有一个陷阱： 在视图上执行的每个操作都必须有一个无条件的INSTEAD规则。如果规则是有条件的，或者不是INSTEAD，则系统仍将拒绝执行更新操作的尝试，因为它认为在某些情况下可能最终尝试对视图的虚拟表执行操

作。如果要处理条件规则中的所有有用情况，请添加无条件的DO INSTEAD NOTHING规则，以确保系统理解它将永远不会被调用它来更新虚拟表。然后将条件规则设为non-INSTEAD；在应用它们的情况下，它们会添加到默认的INSTEAD NOTHING操作中。（但是，该方法当前不适用于支持RETURNING查询。）

Note:

这是很简单的自动更新的视图（参见[CREATE VIEW](#)）以可更新并不需要用户创建的规则。尽管您仍然可以创建显式规则，但是自动更新转换通常会胜过显式规则。

参数

name

要创建的规则的名称。此名称必须与同一表的任何其他规则的名称不同。同一表和同一事件类型上的多个规则以字母名称顺序应用。

event

该事件是SELECT, INSERT, UPDATE或DELETE之一。

table_name

规则所适用的表或视图的名称（可以由模式指定）。

condition

任何SQL条件表达式（返回布尔值）。条件表达式可能不引用除NEW和OLD以外的任何表，并且可能不包含聚合函数。

NEW和OLD引用参考表中的值。NEW在ON INSERT和ON UPDATE规则中有效，以引用要插入或更新的新行。OLD在ON UPDATE和ON DELETE规则中有效，以引用要更新或删除的现有行。

INSTEAD

INSTEAD NOTHING指示应执行的命令而不是原始命令。

ALSO

ALSO指示除了原始命令外，还应该执行命令。如果未指定ALSO或INSTEAD，则默认为ALSO。

command

组成规则操作的一个或多个命令。有效的命令是SELECT, INSERT, UPDATE或DELETE。特殊表名NEW和OLD可以用来引用表中的值。NEW在ON INSERT和ON UPDATE规则中有效，以引用要插入或更新的新行。OLD在ON UPDATE和ON DELETE规则中有效，以引用要更新或删除的现有行。

注解

您必须是表的所有者才能创建或更改该表的规则。

注意避免循环规则非常重要。递归规则在规则创建时未得到验证，但是将在执行时报告错误。

示例

创建一个规则，当用户尝试向分区的父表rank中插入行时，将行插入到子表b2001中：

```
CREATE RULE b2001 AS ON INSERT TO rank WHERE gender='M' and year='2001' DO INSTEAD INSERT INTO b2001 VALUES (NEW.id, NEW.rank, NEW.year, NEW.gender, NEW.count);
```

兼容性

CREATE RULE是Greenplum数据库语言的扩展，整个查询重写系统也是如此。

另见

[ALTER RULE](#) [DROP RULE](#) , [CREATE TABLE](#) , [CREATE VIEW](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一个新的模式。

概要

```
CREATE SCHEMA schema_name [ AUTHORIZATION username ]
[schema_element [ ... ]]

CREATE SCHEMA AUTHORIZATION rolename [schema_element [ ... ]]

CREATE SCHEMA IF NOT EXISTS schema_name [ AUTHORIZATION user_name ]

CREATE SCHEMA IF NOT EXISTS AUTHORIZATION user_name
```

描述

CREATE SCHEMA将新模式加入当前数据库。 模式名称必须不同于当前数据库中任何现有模式的名称。

模式本质上是一个名称空间：它包含命名对象（表，数据类型，函数和运算符），其名称可能与其他模式中存在的其他对象的名称重复。 通过使用模式名称作为前缀来限定其名称，或设置包含所需模式的搜索路径，可以访问命名对象。 使用CREATE命令指定了不合格的对象名称，将在当前模式（搜索路径前面的对象，可以使用户current_schema函数确定）中创建该对象。

(可选) CREATE SCHEMA可以包含子命令以在新模式中创建对象。 子命令与创建模式后发出的单独命令基本相同，不同之处在于，如果使用AUTHORIZATION子句，则所有创建的对象将归该角色所有。

参数

schema_name

要创建的模式的名称。如果省略，则将用户名用作名称。 该名称不能以pg_开头，因为此类名称是为系统catalog模式保留的。

user_name

拥有模式的角色的名称。如果省略，则默认为执行命令的角色。

只有超级用户可以创建除自己以外的角色所有的模式。

schema_element

定义要在模式内创建的对象的SQL语句。当前，在CREATE SCHEMA中，只有CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE SEQUENCE, CREATE TRIGGER和GRANT被接受为子句。创建模式后，可以在单独的命令中创建其他类型的对象。

Note: Greenplum数据库不支持触发器。

IF NOT EXISTS

如果已经存在相同名称的模式，则不执行任何操作（发出通知除外）。使用此选项时，不能包含*schema_element*子命令。

注解

要创建模式，调用用户必须对当前数据库具有CREATE特权，或者必须是超级用户。

示例

创建一个模式：

```
CREATE SCHEMA myschema;
```

为角色joe创建一个模式（该模式也将命名为joe）：

```
CREATE SCHEMA AUTHORIZATION joe;
```

创建一个名为test的模式，该模式将由用户joe拥有，除非已经有一个名为test的模式。（joe是否拥有预先存在的模式是没有影响的。）

```
CREATE SCHEMA IF NOT EXISTS test AUTHORIZATION joe;
```

兼容性

SQL标准允许在CREATE SCHEMA中使用DEFAULT CHARACTER

SET子句， 以及比Greenplum数据库目前接受的子命令类型更多的子命令类型。

SQL标准指定CREATE SCHEMA中的子命令可以按任何顺序出现。 当前的Greenplum数据库实现并未处理子命令中所有前向引用的情况；有时可能需要对子命令重新排序，以避免前向引用。

根据SQL标准，模式的所有者始终拥有其中的所有对象。

Greenplum数据库允许模式包含模式所有者以外的用户拥有的对象。仅当模式所有者将模式的CREATE特权授予其他人，或者超级用户选择在其中创建对象时，才会发生这种情况。

IF NOT EXISTS选项是Greenplum数据库扩展。

另见

[ALTER SCHEMA , DROP SCHEMA](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一个新的序列生成器。

概要

```
CREATE [TEMPORARY | TEMP] SEQUENCE name
      [INCREMENT [BY] value]
      [MINVALUE minvalue | NO MINVALUE]
      [MAXVALUE maxvalue | NO MAXVALUE]
      [START [ WITH ] start]
      [CACHE cache]
      [[NO] CYCLE]
      [OWNED BY { table.column | NONE }]
```

描述

CREATE SEQUENCE创建一个新的序列生成器。这涉及创建和初始化新的特殊单行表。生成器将由发出命令的用户所有。

如果指定了模式名称，则会在指定的模式中创建序列。否则，它将在当前模式中创建。临时序列存在于特殊模式中，因此在创建临时序列时可能不会给出模式名称。序列名称必须与同一模式中任何其他序列，表，索引，视图或外部表的名称不同。

创建序列后，可以使用nextval()函数对序列进行操作。例如，要将行插入到获取序列的下一个值的表中：

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

您还可以使用函数setval()对序列进行操作，但仅用于不对分布式数据进行操作的查询。例如，允许以下查询，因为它会重置master上序列生成器进程的序列计数器值：

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

但是以下查询在Greenplum数据库中将被拒绝，因为它对分布式数据进行操作：

```
INSERT INTO product VALUES (setval('myserial', 201),
 'gizmo');
```

在常规（非分布式）数据库中，对序列进行操作的函数会转到本地序列表以根据需要获取值。但是，请记住，在Greenplum数据库中，每个segment都是其自己不同的数据库进程。因此，segment需要一个真实单点来获取序列值，以便所有segment正确递增，并且序列以正确的顺序前进。序列服务器进程在master上运行，并且是Greenplum分布式数据库中序列的真实点。segment在运行时从master获取序列值。

由于这种分布式序列设计，因此在Greenplum数据库中对序列操作的函数存在一些限制：

- `lastval()`和`currval()`函数不被支持。
- `setval()`仅可用于在master上设置序列生成器的值，而不能在子查询中用于更新分布式表数据上的记录。
- 根据查询的不同，`nextval()`有时会从master获取一个值块以供segment使用。因此，如果在segment级别不需要所有块，有时可能会跳过序列中的值。请注意，常规PostgreSQL数据库也可以执行此操作，因此这并不是Greenplum数据库所独有的。

尽管您无法直接更新序列，但可以使用类似以下的查询：

```
SELECT * FROM sequence_name;
```

检查序列的参数和当前状态。特别是，序列的*last_value*字段显示了任何会话分配的最后一个值。

参数

TEMPORARY | TEMP

如果指定，则仅为此会话创建序列对象，并在会话退出时自动将其删除。具有相同名称的现有永久序列在临时序列存在时不可见（在此会话中），除非使用模式限定名称引用它们。

name

要创建的序列的名称（可以由模式指定）。

increment

指定将哪个值添加到当前序列值以创建新值。正值将形成一个升序，负值将形成一个降序。默认值为1。

minvalue

NO MINVALUE

确定序列可以生成的最小值。如果未提供此子句或指定了NO

`MINVALUE`, 则将使用默认值。升序和降序的默认值分别为1和-263-1。

maxvalue

NO MAXVALUE

确定序列的最大值。如果未提供此子句或指定了NO MAXVALUE, 则将使用默认值。升序和降序的默认值分别为263-1和-1。

start

允许序列从任何地方开始。默认的起始值为升序的最小值和降序的最大值。

cache

指定要预分配多少序号并将其存储在内存中, 以加快访问速度。最小(默认)值为1(无高速缓存)。

CYCLE

NO CYCLE

当达到最大值(递增)或最小值(递减)时, 允许序列回绕。如果达到限制, 则生成的下一个数字将是最小值(升序)或最大值(降序)。如果指定了NO CYCLE, 则在序列达到最大值之后, 对`nextval()`的任何调用都将返回错误。如果未指定, 则默认为NO CYCLE。

OWNED BY *table.column*

OWNED BY NONE

使序列与特定的表列相关联, 这样, 如果该列(或其整个表)被删除, 该序列也将被自动删除。指定的表必须具有相同的所有者, 并且与序列具有相同的模式。OWNED BY NONE(默认值)指定不存在这种关联。

注解

序列基于bigint算术, 因此范围不能超过八字节整数的范围(-9223372036854775808至9223372036854775807)。

尽管保证多个会话分配不同的序列值, 但是当考虑所有会话时, 这些值可能会不按顺序生成。例如, 会话A可能保留值1..10并返回`nextval=1`, 然后会话B可能保留值11..20并在会话A生成`nextval=2`之前返回`nextval=11`。因此, 您仅应假设`nextval()`值都是不同的, 而不是纯粹按顺序生成它们。同样, `last_value`将反映任何会话保留的最新值, 无论`nextval()`是否已返回该值。

示例

创建一个名为`myseq`的序列：

```
CREATE SEQUENCE myseq START 101;
```

在表中插入一行，以获取名为`idseq`的序列的下一个值：

```
INSERT INTO distributors VALUES (nextval('idseq'), 'acme');
```

在master上重置序列计数器值：

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

在Greenplum数据库中非法使用`setval()`（在分布式数据上设置序列值）：

```
INSERT INTO product VALUES (setval('myseq', 201), 'gizmo');
```

兼容性

`CREATE SEQUENCE`符合SQL标准，但以下情况除外：

- 不支持SQL标准中指定的`AS data_type`表达式。
- 使用`nextval()`函数代替SQL标准中指定的`NEXT VALUE FOR`表达式来获取下一个值。
- `OWNED BY`子句是Greenplum数据库扩展。

另见

[ALTER SEQUENCE](#) , [DROP SEQUENCE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一个新的外部服务器。

概要

```
CREATE SERVER server_name [ TYPE 'server_type' ] [ VERSION 'server_version' ]
    FOREIGN DATA WRAPPER fdw_name
    [ OPTIONS ( [ mpp_execute { 'master' | 'any' | 'all
    segments' } [ , ] ] option 'value' [ , ... ] ) ]
```

描述

CREATE SERVER 定义一个新的外部服务器。定义服务器的用户将成为其所有者。

外部服务器通常封装外部数据包装器用来访问外部数据源的连接信息。可以通过用户映射指定其他特定于用户的连接信息。

创建服务器需要对指定的外部数据包装器具有USAGE特权。

参数

server_name

要创建的外部服务器的名称。服务器名称在数据库中必须唯一。

server_type

可选服务器类型，可能对外部数据包装器有用。

server_version

可选服务器版本，可能对外部数据包装器有用。

fdw_name

管理服务器的外部数据包装器的名称。



```
OPTIONS ( option 'value' [, ... ] )
```

新的外部服务器的选项。这些选项通常定义服务器的连接详细信息，但是实际的名称和值取决于服务器的外部数据包装器。

```
mpp_execute { 'master' | 'any' | 'all segments' }
```

一个选项，用于标识外部数据包装器从其请求数据的主机：

- **master** (默认设置) - 从master主机请求数据。
- **any** — 向master主机或任一segment请求数据，具体取决于哪条路径的成本更低。
- **all segments** — 从所有segment中请求数据。要支持此选项值，外部数据包装器必须具有将segment与数据匹配的策略。

可以在多个命令中指定mpp_execute选项：CREATE FOREIGN TABLE, CREATE SERVER和CREATE FOREIGN DATA WRAPPER。外部表设置优先于外部服务器设置，然后是外部数据包装器设置。

注解

使用dblink模块（请参阅[dblink](#)）时，可以将外部服务器名称用作dblink_connect()函数的参数来提供连接参数。您必须在外部服务器上具有USAGE特权才能以这种方式使用它。

示例

创建一个名为myserver的外部服务器，该服务器使用名为pgsql的外部数据包装器并包含连接选项：

```
CREATE SERVER myserver FOREIGN DATA WRAPPER pgsql
    OPTIONS (host 'foo', dbname 'foodb', port '5432');
```

兼容性

CREATE SERVER符合ISO/IEC 9075-9 (SQL/MED)。

另见

[ALTER SERVER, DROP SERVER, CREATE FOREIGN DATA WRAPPER, CREATE USER MAPPING](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

 参考指南 SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一个新表。

Note: 引用完整性语法 (外键约束) 被接受但未强制执行。

概要

```

CREATE [ [GLOBAL | LOCAL] {TEMPORARY | TEMP} | UNLOGGED ]
TABLE [IF NOT EXISTS]
    table_name (
        [ { column_name data_type [ COLLATE collation ]
[column_constraint [ ... ] ]
[ ENCODING ( storage_directive [, ...] ) ]
            | table_constraint
            | LIKE source_table [ like_option ... ] }
            | [ column_reference_storage_directive [, ...]
            [, ... ]
        ]
        [ INHERITS ( parent_table [, ...] ) ]
        [ WITH ( storage_parameter [=value] [, ...] )
            | WITH OIDS | WITHOUT OIDS ]
        [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
        [ TABLESPACE tablespace_name ]
        [ DISTRIBUTED BY (column [opclass], [ ... ] )
            | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
        [ PARTITION BY partition_type (column)
            [ SUBPARTITION BY partition_type (column)
                [ SUBPARTITION TEMPLATE ( template_spec ) ]
                [ ... ]
            ( partition_spec )
            | [ SUBPARTITION BY partition_type (column) ]
            [ ... ]
            ( partition_spec
            [ ( subpartition_spec
                [(...)]
            ) ]
            ) ]
        ]
    )
CREATE [ [GLOBAL | LOCAL] {TEMPORARY | TEMP} | UNLOGGED ]
TABLE [IF NOT EXISTS]
    table_name
    OF type_name [ (
        { column_name WITH OPTIONS [ column_constraint [ ... ] ]
            | table_constraint }
            [, ... ]
    )
    [ WITH ( storage_parameter [=value] [, ...] ) | OIDS
        | WITHOUT OIDS ]
    [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
    [ TABLESPACE tablespace_name ]

```

其中*column_constraint*是：

```
[ CONSTRAINT constraint_name ]
{ NOT NULL
  | NULL
  | CHECK ( expression ) [ NO INHERIT ]
  | DEFAULT default_expr
  | UNIQUE index_parameters
  | PRIMARY KEY index_parameters
  | REFERENCES reftable [ ( refcolumn ) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
    [ ON DELETE key_action ] [ ON UPDATE key_action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED |
INITIALLY IMMEDIATE ]
```

*table_constraint*是：

```
[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) [ NO INHERIT ]
  | UNIQUE ( column_name [ , ... ] ) index_parameters
  | PRIMARY KEY ( column_name [ , ... ] ) index_parameters
  | FOREIGN KEY ( column_name [ , ... ] )
    REFERENCES reftable [ ( refcolumn [ , ... ] ) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
    [ ON DELETE key_action ] [ ON UPDATE key_action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED |
INITIALLY IMMEDIATE ]
```

*like_option*是：

```
{ INCLUDING | EXCLUDING }
{ DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS | ALL }
```

UNIQUE和PRIMARY KEY中的*index_parameters*约束为：

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

列的*storage_directive*是：

```
compressstype={ZLIB|ZSTD|QUICKLZ|RLE_TYPE|NONE}
[ compresslevel={0-9}]
[ blocksize={8192-2097152}]
```

表的*storage_parameter*是：

```
appendoptimized={TRUE|FALSE}
```

```

blocksize={8192-2097152}
orientation={COLUMN|ROW}
checksum={TRUE|FALSE}
compressstype={ZLIB|ZSTD|QUICKLZ|RLE_TYPE|NONE}
compresslevel={0-9}
fillfactor={10-100}
oids[=TRUE|FALSE]

```

*key_action*是：

```

ON DELETE
| ON UPDATE
| NO ACTION
| RESTRICT
| CASCADE
| SET NULL
| SET DEFAULT

```

*partition_type*是：

```
LIST | RANGE
```

*partition_specification*是：

```
partition_element [, ...]
```

*partition_element*是：

```

DEFAULT PARTITION
    name
| [PARTITION name] VALUES (list_value [,...])
| [PARTITION name]
    START ([datatype] 'start_value') [INCLUSIVE |
EXCLUSIVE]
    [ END ([datatype] 'end_value') [INCLUSIVE |
EXCLUSIVE] ]
    [ EVERY ([datatype] [number /INTERVAL]
interval_value) ]
| [PARTITION name]
    END ([datatype] 'end_value') [INCLUSIVE | EXCLUSIVE]
    [ EVERY ([datatype] [number /INTERVAL]
interval_value) ]
[ WITH ( partition_storage_parameter=value [, ...] ) ]
[ column_reference_storage_directive [, ...] ]
[ TABLESPACE tablespace ]

```

其中*subpartition_spec*或*template_spec*是：

```
subpartition_element [, ...]
```

*subpartition_element*是：

```

DEFAULT SUBPARTITION name
| [SUBPARTITION name] VALUES (list_value [,...] )
| [SUBPARTITION name]
    START ([datatype] 'start_value') [INCLUSIVE |
EXCLUSIVE]
    [ END ([datatype] 'end_value') [INCLUSIVE |
EXCLUSIVE] ]
    [ EVERY ([datatype] [number / INTERVAL]
'interval_value') ]
| [SUBPARTITION name]
    END ([datatype] 'end_value') [INCLUSIVE | EXCLUSIVE]
    [ EVERY ([datatype] [number / INTERVAL]
'interval_value') ]
[ WITH ( partition_storage_parameter=value [, ...] )
[ column_reference_storage_directive [, ...] ]
[ TABLESPACE tablespace ]

```

其中分区的*storage_parameter*是：

```

appendoptimized={TRUE | FALSE}
blocksize={8192-2097152}
orientation={COLUMN | ROW}
checksum={TRUE | FALSE}
compressstype={ZLIB | ZSTD | QUICKLZ | RLE_TYPE | NONE}
compresslevel={1-19}
fillfactor={10-100}
oids[=TRUE | FALSE]

```

描述

CREATE TABLE在当前数据库中创建一个最初为空的表。执行命令的用户拥有该表。

为了能够创建表，您必须分别对所有列类型或OF子句中的类型具有USAGE特权。

如果指定模式名称，Greenplum将在指定的模式中创建表。否则，Greenplum将在当前模式中创建表。临时表存在于特殊的模式中，因此在创建临时表时不能指定模式名称。表名称必须与同一模式中的任何其他表，外部表，序列，索引，视图或外部表的名称不同。

CREATE TABLE还会自动创建一个数据类型，该数据类型表示与表的一行相对应的复合类型。因此，表不能与同一模式中的任何现有数据类型具有相同的名称。

可选的约束子句指定新行或更新行必须满足的条件才能成功执行插入或更新操作。 约束是一个SQL对象，可以通过多种方式帮助定义表中的有效值集。 约束适用于表，而不适用于分区。 您不能将约束添加到分区或子分区。

引用完整性约束（外键）被接受但不强制执行。 该信息保留在系统catalog中，否则将被忽略。

有两种定义约束的方法：表约束和列约束。 列约束被定义为列定义的一部分。 表约束定义不与特定列绑定，并且可以包含多个列。 每个列约束也可以写为表约束。 当约束仅影响一列时，使用列约束只是一种符号上的方便。

创建表时，还有一个附加子句来声明Greenplum数据库分发策略。 如果未提供DISTRIBUTED BY, DISTRIBUTED RANDOMLY或DISTRIBUTED REPLICATED子句，则Greenplum数据库将通过使用PRIMARY KEY（如果表具有一个）或表的第一列作为分发键，向该表分配哈希分发策略。 几何或用户定义数据类型的列不符合Greenplum分布键列的要求。 如果表中没有符合条件的数据类型的列，则将根据轮询或随机分布来分配行。 为了确保数据在Greenplum数据库系统中的均匀分配，您希望选择一个对于每个记录都是唯一的分配键，或者如果不可能，则选择DISTRIBUTED RANDOMLY。

如果提供了DISTRIBUTED REPLICATED子句，则Greenplum数据库会将表的所有行分配给Greenplum数据库系统中的所有segment。 如果用户定义的函数必须在segment上执行，并且这些函数需要访问表的所有行，则可以使用此选项。 复制函数还可以用于防止表的broadcast motions，从而提高查询性能。 DISTRIBUTED REPLICATED子句不能与PARTITION BY子句或INHERITS子句一起使用。 复制的表也不能被另一个表继承。 隐藏的系统列（ctid, cmin, cmax, xmin, xmax和gp_segment_id）无法在复制表的用户查询中引用，因为它们没有单一的，无歧义的值。

通过PARTITION BY子句，您可以将表分为多个子表（或部分），这些子表一起构成父表并共享其模式。 尽管子表作为独立表存在，但是Greenplum数据库以重要方式限制了它们的使用。 在内部，分区被实现为继承的一种特殊形式。 每个子表分区都是根据不同的CHECK约束创建的，该约束根据一些定义条件限制表可以包含的数据。 查询优化器还使用CHECK约束来确定要扫描哪些表分区以满足给定的查询谓词。 这些分区约束由Greenplum数据库自动管理。

参数

GLOBAL | LOCAL

提供这些关键字是为了实现SQL标准兼容性，但在Greenplum数据库中无效，并且已弃用。

TEMPORARY | TEMP

如果指定，该表将被创建为临时表。临时表在会话结束时或在当前事务结束时自动删除（请参见ON COMMIT）。临时表存在时，具有相同名称的现有永久表在当前会话中不可见，除非使用模式限定名称引用它们。在临时表上创建的所有索引也会自动成为临时索引。

UNLOGGED

如果指定，该表将创建为未记录表。写入未记录表的数据不会写入预写（WAL）日志，这使它们比普通表快得多。但是，未记录表的内容不会复制到mirror实例。同样，未记录的表也不是崩溃安全的。segment实例崩溃或异常关闭后，该segment上未记录表的数据将被截断。在未记录表上创建的所有索引也会自动成为未记录索引。

table_name

要创建的新表的名称（可以由模式指定）。

OF *type_name*

创建一个类型化的表，该表从指定的组合类型（名称可以由模式指定）获取其结构。类型化的表与其类型相关联。例如，如果删除了类型（使用DROP TYPE ... CASCADE），则将删除该表。

当一个类型化的表被创建时，列的数据类型由底层的组合类型决定而没有在CREATE TABLE命令中直接指定。但是CREATE TABLE命令可以对表增加默认值和约束，并且可以指定存储参数。

column_name

要在新表中创建的列的名称。

data_type

列的数据类型。这可能包括数组说明符。

对于包含文本数据的表列，请指定数据类型VARCHAR或TEXT。

不建议指定数据类型CHAR。在Greenplum数据库中，数据类型VARCHAR或TEXT处理作为有效字符添加到数据（在最后一个非空格字符之后添加的空格字符）的填充，而数据类型CHAR不处理。请参阅[注解](#)。

COLLATE *collation*

COLLATE子句为该列分配排序规则（该排序规则必须是可排序的数据类型）。如果未指定，则使用列数据类型的默认排序规则。

Note: 仅当查询中的所有列使用相同的排序规则时，GPORCA才支持排序规则。如果查询中的列使用不同的排序规则，

则Greenplum使用Postgres查询优化器。

DEFAULT *default_expr*

DEFAULT子句为出现在其列定义中的列分配默认数据值。该值是任何不带变量的表达式（不允许对当前表中的其他列进行子查询和交叉引用）。默认表达式的数据类型必须与列的数据类型匹配。默认表达式将在未为列指定值的任何插入操作中使用。如果列没有默认值，则默认值为null。

ENCODING (*storage_directive* [, ...])

对于列，可选的ENCODING子句指定列数据的压缩类型和块大小。有

关compressstype, compresslevel和blocksize值，请参见[storage_options](#)。

该子句仅对追加优化的，面向列的表有效。

列压缩设置从表级别继承到分区级别再到子分区级别。最低级别的设置具有优先权。

INHERITS (*parent_table* [, ...])

可选的INHERITS子句指定一个表列表，新表将从中自动继承所有列。使用INHERITS将在新的子表及其父表之间创建持久关系。对父级的模式修改通常也会传播到子级，默认情况下，子级表的数据包含在父级扫描中。

在Greenplum数据库中，创建分区表时不使用INHERITS子句。

尽管在分区层次结构中使用了继承的概念，但是使用[PARTITION BY](#)子句创建了分区表的继承结构。

如果一个以上的父表中存在相同的列名，则将报告错误，除非每个父表中的列的数据类型都匹配。如果没有冲突，则将重复的列合并以在新表中形成一个列。如果新表的列名列表包含一个也被继承的列名，则数据类型必须同样与继承的列匹配，并且列定义将合并为一个。如果新表显式指定了该列的默认值，则该默认值将覆盖该列的继承声明中的所有默认值。否则，为该列指定默认值的所有父项都必须指定相同的默认值，否则将报告错误。

CHECK约束基本上以与列相同的方式合并：如果多个父表或新表定义包含名称相同的约束，则这些约束必须全部具有相同的校验表达式，否则将报告错误。具有相同名称和表达式的约束将合并为一个副本。不会考虑在父级中标记为NO INHERIT的约束。请注意，新表中未命名的CHECK约束将永远不会被合并，因为将始终为其选择唯一的名称。

列STORAGE设置也会从父表中复制。

LIKE *source_table* *like_option* . . .]

LIKE子句指定一个表，新表将从该表中自动复制所有列名，其数据类型，非空约束和分发策略。不会复制诸如追加优化或分区结构之类的存储属性。与INHERITS不同，新表和原始表在创建完成后完全解耦。

仅当指定INCLUDING DEFAULTS时，才会复制复制的列定义的

默认表达式。默认行为是排除默认表达式，导致新表中复制的列具有空默认值。

非空约束始终会复制到新表中。仅当指定 INCLUDING CONSTRAINTS 时，才会复制 CHECK 约束。列约束和表约束之间没有区别。

仅在指定 INCLUDING INDEXES 子句的情况下，才会在新表上创建原始表的索引， PRIMARY KEY 和 UNIQUE 约束。不论原始名称如何命名，都会根据默认规则选择新索引和约束的名称。（此行为避免了新索引可能出现的重复名称错误。）

除非指定了 INCLUDING INDEXES 子句，否则不会在新表上创建原始表上的任何索引。

仅当指定了 INCLUDING STORAGE 时，才会复制复制的列定义的 STORAGE 设置。默认行为是排除 STORAGE 设置，导致新表中复制的列具有特定于类型的默认设置。

仅当指定 INCLUDING COMMENTS 时，才会复制复制的列，约束和索引的注释。默认行为是排除注释，导致新表中复制的列和约束没有注释。

`INCLUDING ALL` 是 `INCLUDING DEFAULTS INCLUDING CONSTRAINTS INCLUDING INDEXES INCLUDING STORAGE INCLUDING COMMENTS` 的缩写形式。

请注意，与 `INHERITS` 不同，`LIKE` 复制的列和约束不会与名称相似的列和约束合并。如果显式指定了相同的名称，或者在另一个 `LIKE` 子句中指定了相同的名称，则将指示错误。

`LIKE` 子句还可用于从视图、外部表或复合类型中复制列。不适用的选项（例如，从视图 `INCLUDING INDEXES`）将被忽略。

CONSTRAINT *constraint_name*

列或表约束的可选名称。如果违反了约束，那么约束名称将出现在错误消息中，因此可以使用约束名称（例如列必须为正）来将有用的约束信息传达给客户端应用程序。（需要双引号指定包含空格的约束名称。）如果未指定约束名称，则系统将生成一个名称。

Note: 指定的 *constraint_name* 用于约束，但系统生成的唯一名称用于索引名。在某些以前的版本中，提供的名称同时用于约束名称和索引名称。

NULL | NOT NULL

指定是否允许该列包含空值。默认值为 `NULL`。

CHECK (*expression*) [NO INHERIT]

`CHECK` 子句指定一个生成布尔结果的表达式，新的或更新的行必须满足才能使插入或更新操作成功。计算为 `TRUE` 或 `UNKNOWN` 的表达式会成功。如果插入或更新操作的任何行都产生 `FALSE` 结果，则会引发错误异常，并且插入或更新不会更改数据库。指定为列约束的检查约束应仅引用该列的值，而出现在表约束中的表达式可以引用多个列。

标有 `NO INHERIT` 的约束不会传播到子表。

当前，CHECK表达式不能包含子查询，也不能引用当前行的列以外的变量。

UNIQUE (*column_constraint*)

UNIQUE (*column_name* [, ...]) (*table_constraint*)

UNIQUE约束指定表的一组一个或多个列只能包含唯一值。唯一表约束的行为与列约束的行为相同，但具有跨多个列的附加功能。出于唯一约束的目的，空值不视为相等。唯一的列必须包含Greenplum分布键的所有列。此外，如果表已分区，则必须包含分区键中的所有列。请注意，分区表中的约束与简单的UNIQUE INDEX不同。

有关唯一约束管理和限制的信息，请参见[注解](#)。

PRIMARY KEY (*column_constraint*)

PRIMARY KEY (*column_name* [, ...]) (*table_constraint*)

PRIMARY KEY约束指定表的一列或多列只能包含唯一（非重复），非null值。只能为一个表指定一个主键，无论是作为列约束还是表约束。

要使一个表具有主键，它必须是哈希分布的（不是随机分布的），并且主键（唯一的列）必须包含Greenplum分布键的所有列。此外，如果表已分区，则必须包含分区键中的所有列。请注意，分区表中的约束与简单的UNIQUE INDEX不同。

PRIMARY KEY强制执行与UNIQUE和NOT NULL相同的组合数据约束，但是将一组列标识为主键也可以提供有关模式设计的元数据，因为主键标识其他表可以依赖这一个列集合作为行的唯一标识符。

有关主键管理和限制的信息，请参阅[注解](#)。

REFERENCES *reftable* [(*refcolumn*)]

[MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]

[ON DELETE | ON UPDATE] [*key_action*]

FOREIGN KEY (*column_name* [, ...])

REFERENCES和FOREIGN KEY子句指定引用完整性约束（外键约束）。Greenplum接受PostgreSQL语法中指定的参照完整性约束，但不强制执行。有关引用完整性约束的信息，请参见PostgreSQL文档。

DEFERRABLE

NOT DEFERRABLE

[NOT] DEFERRABLE子句控制约束是否可以被推迟。一个不可推迟的约束将在每个命令后立即进行检查。可以推迟检查约束，直到事务结束（使用SET CONSTRAINTS命令）。默认值是NOT DEFERRABLE。当前，只有UNIQUE和PRIMARY KEY约束是可推迟的。NOT NULL和CHECK约束不可延迟。

REFERENCES（外键）约束接受此子句，但不强制执行。

INITIALLY IMMEDIATE

INITIALLY DEFERRED

如果约束是可延迟的，则此子句指定检查约束的默认时间。如果该约束是INITIALLY IMMEDIATE，则在每个语句之后对其进行检查。这是默认值。如果约束是INITIALLY DEFERRED，则仅在事务结束时进行检查。可以使用SET CONSTRAINTS命令更改约束检查时间。

WITH (*storage_parameter*=*value*)

WITH子句可以为表以及与UNIQUE或PRIMARY约束关联的索引指定存储参数。请注意，您还可以通过在分区规范中声明WITH子句来在特定分区或子分区上设置存储参数。最低级别的设置具有优先权。

某些表存储选项的默认值可以使用服务器配置参数gp_default_storage_options指定。有关设置默认存储选项的信息，请参阅[注解](#)。

可以使用以下存储选项：

appendoptimized — 设置为TRUE可将表创建为追加优化的表。如果为FALSE或未声明，则将表创建为常规堆存储表。

blocksize — 设置为表中每个块的大小（以字节为单位）。blocksize必须介于8192和2097152字节之间，并且是8192的倍数。默认值为32768。

orientation — 设置为column以用于列式存储，或设置为row（默认）以用于行式存储。仅当appendoptimized=TRUE时，此选项才有效。堆存储表只能是面向行的。

checksum — 此选项仅对追加优化的表

(appendoptimized=TRUE) 有效。值TRUE是默认值，并为追加优化表启用CRC校验和验证。校验和是在块创建期间计算的，并存储在磁盘上。在块读取期间执行校验和验证。如果在读取期间计算出的校验和与存储的校验和不匹配，则事务中止。如果将值设置为FALSE以禁用校验和验证，将不会执行检查表数据是否存在磁盘损坏的操作。

compressstype — 设置为ZLIB（默认值），ZSTD，RLE_TYPE或QUICLZ¹以指定使用的压缩类型。值NONE禁用压缩。Zstd提供速度或良好的压缩率，可通过compresslevel选项进行调整。提供QuickLZ和zlib是为了向后兼容。在通常的工作负载上，Zstd的性能优于这些压缩类型。仅当appendoptimized=TRUE时，compressstype选项才有效。

Note: 注意：¹QuickLZ压缩仅在商业版本的Pivotal Greenplum数据库中可用。

仅当指定了orientation=column时才支持值RLE_TYPE，它启用游程编码（RLE）压缩算法。当相同的数据值出现在许多连续的行中时，RLE的压缩数据比Zstd， zlib或QuickLZ压缩算法更好。

对于BIGINT, INTEGER, DATE, TIME或TIMESTAMP类型的列，如果将compressstype选项设置为RLE_TYPE压缩，则还将应用增量压缩。增量压缩算法基于连续行中列值之间的增量，旨在改善按排序顺序加载数据或将压缩应用于按排序顺序的列数据时的压缩。

有关使用表压缩的信息，请参阅*Greenplum*数据库管理员指南中的“选择表存储模型”。

compresslevel — 对于追加优化表的Zstd压缩，请将其设置为1（最快压缩）到19（最高压缩率）之间的整数值。对于zlib压缩，有效范围是1到9。QuickLZ压缩级别只能设置为1。如果未声明，则默认值为1。对于RLE_TYPE，压缩级别可以是1（最快压缩）到4（最高压缩率）之间的整数值。
仅当appendoptimized=TRUE时，compresslevel选项才有效。

fillfactor — 有关此索引存储参数的更多信息，请参见[CREATE INDEX](#)。

oids — 设置为oids=False（默认值），以便不为行分配对象标识符。Greenplum强烈建议您在创建表时不要启用OIDS。在大型表上（例如典型的Greenplum数据库系统中的表），对表行使用OID可能会导致32位OID计数器的折回。一旦计数器回绕，就不能再认为OID是唯一的，这不仅使它们对用户应用程序无用，而且还会在Greenplum数据库系统catalog表中引起问题。此外，从表中排除OID会使表每行存储在磁盘上所需的空间减少了每行4个字节，从而略微提高了性能。分区表或追加优化的面向列的表上不允许使用OIDS。

WITH OIDS

WITHOUT OIDS

它们是分别等同于WITH (OIDS)和WITH (OIDS=False)的过时语法。如果希望同时提供OIDS设置和存储参数，则必须使用WITH (...)语法；往上看。Greenplum强烈建议您在创建表时不要启用OIDS。有关使用oids的更多信息，请参见oids参数说明。

ON COMMIT

可以使用ON COMMIT控制事务块末尾的临时表的行为。这三个选项是：

PRESERVE ROWS - 临时表的事务结束时不会采取任何特殊操作。这是默认行为。

DELETE ROWS - 临时表中的所有行将在每个事务块的末尾删除。本质上，每次提交都会自动执行一次TRUNCATE。

DROP - 临时表将在当前事务块的末尾删除。

TABLESPACE *tablespace*

要在其中创建新表的表空间的名称。如果未指定，则使用数据库

的默认表空间。

USING INDEX TABLESPACE *tablespace*

该子句允许选择将在其中创建与UNIQUE或PRIMARY KEY约束关联的索引的表空间。如果未指定，则使用数据库的默认表空间。

DISTRIBUTED BY (*column [opclass], [...]*)

DISTRIBUTED RANDOMLY

DISTRIBUTED REPLICATED

用于声明表的Greenplum数据库分布策略。DISTRIBUTED BY使用具有一个或多个声明为分布键的列的哈希分布。为了获得最均匀的数据分配，分布键应为表的主键或唯一列（或一组列）。如果无法做到这一点，则可以选择DISTRIBUTED RANDOMLY，它将数据轮询发送到segment实例。此外，可以指定运算符类*opclass*，以使用非默认哈希函数。

如果在创建表时未指定DISTRIBUTED BY子句，则Greenplum数据库服务器配置参数

`gp_create_table_random_default_distribution`将控制表默认分布策略。如果未指定分布策略，Greenplum数据库将遵循以下规则来创建表。

如果参数的值是`off`（默认值），则Greenplum数据库根据以下命令选择表分布键：

- 如果指定了LIKE或INHERITS子句，则Greenplum从源表或父表复制分布键。
- 如果指定了PRIMARY KEY或UNIQUE约束，则Greenplum选择所有键列中最大的子集作为分布键。
- 如果既未指定约束，也未指定LIKE或INHERITS子句，则Greenplum选择第一个合适的列作为分布键。（具有几何或用户定义数据类型的列不符合作为Greenplum分布键列的条件。）

如果参数的值设置为`on`，则Greenplum数据库遵循以下规则：

- 如果未指定PRIMARY KEY或UNIQUE列，则表的分布是随机的（DISTRIBUTED RANDOMLY）。即使表创建命令包含LIKE或INHERITS子句，表分布也是随机的。
- 如果指定了PRIMARY KEY或UNIQUE列，则还必须指定DISTRIBUTED BY子句。如果在表创建命令中未指定DISTRIBUTED BY子句，则该命令将失败。

有关设置默认表分布策略的更多信息，请参

见[gp_create_table_random_default_distribution](#)。

DISTRIBUTED REPLICATED子句将整个表复制到所

有Greenplum数据库segment实例。当函数需要访问表中的所有行或需要通过阻止broadcast motion来提高查询性能时，必须在segment上执行用户定义的函数时可以使用它。

PARTITION BY

声明用于对表进行分区的一列或多列。

创建分区表时，Greenplum数据库使用指定的表名创建根分区表（根分区）。Greenplum数据库还会根据您指定的分区选项创建表，子表的层次结构，这些表是子分区。Greenplum数据库`pg_partition`*系统视图包含有关子分区表的信息。

对于每个分区级别（表的每个层次结构级别），一个分区表最多可以有32,767个分区。

Note: Greenplum数据库将分区表数据存储在叶子表中，叶子表是子表层次结构中的最低级表，供分区表使用。

partition_type

声明分区类型：LIST（值列表）或RANGE（数字或日期范围）。

partition_specification

声明要创建的各个分区。可以单独定义每个分区，或者对于范围分区，可以使用EVERY子句（带有START和可选END子句）来定义用于创建单个分区的增量模式。

DEFAULT PARTITION name — 声明默认分区。当数据与现有分区不匹配时，会将其插入默认分区。没有默认分区的分区设计将拒绝与现有分区不匹配的传入行。

PARTITION name — 声明要用于分区的名称。使用以下命名约定创建分区：`parentname_level#_prt_givenname`。

VALUES — 对于列表分区，定义分区将包含的值。

START — 对于范围分区，定义分区的起始范围值。默认情况下，起始值为INCLUSIVE。例如，如果您声明开始日期为'2016-01-01'，则分区将包含所有大于或等于'2016-01-01'的日期。通常，START表达式的数据类型与分区键列的类型相同。如果不是这种情况，则必须显式转换为预期的数据类型。

END — 对于范围分区，定义分区的结束范围值。默认情况下，结束值为EXCLUSIVE。例如，如果您声明结束日期为'2016-02-01'，则分区将包含所有小于但不等于'2016-02-01'的日期。通常，END表达式的数据类型与分区键列的类型相同。如果不是这种情况，则必须显式转换为预期的数据类型。

EVERY — 对于范围分区，定义如何将值从START递增到END以创建单个分区。通常，EVERY表达式的数据类型与分区键列的类型相同。如果不是这种情况，则必须显式转换为预期的数据类型。

WITH — 设置分区的表存储选项。例如，您可能希望将较旧的分区作为追加优化表，而将较新的分区作为常规堆表。

TABLESPACE —

要在其中创建分区的表空间的名称。

SUBPARTITION BY

声明用于对表的第一级分区进行子分区的一个或多个列。子分区的规范格式类似于上述分区的规范格式。

SUBPARTITION TEMPLATE

您可以选择声明一个用于创建子分区的子分区模板（低级别子表），而不是为每个分区分别声明每个子分区定义。然后，此子分区规范将应用于所有父分区。

注解

- 在Greenplum数据库（基于Postgres的系统）中，数据类型VARCHAR或TEXT处理填充作为有效字符添加到文本的数据（最后一个非空格字符之后添加空格字符）；数据类型CHAR则没有。在Greenplum数据库中，CHAR(*n*)类型的值用尾随空格填充到指定的宽度*n*。值将存储并显示为空格。但是，填充空格在语义上无关紧要。分配值时，将忽略尾随空格。比较数据类型CHAR的两个值时，尾随空格在语义上也无关紧要，而将字符值转换为其他字符串类型之一时，尾随空格也将被删除。
- 不建议在新应用程序中使用OID：在可能的情况下，最好使用SERIAL或其他序列生成器作为表的主键。但是，如果您的应用程序确实使用OID来标识表的特定行，则建议在该表的OID列上创建唯一约束，以确保即使在计数器回绕后，表中的OID的确可以唯一地标识行。避免假设OID在表之间是唯一的；如果需要数据库范围的唯一标识符，则可以使用表OID和行OID的组合。
- Greenplum数据库对于主键和作为Greenplum表中的分布键的列的唯一约束具有一些特殊条件。为了在Greenplum数据库中实施唯一约束，表必须是哈希分布的（不是DISTRIBUTED RANDOMLY），并且约束列必须与表的分布键列相同（或作为其的超集）。另外，分布键必须是约束列的左子集，并且列的顺序正确。例如，如果主键是(a, b, c)，则分布键只能是以下之一：(a)，(a, b)或(a, b, c)。

复制表(DISTRIBUTED REPLICATED)可以同时具有PRIMARY KEY和UNIQUE列约束。

主键约束只是唯一约束和非空约束的组合。

Greenplum数据库自动为每个UNIQUE或PRIMARY KEY约束创建一个UNIQUE索引，以强制执行唯一性。因此，没有必要为主键列显式创建索引。在追加优化表上不允许使用UNIQUE和PRIMARY KEY约束，因为在追加优化表上不允许通过约束创建的UNIQUE索引。

Greenplum数据库中不支持外键约束。

对于继承的表，当前实现中不会继承唯一约束，主键约束，索引和表特权。

- 对于追加优化的表，可重复读或可序列化事务中不允许UPDATE和DELETE，这将导致事务中止。
CLUSTER, DECLARE...FOR UPDATE和触发器不支持追加优化的表。
- 要将数据插入分区表中，请指定根分区表，即使用CREATE TABLE命令创建的表。您还可以在INSERT命令中指定分区表的叶子表。如果数据对于指定的叶子表无效，则返回错误。不支持在INSERT命令中指定不是叶子表的子表。不支持在分区表的任何子表上执行其他DML命令，例如UPDATE和DELETE。这些命令必须在根分区表（使用CREATE TABLE命令创建的表）上执行。
- 可以使用服务器配置参数gp_default_storage_option指定这些表存储选项的默认值。
 - appendoptimized
 - blocksize
 - checksum
 - compressstype
 - compresslevel
 - orientation

可以为系统，数据库或用户设置默认值。有关设置存储选项的信息，请参阅服务器配置参数[gp_default_storage_options](#)。

Important: 当前的Greenplum数据库Postgres优化器允许使用具有多列（复合）分区键的列表分区。GPORC不支持复合键，因此不建议使用复合分区键。

示例

在名为baby的模式中创建一个名为rank的表，并使用rank, gender和year列分发数据：

```
CREATE TABLE baby.rank (id int, rank int, year smallint,
gender char(1), count int ) DISTRIBUTED BY (rank, gender,
year);
```

创建表files和表分配器（默认情况下，主键将用作Greenplum分布键）：

```
CREATE TABLE films (
```

```

code      char(5) CONSTRAINT firstkey PRIMARY KEY,
title    varchar(40) NOT NULL,
did      integer NOT NULL,
date_prod date,
kind     varchar(10),
len      interval hour to minute
);

CREATE TABLE distributors (
did      integer PRIMARY KEY DEFAULT nextval('serial'),
name    varchar(40) NOT NULL CHECK (name <> '')
);

```

创建一个gzip压缩的，追加优化的表：

```

CREATE TABLE sales (txn_id int, qty int, date date)
WITH (appendoptimized=true, compresslevel=5)
DISTRIBUTED BY (txn_id);

```

使用子分区模板和每个级别的默认分区创建一个三级分区表：

```

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 (2008) END (2016) EVERY (1),
DEFAULT PARTITION outlying_years);

```

兼容性

CREATE TABLE命令符合SQL标准，但以下情况除外：

- **Temporary Tables** — 在SQL标准中，临时表仅定义一次，并在每个需要它们的会话中自动存在（从空内容开始）。相反，Greenplum数据库要求每个会话为要使用的每个临时表发出自己的CREATE TEMPORARY TABLE命令。这允许不同的会话出于

不同的目的而使用相同的临时表名称，而标准的方法将给定临时表名称的所有实例约束为具有相同的表结构。

全局和本地临时表之间的标准区别不在Greenplum数据库中。

Greenplum数据库将在临时表声明中接受GLOBAL和LOCAL关键字，但它们无效且已弃用。

如果省略ON COMMIT子句，则SQL标准将默认行为指定为ON COMMIT DELETE ROWS。但是，Greenplum数据库中的默认行为是ON COMMIT PRESERVE ROWS。SQL标准中不存在ON COMMIT DROP选项。

- **Column Check Constraints** — SQL标准说，CHECK列约束只能引用它们所适用的列。只有CHECK表约束可以引用多个列。Greenplum数据库不强制执行此限制；它对待列和表检查约束都一样。
- **NULL Constraint** — NULL约束是对SQL标准的Greenplum数据库扩展，为了与某些其他数据库系统兼容（以及对称的NOT NULL约束）。由于它是任何列的默认值，因此不需要它的存在。
- **Inheritance** — 通过INHERITS子句的多重继承是Greenplum数据库语言的扩展。SQL：1999及更高版本使用不同的语法和语义定义了单个继承。Greenplum数据库尚不支持SQL：1999样式的继承。
- **Partitioning** — 通过PARTITION BY子句进行的表分区是Greenplum数据库语言的扩展。
- **Zero-column tables** — Greenplum数据库允许创建不包含任何列的表（例如CREATE TABLE foo();）。这是SQL标准的扩展，不允许使用零列表。零列表本身并没有什么用，但是不允许使用零列表在ALTER TABLE DROP COLUMN时会产生奇怪的特殊情况，因此Greenplum决定忽略此规范限制。
- **LIKE** — 尽管SQL标准中存在LIKE子句，但Greenplum数据库接受的许多选项都不在该标准中，并且Greenplum数据库并未实现该标准的某些选项。
- **WITH clause** — WITH子句是Greenplum数据库扩展。存储参数和OID都不在标准中。
- **Tablespaces** — 表空间的Greenplum数据库概念不是SQL标准的一部分。子句TABLESPACE和USING INDEX TABLESPACE是扩展。
- **Data Distribution** — 并行或分布式数据库的Greenplum数据库概念不是SQL标准的一部分。DISTRIBUTED子句是扩展。

另见

[ALTER TABLE](#) , [DROP TABLE](#) , [CREATE EXTERNAL TABLE](#) ,
[CREATE TABLE AS](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

根据查询结果定义一个新表。

概要

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ]
] TABLE table_name
[ (column_name [, ...] ) ]
[ WITH ( storage_parameter [= value] [, ...] ) |
WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP }
]
[ TABLESPACE tablespace_name ]
AS query
[ WITH [ NO ] DATA ]
[ DISTRIBUTED BY (column [, ...] ) | DISTRIBUTED
RANDOMLY | DISTRIBUTED REPLICATED ]
```

其中*storage_parameter*是：

```
appendoptimized={TRUE|FALSE}
blocksize={8192-2097152}
orientation={COLUMN|ROW}
compressstype={ZLIB|ZSTD|QUICKLZ}
compresslevel={1-19 | 1}
fillfactor={10-100}
oids[=TRUE|FALSE]
```

描述

CREATE TABLE AS创建一个表，并用[SELECT](#)命令计算的数据填充该表。表列具有与SELECT的输出列关联的名称和数据类型，但是您可以通过提供新列名称的显式列表来覆盖列名称。

CREATE TABLE AS创建一个新表并仅对查询进行一次取值以填充一次新表。新表将不会跟踪对查询源表的后续更改。

参数



GLOBAL | LOCAL

忽略兼容性。这些关键字已弃用；有关详细信息，请参考[CREATE TABLE](#)。

TEMPORARY | TEMP

如果指定，则将新表创建为临时表。临时表在会话结束时或在当前事务结束时自动删除（请参见ON COMMIT）。临时表存在时，具有相同名称的现有永久表在当前会话中不可见，除非使用模式限定名称引用它们。在临时表上创建的所有索引也会自动成为临时索引。

UNLOGGED

如果指定，该表将创建为未记录表。写入未记录表的数据不会写入预写（WAL）日志，这使它们比普通表快得多。但是，未记录表的内容不会复制到mirror实例。同样，未记录的表也不是崩溃安全的。segment实例崩溃或异常关闭后，该segment上未记录表的数据将被截断。在未记录表上创建的所有索引也会自动成为未记录索引。

table_name

要创建的新表的名称（可以由模式指定）。

column_name

新表中的列名。如果未提供列名，则它们取自查询的输出列名。

WITH (*storage_parameter*=*value*)

WITH子句可用于设置表或其索引的存储选项。请注意，您还可以通过在分区规范中声明WITH子句，在特定分区或子分区上设置不同的存储参数。可以使用以下存储选项：

appendoptimized — 设置为TRUE可将表创建为追加优化的表。如果为FALSE或未声明，则将表创建为常规堆存储表。

blocksize — 设置为表中每个块的大小（以字节为单位）。blocksize必须介于8192和2097152字节之间，并且是8192的倍数。默认值为32768。

orientation — 设置为column以用于列式存储，或设置为row（默认）以用于行式存储。仅当appendoptimized=TRUE时，此选项才有效。堆存储表只能是面向行的。

compressstype — 设置为ZLIB（默认值），ZSTD或QUICKLZ¹以指定使用的压缩类型。值NONE禁用压缩。Zstd提供速度或良好的压缩率，可通过compresslevel选项进行调整。提供QuickLZ和zlib是为了向后兼容。在通常的工作负载上，Zstd的性能优于这些压缩类型。仅当appendoptimized=TRUE时，compressstype选项才有效。

Note: ¹QuickLZ压缩仅在Pivotal Greenplum数据库的商业版本中可用。

compresslevel — 对于附加优化表的Zstd压缩，请将其设置为1（最快压缩）到19（最高压缩率）之间的整数值。对

于zlib压缩，有效范围是1到9。QuickLZ压缩级别只能设置为1。如果未声明，则默认值为1。`compresslevel`选项仅在`appendoptimized=TRUE`时有效。

fillfactor — 有关此索引存储参数的更多信息，请参见[CREATE INDEX](#)。

oids — 设置为`oids=FALSE`（默认值），以便不为行分配对象标识符。Greenplum强烈建议您在创建表时不要启用OIDS。在大型表上（例如典型的Greenplum数据库系统中的表），对表行使用OID可能会导致32位OID计数器的折回。一旦计数器回绕，就不能再认为OID是唯一的，这不仅使它们对用户应用程序无用，而且还会在Greenplum数据库系统catalog表中引起问题。此外，从表中排除OID会使表每行存储在磁盘上所需的空间减少了每行4个字节，从而略微提高了性能。面向列的表不允许使用OIDS。

ON COMMIT

可以使用`ON COMMIT`控制事务块末尾的临时表的行为。这三个选项是：

PRESERVE ROWS — 临时表的事务结束时不会采取任何特殊操作。这是默认行为。

DELETE ROWS — 临时表中的所有行将在每个事务块的末尾删除。本质上，每次提交都会自动执行一次`TRUNCATE`。

DROP — 临时表将在当前事务块的末尾删除。

TABLESPACE *tablespace_name*

*tablespace_name*参数是要在其中创建新表的表空间的名称。

如果未指定，则使用数据库的默认表空间。

AS *query*

`SELECT`, `TABLE`或`VALUES`命令，或运行准备的`SELECT`或`VALUES`查询的`EXECUTE`命令。

`DISTRIBUTED BY ({column [opclass]}, [...])`

`DISTRIBUTED RANDOMLY`

`DISTRIBUTED REPLICATED`

用于声明表的Greenplum数据库分发策略。`DISTRIBUTED BY`使用具有一个或多个声明为分发键的列的哈希分发。为了获得最均匀的数据分配，分发键应为表的主键或唯一列（或一组列）。如果无法做到这一点，则可以选择`DISTRIBUTED RANDOMLY`，它将数据轮询发送到segment实例。

`DISTRIBUTED REPLICATED`将表中的所有行复制到所有Greenplum数据库segment。它不能与分区表或从其他表继承的表一起使用。

如果在创建表时未指定`DISTRIBUTED BY`子句，则Greenplum数据库服务器配置参

数`gp_create_table_random_default_distribution`将控制默认表分发策略。如果未指定分发策略，Greenplum数据库将遵循以下规则来创建表。

如果Postgres查询优化器创建了表，并且该参数的值是off，则根据命令确定表分配策略。

- 如果Postgres查询优化器创建了表，并且该参数的值为on，则表分配策略是随机的。
- 如果GPORCA创建表，则表分配策略是随机的。参数值无效。

有关设置默认表分发策略的更多信息，请参见[gp_create_table_random_default_distribution](#)。

有关Postgres查询优化器和GPORCA的信息，请参阅*Greenplum*数据库管理员指南中的[查询数据](#)。

注解

该命令在功能上类似于[SELECT INTO](#)，但它是首选的，因为它不太可能与SELECT INTO语法的其他用法混淆。此外，CREATE TABLE AS提供了SELECT INTO提供的功能的超集。

CREATE TABLE AS可用于从外部表数据源快速加载数据。请参阅[CREATE EXTERNAL TABLE](#)。

示例

创建一个新表films_recent，该表仅包含表films中的最新条目：

```
CREATE TABLE films_recent AS SELECT * FROM films WHERE
date_prod >= '2007-01-01';
```

使用预编译语句创建一个新的临时表films_recent，该表仅包含表films中的最新条目。新表具有OID，并将在提交时删除：

```
PREPARE recentfilms(date) AS SELECT * FROM films WHERE
date_prod > $1;
CREATE TEMP TABLE films_recent WITH (OIDS) ON COMMIT DROP
AS
EXECUTE recentfilms('2007-01-01');
```

兼容性

CREATE TABLE AS符合SQL标准，但以下情况除外：

- 该标准要求在子查询子句两边加上括号；在Greenplum数据库中，这些括号是可选的。
- 该标准定义了WITH [NO] DATA子句；Greenplum数据库当前未实现此功能。Greenplum数据库提供的行为等同于标准的WITH DATA情况。可以通过将LIMIT 0附加到查询来模拟WITH NO DATA。
- Greenplum数据库对临时表的处理方式与标准处理方式有所不同。有关详细信息，请参见CREATE TABLE。
- WITH子句是Greenplum数据库扩展。storage参数和OIDs都不在标准中。
- 表空间的Greenplum数据库概念不是该标准的一部分。TABLESPACE子句是扩展。

另见

[CREATE EXTERNAL TABLE](#), [CREATE EXTERNAL TABLE](#),
[EXECUTE](#), [SELECT](#), [SELECT INTO](#), [VALUES](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一个新的表空间。

概要

```
CREATE TABLESPACE tablespace_name [ OWNER username ]
LOCATION '/path/to/dir'
[WITH (contentID_1='/path/to/dir1'[,  
contentID_2='/path/to/dir2' ... ])]
```

描述

CREATE TABLESPACE为Greenplum数据库系统注册并配置新的表空间。表空间名称必须不同于系统中任何现有表空间的名称。表空间是Greenplum数据库系统对象（全局对象），如果您具有适当的特权，则可以使用任何数据库中的表空间。

表空间允许超级用户定义包含数据库对象（例如表和索引）的数据文件所在的备用主机文件系统位置。

具有适当特权的用户可以将表空间名称传递给[CREATE DATABASE](#), [CREATE DATABASE](#)或[CREATE INDEX](#)，以将这些对象的数据文件存储在指定的表空间中。

在Greenplum数据库中，文件系统位置必须存在于所有主机上，包括运行master, standby, 每个primary和每个mirror的主机。

参数

tablespacename

要创建的表空间的名称。该名称不能以pg_或gp_开头，因为此类名称是为系统表空间保留的。

OWNER username

拥有表空间的用户名。如果省略，则默认为执行命令的用户。

只有超级用户可以创建表空间，但是他们可以将表空间的所有权分配给非超级用户。

LOCATION '/path/to/dir'

目录的绝对路径（主机系统文件位置），它将是表空间的根目录。注册表空间时，该目录应该为空，并且必须归Greenplum数据库系统用户所有。该目录必须由绝对路径名指定。

对于每个segment实例，您可以在WITH子句中为表空间指定不同的目录。

content *ID_i*'*/path/to/dir_i*

值*ID_i*是segment实例的content ID。*/path/to/dir_i*是segment实例用作表空间的根目录的主机系统文件位置的绝对路径。您不能指定master实例的content ID（-1）。您可以为多个segment指定相同的目录。

如果WITH子句中未列出segment实例，则Greenplum数据库将使用LOCATION子句指定的目录。

注册表空间时，目录应为空，并且必须由Greenplum数据库系统用户拥有。

注解

表空间仅在支持符号链接的系统上受支持。

CREATE TABLESPACE不能在事务块内执行。

创建表空间时，请确保文件系统位置具有足够的I/O速度和可用磁盘空间。

CREATE TABLESPACE创建从master和segment实例数据目录中的pg_tblspc目录到命令中指定目录的符号链接。

系统catalog表pg_tablespace存储表空间信息。此命令显示表空间的OID值，名称和所有者。

```
SELECT oid, spcname, spcowner FROM pg_tablespace ;
```

Greenplum数据库内置函

数gp_tablespace_location(*tablespace_oid*)显示所有segment实例的表空间主机系统文件位置。此命令列出OID为16385的表空间的segment数据库ID和主机系统文件位置。

```
SELECT * FROM gp_tablespace_location(16385)
```

示例

创建一个新表空间，并为master和所有segment实例指定文件系统位置：

```
CREATE TABLESPACE mytblspace LOCATION  
' /gpdbtspc/mytestspace' ;
```

创建一个新表空间，并为content ID为0和1的segment实例指定一个位置。对于未在WITH子句中列出的master和segment实例，在LOCATION子句中指定该表空间的文件系统位置。

```
CREATE TABLESPACE mytblspace LOCATION  
' /gpdbtspc/mytestspace' WITH (content0='/temp/mytest',  
content1='/temp/mytest');
```

该示例为两个segment实例指定了相同的位置。您可以为每个segment指定不同的位置。

兼容性

CREATE TABLESPACE是Greenplum数据库扩展。

另见

[CREATE DATABASE](#), [CREATE TABLE](#), [CREATE INDEX](#), [DROP TABLESPACE](#), [ALTER TABLESPACE](#)

Parent topic: [SQL Command Reference](#)

CONFIGURATION

Greenplum数据库® 6.0文档

定义新的文本搜索配置。

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

概要

```
CREATE TEXT SEARCH CONFIGURATION name (
    PARSER = parser_name |
    COPY = source_config
)
```

描述

`CREATE TEXT SEARCH CONFIGURATION`创建一个新的文本搜索配置。文本搜索配置指定了文本搜索解析器，该文本搜索解析器可以将字符串划分为标记，外加一些词典（可被用来决定哪些标记是搜索感兴趣的）。

如果仅指定了解析器，则新的文本搜索配置最初没有从标记类型到字典的映射，因此将忽略所有单词。随后的`ALTER TEXT SEARCH CONFIGURATION`命令必须用于创建映射以使配置有用。或者，可以复制现有的文本搜索配置。

如果指定了模式名称，则会在指定的模式中创建文本搜索配置。否则，它将在当前模式中创建。

定义文本搜索配置的用户将成为其所有者。

更多信息，请参考[使用全文搜索](#)。

参数

name

要创建的文本搜索配置的名称。该名称可以是模式的。

parser_name

用于此配置的文本搜索解析器的名称。

source_config

要复制的现有文本搜索配置的名称。

注解

PARSER和COPY选项是互斥的，因为复制现有配置时，它的解析器选择也会被复制。

兼容性

SQL标准中没有CREATE TEXT SEARCH CONFIGURATION语句。

另见

[ALTER TEXT SEARCH CONFIGURATION , DROP TEXT SEARCH CONFIGURATION](#)

Parent topic: [SQL Command Reference](#)

DICTIONARY

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLE

ALTER FOREIGN
DATA WRAPPER

ALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一个新的文本搜索字典。

概要

```
CREATE TEXT SEARCH DICTIONARY name (
    TEMPLATE = template
    [, option = value [, ... ]]
)
```

描述

CREATE TEXT SEARCH DICTIONARY 创建一个新的文本搜索字典。文本搜索字典指定一种识别有趣或不有趣的单词的搜索方式。词典取决于文本搜索模板，该模板指定了实际执行工作的函数。通常，字典提供一些选项来控制模板函数的详细行为。

如果指定了模式名称，则会在指定的模式中创建文本搜索字典。否则，它将在当前模式中创建。

定义文本搜索词典的用户将成为其所有者。

更多信息请参考[使用全文搜索](#)。

参数

name

要创建的文本搜索字典的名称。该名称可以由模式指定。

template

文本搜索模板的名称，它将定义此词典的基本行为。

option

要为此字典设置的模板特定选项的名称。

value

用于模板特定选项的值。如果该值不是简单的标识符或数字，则必须用引号引起（但如果需要，您可以始终用引号引起）。



来)。

选项可以按任何顺序出现。

示例

以下示例命令使用不标准的停用词列表创建一个基于Snowball的字典。

```
CREATE TEXT SEARCH DICTIONARY my_russian (
    template = snowball,
    language = russian,
    stopwords = myrussian
);
```

兼容性

SQL标准中没有CREATE TEXT SEARCH DICTIONARY语句。

另见

[ALTER TEXT SEARCH DICTIONARY](#) , [DROP TEXT SEARCH DICTIONARY](#)

Parent topic: [SQL Command Reference](#)

PARSER

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)

[ABORT](#)

[ALTER AGGREGATE](#)

[ALTER COLLATION](#)

[ALTER CONVERSION](#)

[ALTER DATABASE](#)

[ALTER DEFAULT PRIVILEGES](#)

[ALTER DOMAIN](#)

[ALTER EXTENSION](#)

[ALTER EXTERNAL TABLE](#)

[ALTER FOREIGN DATA WRAPPER](#)

[ALTER FOREIGN TABLE](#)

[ALTER FUNCTION](#)

[ALTER GROUP](#)

[ALTER INDEX](#)

[ALTER LANGUAGE](#)

[ALTER OPERATOR](#)

Description

定义一个新的文本搜索解析器。

概要

```
CREATE TEXT SEARCH PARSER name (
    START = start_function ,
    GETTOKEN = gettoken_function ,
    END = end_function ,
    LEXTYPES = lextypes_function
    [ , HEADLINE = headline_function ]
)
```

描述

CREATE TEXT SEARCH PARSER创建一个新的文本搜索解析器。文本搜索解析器定义了一种方法，用于将文本字符串拆分为标记并为标记分配类型（类别）。解析器本身并不是特别有用，但必须与一些用于搜索的文本搜索字典一起绑定到文本搜索配置中。

如果指定了模式名称，则会在指定的模式中创建文本搜索解析器。否则，它将在当前模式中创建。

您必须是超级用户才能使用CREATE TEXT SEARCH PARSER。（之所以做出此限制，是因为错误的文本搜索解析器定义可能会使服务器困惑甚至崩溃。）

更多信息请参考[使用全文搜索](#)。

参数

name



要创建的文本搜索解析器的名称。该名称可以由模式指定。

start_function

解析器的启动函数的名称。

gettoken_function

解析器的get-next-token函数的名称。

end_function

解析器的结束函数的名称。

lextypes_function

解析器的lextypes函数的名称（该函数返回有关它产生的标记类型集的信息）。

headline_function

解析器的标题函数的名称（该函数汇总一组标记）。

如有必要，可以对函数名称进行模式限定。由于每种函数类型的参数列表都是预先确定的，因此未提供参数类型。除标题函数外，所有其他函数都是必需的。

参数可以按任何顺序出现，而不仅仅是上面显示的顺序。 \

兼容性

SQL标准中没有CREATE TEXT SEARCH PARSE语句。

另见

[ALTER TEXT SEARCH PARSER, DROP TEXT SEARCH PARSER](#)

Parent topic: [SQL Command Reference](#)

TEMPLATE

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)

[ABORT](#)

[ALTER AGGREGATE](#)

[ALTER COLLATION](#)

[ALTER CONVERSION](#)

[ALTER DATABASE](#)

[ALTER DEFAULT PRIVILEGES](#)

[ALTER DOMAIN](#)

[ALTER EXTENSION](#)

[ALTER EXTERNAL TABLE](#)

[ALTER FOREIGN DATA WRAPPER](#)

[ALTER FOREIGN TABLE](#)

[ALTER FUNCTION](#)

[ALTER GROUP](#)

[ALTER INDEX](#)

[ALTER LANGUAGE](#)

[ALTER OPERATOR](#)

Description

定义一个新的文本搜索模板。

概要

```
CREATE TEXT SEARCH TEMPLATE name (
    [ INIT = init_function , ]
    LEXIZE = lexize_function
)
```

描述

`CREATE TEXT SEARCH TEMPLATE`创建一个新的文本搜索模板。文本搜索模板定义了实现文本搜索词典的函数。模板本身不是有用的，但必须实例化为要使用的字典。字典通常指定要提供给模板函数的参数。

如果指定了模式名称，则会在指定的模式中创建文本搜索模板。否则，它将在当前模式中创建。

您必须是超级用户才能使用`CREATE TEXT SEARCH TEMPLATE`。之所以做出此限制，是因为错误的文本搜索模板定义可能会使服务器混乱甚至崩溃。将模板与词典分开的原因是模板封装了定义词典的“不安全”方面。定义字典时可以设置的参数对于没有特权的用户是安全的，因此创建字典不必是特权操作。

更多信息请参考[使用全文搜索](#)。

参数

name



要创建的文本搜索模板的名称。该名称可以由模式指定。

init_function

模板的初始化函数的名称。

lexize_function

模板的lexize函数的名称。

如有必要，可以对函数名称进行模式限定。由于每种函数类型的参数列表都是预先确定的，因此未提供参数类型。`lexize`函数是必需的，但`init`函数是可选的。

参数可以以任何顺序出现，而不仅是上面显示的顺序。

兼容性

SQL标准中没有`CREATE TEXT SEARCH TEMPLATE`语句。

另见

[CREATE TEXT SEARCH TEMPLATE, DROP TEXT SEARCH TEMPLATE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT

PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义新的数据类型。

概要

```

CREATE TYPE name AS
( attribute_name data_type [ COLLATE collation ] [ , ... ] )
)

CREATE TYPE name AS ENUM
( [ 'label' [ , ... ] ] )

CREATE TYPE name AS RANGE (
    SUBTYPE = subtype
    [ , SUBTYPE_OPCLASS = subtype_operator_class ]
    [ , COLLATION = collation ]
    [ , CANONICAL = canonical_function ]
    [ , SUBTYPE_DIFF = subtype_diff_function ]
)

CREATE TYPE name (
    INPUT = input_function,
    OUTPUT = output_function
    [ , RECEIVE = receive_function ]
    [ , SEND = send_function ]
    [ , TYPMOD_IN = type_modifier_input_function ]
    [ , TYPMOD_OUT = type_modifier_output_function ]
    [ , INTERNALLENGTH = {internallength | VARIABLE} ]
    [ , PASSEDBYVALUE ]
    [ , ALIGNMENT = alignment ]
    [ , STORAGE = storage ]
    [ , LIKE = like_type ]
    [ , CATEGORY = category ]
    [ , PREFERRED = preferred ]
    [ , DEFAULT = default ]
    [ , ELEMENT = element ]
    [ , DELIMITER = delimiter ]
    [ , COLLATABLE = collatable ]
    [ , COMPRESSSTYPE = compression_type ]
    [ , COMPRESSLEVEL = compression_level ]
    [ , BLOCKSIZE = blocksize ] )
)

```

CREATE TYPE *name*

描述

CREATE TYPE注册一个新的数据类型以供当前数据库使用。 定义类



型的用户将成为其所有者。

如果提供了模式名称，则在指定的模式中创建类型。否则，它将在当前模式中创建。类型名称必须不同于同一模式中任何现有类型或域的名称。类型名称还必须与同一模式中任何现有表的名称都不同。

如上面的语法概要所示，有五种形式的CREATE TYPE。它们分别创建组合类型，枚举类型，范围类型，基础类型或shell类型。下面依次讨论其中的前四个。shell类型只是用于稍后定义的类型的占位符；它是通过发出CREATE TYPE来创建的，除了类型名称外，不带任何参数。如这些部分所述，在创建范围类型和基础类型时，需要使用Shell类型作为前向引用。

组合类型

第一种形式的CREATE TYPE创建组合类型。组合类型由一个属性名和数据类型的列表指定。如果属性的数据类型是可排序的，也可以指定该属性的排序规则。组合类型本质上和表的行类型相同，但是如果只想定义一种类型，使用CREATE TYPE避免了创建一个实际的表。单独的组合类型也是很有用的，例如可以作为函数的参数或者返回类型。

为了能够创建组合类型，必须拥有在其所有属性类型上的USAGE特权。

枚举类型

正如PostgreSQL文档中描述的[枚举类型](#)，第二种形式的CREATE TYPE创建枚举类型。枚举类型需要由一个或者更多带引号的标签构成的列表，每一个标签长度必须不超过NAMEDATALEN字节（在标准的PostgreSQL编译中是 64 字节）。

范围类型

正如[范围类型](#)描述，第三种形式的CREATE TYPE创建范围类型。

范围类型的*subtype*可以是任何带有一个相关的B树操作符类（用来决定该范围类型值的顺序）的类型。通常，子类型的默认B树操作符类被用来决定顺序。要使用一种非默认操作符类，可以用*subtype_opclass*指定它的名字。如果子类型是可排序的并且希望在该范围的顺序中使用一种非默认的排序规则，可以用*collation*选项来指定。

可选的*canonical*函数必须接受一个所定义的范围类型的参数，并且返回同样类型的一个值。在适用时，它被用来把范围值转换成一种规范的形式。更多信息请见[定义新的范围类型](#)。创建一个*canonical*函数有点棘手，因为必须在声明范围类型之前定义它。要这样做，必须首先

创建一种shell类型，它是一种没有属性只有名称和拥有者的占位符类型。这可以通过发出不带额外参数的命令CREATE TYPE *name*来完成。然后可以使用该shell类型作为参数和结果来声明该函数，并且最终用同样的名称来声明范围类型。这会自动用一种合法的范围类型替换shell类型项。

可选的*subtype_diff*函数必须接受两个*subtype*类型的值作为参数，并且返回一个double precision值表示两个给定值之间的差别。虽然这是可选的，但是提供这个函数会让该范围类型列上GiST索引效率更高。更多信息请参考[定义新的范围类型](#)。

Base Types

第四种形式的CREATE TYPE创建一种新的基础类型（标量类型）。为了创建一种新的基础类型，你必须是一个超级用户。参数可以以任意顺序出现（而不仅是按照语法所示的顺序），并且大部分是可选的。在定义类型前，必须注册两个或者更多函数（使用CREATE FUNCTION）。支持函数*input_function*以及*output_function*是必需的，而函

数*receive_function*、*send_function*、*type_modifier_input_function*、*type_modifier_output_function*和*analyze_function*是可选的。通常来说这些函数必须是用C或者另外一种低层语言编写的。在Greenplum数据库中，用于实现数据类型的任何函数都必须定义为IMMUTABLE。

*input_function*将类型的外部文本表达转换成为该类型定义的操作符和函数所使用的内部表达。*output_function*执行反向的转换。输入函数可以被声明为有一个cstring类型的参数，或者有三个类型分别为cstring、oid、integer的参数。第一个参数是以C字符串存在的输入文本，第二个参数是该类型自身的OID（对于数组类型则是其元素类型的OID），第三个参数是目标列的typmod，如果知道（不知道则将传递-1）。输入函数必须返回一个该数据类型本身的价值。通常，一个输入函数应该被声明为STRICT。如果不是这样，在读到一个NULL输入值时，调用它时第一个参数会是NULL。在这种情况下，该函数必须仍然返回NULL，除非它发生了错误（这种情况主要是想支持域输入函数，它们可能需要拒绝NULL输入）。输出函数必须被声明为有一个新数据类型的参数。输出函数必须返回类型cstring。对于NULL值不会调用输出函数。

可选的*receive_function*会把类型的外部二进制表达转换成内部表达。如果没有提供这个函数，该类型不能参与到二进制输入中。二进制表达转换成内部形式代价更低，然而却更容易移植（例如，标准的整数数据类型使用网络字节序作为外部二进制表达，而内部表达是机器本地的字节序）。接收函数应该执行足够的检查以确保该值是有效的。接收函数可以被声明为有一个internal类型的参数，或者有三个类型分别为internal、oid、integer的参数。第一个参数是一个指

向`StringInfo`缓冲区的指针，其中保存着接收到的字节串。其余可选的参数和文本输入函数的相同。接收函数必须返回一个该数据类型本身的值。通常，一个接收函数应该被声明为`STRICT`。如果不是这样，在读到一个`NULL`输入值时，调用它时第一个参数会是`NULL`。在这种情况下，该函数必须仍然返回`NULL`，除非它发生了错误（这种情况下主要是想支持域接收函数，它们可能需要拒绝`NULL`输入）。类似地，可选的`send_function`将内部表达转换成外部二进制表达。如果没有提供这个函数，该类型将不能参与到二进制输出中。发送函数必须被声明为有一个新数据类型的参数。发送函数必须返回类型`bytea`。对于`NULL`值不会调用发送函数。

如果类型支持修饰符，则需要可选的`type_modifier_input_function`和`type_modifier_output_function`。修饰符是附加到类型声明的可选约束，例如`char(5)`或`numeric(30,2)`。尽管Greenplum数据库允许用户定义的类型将一个或多个简单的常量或标识符用作修饰符，但此信息必须适合单个非负整数值，以存储在系统catalog中。Greenplum数据库以`cstring`数组的形式将声明的修饰符传递给`type_modifier_input_function`。修饰符输入函数必须检查值的有效性，如果值不正确，则会引发错误。如果值正确，则修饰符输入函数将返回单个非负整数值，该值将被Greenplum数据库存储为`typmod`列。如果未使用`type_modifier_input_function`定义类型，则拒绝类型修饰符。`type_modifier_output_function`将内部整数`typmod`值转换回正确的格式以供用户显示。修饰符输出函数必须返回一个`cstring`值，该值是要附加到类型名称后的确切字符串。例如，数字函数可能返回`(30,2)`。`type_modifier_output_function`是可选的。如果未指定，则默认显示格式为括号内存储的`typmod`整数值。

到这里你应该在疑惑输入和输出函数是如何能被声明为具有新类型的结果或参数的。因为必须在创建新类型之前创建这两个函数。这个问题的答案是，新类型应该首先被定义为一种shell类型，它是一种占位符类型，除了名称和拥有者之外它没有其他属性。这可以通过不带额外参数的命令`CREATE TYPE name`做到。然后I/O函数可以被定义为引用这种shell类型。最后，用带有完整定义的`CREATE TYPE`把该shell类型替换为一个完全的、合法的类型定义，之后新类型就可以正常使用了。

`like_type`参数提供了一种用于指定数据类型的基本表示形式属性的替代方法：从某些现有类型中复制它们。从指定类型复制值`internallength`, `passedbyvalue`, `alignment`和`storage`。（尽管通常不希望这样做，但可以通过与`LIKE`子句一起指定它们来覆盖其中的某些值。）以这种方式指定表示形式对于在现有类型中新类型“搭载”的低级实现在某种方式上非常有用。

虽然新类型的内部表示的详细信息仅由I/O函数和您创建的与该类型一起使用的其他函数才知道，但是内部表示的一些属性必须声明给Greenplum数据库。其中最重要的是*internallength*。基础数据类型可以是固定长度，在这种情况下，*internallength*是正整数），也可以是可变长度，通过将*internallength*设置为VARIABLE表示）。（内部，这是通过将typelen设置为-1来表示的。）所有可变长度类型的内部表示必须以4字节整数开头，并给出该类型值的总长度。

可选的标志PASSEDBYVALUE表示这种数据类型的值需要被传值而不是传引用。您不能传递内部表示形式大于Datum类型的大小的值类型（大多数计算机上为4字节，少数计算机上为8字节）。

*alignment*参数指定数据类型的存储对齐要求。允许的值等同于以1、2、4或8字节边界对齐。注意变长类型的参数必须至少按4字节对齐，因为它们需要包含一个int4作为它们的第一个组成部分。

*storage*参数允许为变长数据类型选择存储策略（对定长类型只允许plain）。plain指定该类型的数据将总是被存储在线内并且不会被压缩。extended指定系统将首先尝试压缩一个长的数据值，并且将在数据仍然太长的情况下把值移出主表行。external允许值被移出主表，但是系统将不会尝试对它进行压缩。main允许压缩，但是不鼓励把值移出主表。（如果没有其他办法让行的大小变得合适，具有这种存储策略的数据项仍将被移出主表，但比起extended以及external项，这种存储策略的数据项会被优先考虑保留在主表中）。

如果用户希望数据类型的列默认为空值以外的其他值，则可以指定一个默认值。使用DEFAULT关键字指定默认值。（这种默认值可能会被附加到特定列的显式DEFAULT子句覆盖。）

要指示类型是数组，请使用ELEMENT关键字指定数组元素的类型。例如，要定义一个4字节整数(int4)的数组，请指定ELEMENT = int4。有关数组类型的更多详细信息显示在下面。

*category*和*preferred*参数可以被用来帮助控制在混淆的情况下应用哪一种隐式造型。每种数据类型均属于以单个ASCII字符命名的类别，并且每种类型均为“首选”或不在其类别内。当此规则有助于解析重载的函数或运算符时，解析器将更喜欢强制转换为首选类型（但只能从同一类别中的其他类型）。对于没有隐式转换为其他类型或从任何其他类型隐式转换的类型，保留默认设置就足够了。但是，对于具有隐式强制转换的一组相关类型，将它们全部标记为属于一个类别并选择一个或两个“最一般”类型作为该类别中的首选类型通常是有帮助的。当您将用户定义的类型（例如数字或字符串类型）添加到现有内置类别时，*category*参数特别有用。也可以创建新的完全由用户定义的类型类别。选择除大写字母以外的任何ASCII字符以命名此类。

为了指示在此类型的数组的外部表示形式中的值之间使用定界符，可以将定界符设置为特定字符。默认的分隔符是逗号（,）。注意，分隔符与数组元素类型相关联，而不与数组类型本身相关联。

如果可选的布尔参数 `collatable` 为 `true`，则类型的列定义和表达式可以通过使用 `COLLATE` 子句来携带排序规则信息。在该类型上操作的函数的实现负责真正利用这些信息，仅把类型标记为可排序的并不会让它们自动地去使用这类信息。

数组类型

每当创建用户定义的类型时，Greenplum数据库都会自动创建一个关联的数组类型，其名称由元素类型的名称组成，该名称前面带有下划线，并在必要时将其截断以使其长度小于 `NAMEDATALEN` 字节。（如果这样生成的名称与现有类型名称冲突，则重复该过程，直到找到一个非冲突名称为止。）此隐式创建的数组类型为可变长度，并使用内置的输入和输出函数 `array_in` 和 `array_out`。数组类型跟踪其元素类型的所有者或模式中的所有更改，并且在元素类型被删除时也被删除。

如果系统会自动地创建正确的数组类型，你可能会理所当然地问为什么会有个 `ELEMENT` 选项。使用 `ELEMENT` 唯一有用的情况是：当你在创建一种定长类型，它正好在内部是一个多个相同东西的数组，并且除了计划给该类型提供的整体操作之外，你想要允许用下标来直接访问这些东西。例如，类型 `point` 被表示为两个浮点数，可以使用 `point[0]` 以及 `point[1]` 来访问它们。注意，这种功能只适用于内部形式正好是一个相同定长域序列的定长类型。可用下标访问的变长类型必须具有 `array_in` 以及 `array_out` 使用的一般化的内部表达。由于历史原因（即很明显是错的，但现在改已经太晚了），定长数组类型的下标是从零开始的，而不是像变长数组那样。

参数

name

要创建的类型的名称（可以由模式指定）。

attribute_name

复合类型的属性（列）的名称。

data_type

要成为复合类型的列的现有数据类型的名称。

collation

与复合类型的列或范围类型关联的现有排序规则的名称。

label

字符串文字，表示与枚举类型的一个值相关联的文本标签。

subtype

范围类型将代表其范围的元素类型的名称。

subtype_operator_class

子类型的b树运算符类的名称。

canonical_function

范围类型的规范化函数的名称。

subtype_diff_function

子类型的差函数的名称。

input_function

将数据从类型的外部文本形式转换为内部形式的函数的名称。

output_function

将数据从类型的内部形式转换为外部文本形式的函数名。

receive_function

将数据从类型的外部二进制形式转换成内部形式的函数名。

send_function

将数据从类型的内部形式转换为外部二进制形式的函数名。

type_modifier_input_function

将类型的修饰符数组转换为内部形式的函数名。

type_modifier_output_function

将类型的修饰符的内部形式转换为外部文本形式的函数名。

internallength

一个数字常量，指定新类型的内部表示形式的长度（以字节为单位）。默认假设是可变长度。

alignment

该数据类型的存储对齐需求。如果被指定，它必须是char、int2、int4或者double。默认是int4。

storage

该数据类型的存储策略。必须是plain、external、extended或者main。默认是plain。

like_type

与新类型具有相同表达的现有数据类型的名称。会从这个类型中复制*internallength*、*passedbyvalue*、*alignment*以及*storage*的值，除非在这个CREATE TYPE命令的其他地方用显式说明覆盖。

category

此类型的类别代码（单个ASCII字符）。默认值为'U'，表示用户定义的类型。您可以在[pg_type类别代码](#)中找到其他标准类别代码。您还可以将未使用的ASCII字符分配给您创建的自定义类别。

preferred

如果此类型是其类型类别中的首选类型，则为true，否则为false。默认值为false。在现有类型类别中创建新的首选类型时要小心；这可能会导致令人惊讶的行为更改。

default

数据类型的默认值。如果省略，则默认为null。

element

创建的类型是一个数组；这指定了数组元素的类型。

delimiter

在此类型的数组中的值之间使用定界符。

collatable

如果此类型的操作可以使用排序规则信息，则为true。默认为false。

compression_type

设置为ZLIB（默认值），ZSTD，RLE_TYPE或QUICKLZ¹以指定在此类型的列中使用的压缩类型。

Note: ¹QuickLZ压缩仅在Pivotal Greenplum数据库的商业版本中可用。

compression_level

对于Zstd压缩，将其设置为1（最快压缩）到19（最高压缩率）之间的整数值。对于zlib压缩，有效范围是1到9。QuickLZ压缩级别只能设置为1。对于RLE_TYPE，压缩级别可以设置为从1（最快压缩）到4（最高压缩率）的整数值。默认压缩级别为1。

blocksize

设置为列中每个块的大小（以字节为单位）。BLOCKSIZE必须介于8192和2097152字节之间，并且是8192的倍数。默认块大小为32768。

注解

用户定义的类型名称不能以下划线字符（_）开头，并且只能为62个字符长（或者通常为NAMEDATALEN - 2，而不是其他名称允许的NAMEDATALEN - 1个字符）。以下划线开头的类型名称保留给内部创建的数组类型名称。

Greenplum数据库不支持为行或复合类型添加存储选项。

在表级和列级定义的存储选项将覆盖为标量类型定义的默认存储选项。

因为一旦创建数据类型就没有使用限制，所以创建基础类型或范围类型无异于授予对类型定义中提到的函数的公共执行权限。（因此，类型的创建者必须拥有这些函数。）对于在类型定义中有用的各种函数而言，这通常不是问题。但是您在设计一种在将其转换为外部格式或从外部格式转换要求使用“秘密”信息的类型时，可能需要三思而后行。

示例

本示例创建一个复合类型并将其用于函数定义中：

```
CREATE TYPE compfoo AS (f1 int, f2 text);

CREATE FUNCTION getfoo() RETURNS SETOF compfoo AS $$ 
    SELECT fooid, fooname FROM foo
$$ LANGUAGE SQL;
```

本示例创建枚举类型mood，并在表定义中使用它。

```
CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy');
CREATE TABLE person (
    name text,
    current_mood mood
);
INSERT INTO person VALUES ('Moe', 'happy');
SELECT * FROM person WHERE current_mood = 'happy';
   name | current_mood
-----+-----
 Moe   | happy
(1 row)
```

本示例创建一个范围类型：

```
CREATE TYPE float8_range AS RANGE (subtype = float8,
subtype_diff = float8mi);
```

本示例创建基础数据类型box，然后在表定义中使用该类型：

```
CREATE TYPE box;

CREATE FUNCTION my_box_in_function(cstring) RETURNS box AS
... ;

CREATE FUNCTION my_box_out_function(box) RETURNS cstring AS
... ;

CREATE TYPE box (
    INTERNALLENGTH = 16,
    INPUT = my_box_in_function,
    OUTPUT = my_box_out_function
);

CREATE TABLE myboxes (
    id integer,
    description box
);
```

如果box的内部结构是四个float4元素的数组，则可以改用：

```
CREATE TYPE box (
    INTERNALLENGTH = 16,
    INPUT = my_box_in_function,
    OUTPUT = my_box_out_function,
    ELEMENT = float4
);
```

这将允许通过下标访问box值的组件号。否则，该类型的行为与以前相同。

本示例创建一个大型对象类型，并在表定义中使用它：

```
CREATE TYPE bigobj (
    INPUT = lo_filein, OUTPUT = lo_fileout,
    INTERNALLENGTH = VARIABLE
);

CREATE TABLE big_objs (
    id integer,
    obj bigobj
);
```

兼容性

创建复合类型的CREATE TYPE命令的第一种形式符合SQL标准。其他形式是Greenplum数据库扩展。SQL标准中的CREATE TYPE语句还定义了Greenplum数据库中未实现的其他形式。

创建具有零属性的复合类型的能力是Greenplum数据库特定于标准的偏差（类似于CREATE TABLE中的相同情况）。

另见

[ALTER TYPE](#) , [CREATE DOMAIN](#) , [CREATE FUNCTION](#) , [DROP TYPE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

 参考指南 SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

默认情况下，使用LOGIN特权定义一个新的数据库角色。

概要

CREATE USER *name* [[WITH] *option* [...]]其中*option*可以是：

```

SUPERUSER | NOSUPERUSER
| CREATEDB | NOCREATEDB
| CREATEROLE | NOCRAEROLE
| CREATEUSER | NOCREATEUSER
| CREATEEXTTABLE | NOCREATEEXTTABLE
[ ( attribute='value'[, ...] ) ]
    where attributes and value are:
        type='readable' | 'writable'
        protocol='gpfdist' | 'http'
| INHERIT | NOINHERIT
| LOGIN | NOLOGIN
| REPLICATION | NOREPLICATION
| CONNECTION LIMIT connlimit
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
| VALID UNTIL 'timestamp'
| IN ROLE role_name [, ...]
| IN GROUP role_name
| ROLE role_name [, ...]
| ADMIN role_name [, ...]
| USER role_name [, ...]
| SYSID uid
| RESOURCE QUEUE queue_name
| RESOURCE GROUP group_name
| [ DENY deny_point ]
| [ DENY BETWEEN deny_point AND deny_point ]

```

描述

CREATE USER是CREATE ROLE的别名。

CREATE ROLE和CREATE USER之间的唯一区别是CREATE USER默认情况下假定LOGIN，而CREATE ROLE默认情况下假定NOLOGIN。

兼容性

SQL标准中没有CREATE USER语句。

另见

[CREATE ROLE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义用户到外部服务器的新映射。

概要

```
CREATE USER MAPPING FOR { username | USER | CURRENT_USER | PUBLIC }
    SERVER servername
    [ OPTIONS ( option 'value' [, ...] ) ]
```

描述

`CREATE USER MAPPING` 定义用户到外部服务器的映射。您必须是服务器的所有者才能为其定义用户映射。

参数

username

映射到外部服务器的现有用户的名称。

CURRENT_USER和USER与当前用户的名称匹配。PUBLIC用于匹配系统中所有当前和将来的用户名。

servername

Greenplum数据库要为其创建用户映射的现有服务器的名称。

OPTIONS (*option 'value'* [, ...])

新用户映射的选项。这些选项通常定义映射的实际用户名和密码。选项名称必须唯一。选项名称和值特定于服务器的外部数据包装器。

示例

为用户bob和服务器foo创建用户映射：



```
CREATE USER MAPPING FOR bob SERVER foo OPTIONS (user 'bob',  
password 'secret');
```

兼容性

CREATE USER MAPPING 符合 ISO/IEC 9075-9 (SQL/MED)。

另见

[ALTER USER MAPPING](#), [DROP USER MAPPING](#), [CREATE FOREIGN DATA WRAPPER](#), [CREATE SERVER](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一个新的视图。

概要

```
CREATE [OR REPLACE] [TEMP | TEMPORARY] [RECURSIVE] VIEW
name
      [ ( column_name [, ...] ) ]
      [ WITH ( view_option_name [= view_option_value] [, ...
] ) ]
      AS query
      [ WITH [ CASCDED | LOCAL ] CHECK OPTION ]
```

描述

CREATE VIEW 定义查询的视图。该视图未物化。而是，每次在查询中引用视图时运行查询。

CREATE OR REPLACE VIEW 类似，但是如果已经存在相同名称的视图，则将其替换。新查询必须生成与现有视图查询生成列相同的列（即，相同的列名以相同的顺序，并且具有相同的数据类型），但是它可能会将其他列添加到列表的末尾。产生输出列的计算可能完全不同。

如果指定了模式名称，则将在指定的模式中创建视图。否则，它将在当前模式中创建。临时视图存在于特殊的模式中，因此在创建临时视图时可能不会给出模式名称。视图的名称必须与同一模式中的任何其他视图，表，序列，索引或外部表的名称不同。

参数

TEMPORARY | TEMP

如果指定，则将视图创建为临时视图。临时视图会在当前会话结束时自动删除。存在临时视图时，当前会话将看不到具有相同名称的已有永久表，除非使用模式限定名称来引用它们。如果该视图引用的任何表都是临时表，则该视图将被创建为临时视图（无论是否指定TEMPORARY）。

RECURSIVE

创建一个递归视图。语法

```
CREATE RECURSIVE VIEW [ schema . ] view_name
(column_names) AS SELECT ...;
```

等价于

```
CREATE VIEW [ schema . ] view_name AS WITH RECURSIVE
view_name (column_names) AS (SELECT ...) SELECT
column_names FROM view_name;
```

必须为递归视图指定视图列名称列表。

name

要创建的视图的名称（可以由模式指定）。

column_name

用于视图列的名称的可选列表。如果未给出，则从查询中推导出列名。

WITH (view_option_name [=view_option_value] [, ...])

此子句指定视图的可选参数；支持以下参数：

check_option (string)

该参数可以是local，也可以是cascaded，等效于指定**WITH [CASCADED | LOCAL] CHECK OPTION**（请参阅下文）。可以使用[ALTER VIEW](#)在现有视图上更改此选项。

security_barrier (boolean)

如果视图旨在提供行级安全性，则应使用此方法。

query

[SELECT](#)或[VALUES](#)命令，将提供视图的列和行。

注解

Greenplum数据库中的视图为只读。系统不允许在视图上插入、更新或删除。通过将视图上的重写规则创建为其他表上的适当操作，可以得到可更新视图的效果。有关更多信息，请参见[CREATE RULE](#)。

请注意，视图列的名称和数据类型将按照您想要的方式分配。例如：

```
CREATE VIEW vista AS SELECT 'Hello World';
```

是两种形式的错误形式：列名默认为?column?，列数据类型默认为unknown。如果要在视图结果中使用字符串文字，请使用类似以下

内容的方法：

```
CREATE VIEW vista AS SELECT text 'Hello World' AS hello;
```

对视图中引用的表的访问权限是由视图所有者而不是当前用户（即使当前用户是超级用户）的权限决定的。对于超级用户，这可能会造成混淆，因为超级用户通常可以访问所有对象。在视图的情况下，如果超级用户不是视图的所有者，那么即使超级用户也必须被明确授予访问该视图中引用的表的权限。

但是，对视图中调用的函数的处理方式与使用视图直接从查询中直接调用它们的方式相同。因此，视图的用户必须具有调用该视图使用的任何函数的权限。

如果使用ORDER BY子句创建视图，则从视图执行SELECT时将忽略ORDER BY子句。

在现有视图上使用CREATE OR REPLACE VIEW时，仅更改视图定义的SELECT规则。其他视图属性（包括所有权，权限和non-SELECT规则）保持不变。您必须拥有视图才能替换它（包括作为拥有角色的成员）。

示例

创建一个由所有喜剧电影组成的视图：

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

这将创建一个视图，其中包含在创建视图时film表中的列。尽管使用*来创建视图，但是稍后添加到表中的列将不属于视图。

创建一个获取前十名婴儿名字的视图：

```
CREATE VIEW topten AS SELECT name, rank, gender, year FROM
names, rank WHERE rank < '11' AND names.id=rank.id;
```

创建一个由1到100的数字组成的递归视图：

```
CREATE RECURSIVE VIEW public.nums_1_100 (n) AS
VALUES (1)
UNION ALL
SELECT n+1 FROM nums_1_100 WHERE n < 100;
```

请注意，尽管此CREATE VIEW命令中的递归视图名称是模式限定的，但是其内部自引用不是模式限定的。这是因为隐式创建的CTE名称不能通过模式限定。

兼容性

SQL标准为CREATE VIEW语句指定了Greenplum数据库中没有的一些附加功能。 标准中完整SQL命令的可选子句为：

- **CHECK OPTION** — 此选项与可更新视图有关。 将检查视图上的所有INSERT和UPDATE命令，以确保数据满足视图定义条件（即，新数据将通过视图可见）。 如果不这样做，更新将被拒绝。
- **LOCAL** — 在此视图上检查完整性。
- **CASCDED** — 在此视图和任何从属视图上检查完整性。 如果未指定CASCDED或LOCAL，则假定为CASCDED。

CREATE OR REPLACE VIEW是Greenplum数据库语言的扩展。 临时视图的概念也是如此。

另见

[SELECT , DROP VIEW](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)[ABORT](#)[ALTER AGGREGATE](#)[ALTER COLLATION](#)[ALTER CONVERSION](#)[ALTER DATABASE](#)[ALTER DEFAULT PRIVILEGES](#)[ALTER DOMAIN](#)[ALTER EXTENSION](#)[ALTER EXTERNAL TABLE](#)[ALTER FOREIGN DATA WRAPPER](#)[ALTER FOREIGN TABLE](#)[ALTER FUNCTION](#)[ALTER GROUP](#)[ALTER INDEX](#)[ALTER LANGUAGE](#)[ALTER OPERATOR](#)

取消分配预编译语句。

概要

```
DEALLOCATE [ PREPARE ] name
```

描述

DEALLOCATE用于取消分配先前预编译的SQL语句。如果未显式取消分配预编译语句，则在会话结束时将其释放。

有关预编译语句的更多信息，请参见[PREPARE](#)。

参数

PREPARE

可选关键字，将被忽略。

name

要取消分配的预编译语句的名称。

示例

取消分配先前预编译的名为insert_names的语句：

```
DEALLOCATE insert_names;
```

兼容性

SQL标准包括DEALLOCATE语句，但仅用于嵌入式SQL。



另见

[EXECUTE](#) , [PREPARE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义一个游标。

概要

```
DECLARE name [BINARY] [INSENSITIVE] [NO SCROLL] CURSOR
      [{WITH | WITHOUT} HOLD]
      FOR query [FOR READ ONLY]
```

描述

DECLARE允许用户创建游标，该游标可用于一次从较大查询中检索少量行。游标可以使用[FETCH](#)以文本或二进制格式返回数据。

Note: 本页在SQL命令级别描述游标的用法。如果试图在PL/pgSQL函数中使用游标，则规则是不同的。参见[Greenplum PL/pgSQL过程语言](#)。

普通游标以文本格式返回数据，与SELECT会产生相同的结果。由于数据本身以二进制格式存储，因此系统必须进行转换以产生文本格式。一旦信息以文本形式返回，客户端应用程序可能需要将其转换为二进制格式以进行操作。另外，文本格式的数据通常比二进制格式的数据大。二进制游标以二进制表示形式返回数据，可能更易于操作。但是，如果您仍然打算将数据显示为文本，则以文本形式检索数据将节省您在客户端的工作量。

例如，如果查询从整数列返回值1，则将使用默认游标获得字符串1，而使用二进制游标将获得包含值的内部表示形式的4字节字段（在大端字节序）。

二进制游标应谨慎使用。许多应用程序，包括psql，都不准备处理二进制游标，并希望数据以文本格式返回。

Note:

当客户端应用程序使用“扩展查询”协议发出FETCH命令时，“绑定协议”消息指定是以文本还是二进制格式检索数据。此选择掩盖定义光标的方式。因此，当使用扩展查询协议时，二进制游标的概念就已过时了 - 任何游标都可以视为文本或二进制。

可以在`UPDATE`或`DELETE`语句的`WHERE CURRENT OF`子句中指定游标，以更新或删除表数据。`UPDATE`或`DELETE`语句只能在服务器上执行，例如在交互式`psql`会话或脚本中。语言扩展（例如PL/pgSQL）不支持可更新的游标。

参数

name

要创建的游标的名称。

BINARY

使光标返回二进制而不是文本格式的数据。

INSENSITIVE

指示在存在游标时，从游标检索的数据应不受游标基于的表更新的影响。在Greenplum数据库中，所有游标都不敏感。该关键字当前不起作用，出现是为了与SQL标准兼容。

NO SCROLL

游标不能用于以非顺序方式检索行。这是Greenplum数据库中的默认行为，因为不支持滚动游标（SCROLL）。

WITH HOLD

WITHOUT HOLD

`WITH HOLD`指定在成功创建游标的事务提交后可以继续使用游标。`WITHOUT HOLD`指定不能在创建游标的事务之外使用游标。默认为`WITHOUT HOLD`。

当`query`包含`FOR UPDATE`或`FOR SHARE`子句时，不能指定`WITH HOLD`。

query

`SELECT`或`VALUES`命令，它将提供游标要返回的行。

如果在`UPDATE`或`DELETE`命令的`WHERE CURRENT OF`子句中使用了游标，则`SELECT`命令必须满足以下条件：

- 无法引用视图或外部表。

- 仅引用一张表。

该表必须是可更新的。例如，以下内容不可更新：表函数，返回集合的函数，仅追加表，列式表。

- 不能包含以下任何内容：

- 分组子句
- 集合操作，例如`UNION ALL`或`UNION DISTINCT`
- 排序子句
- 窗口子句
- 连接或自连接

在SELECT命令中指定FOR UPDATE子句可防止其他会话在获取行和更新行之间更改行。如果没有FOR UPDATE子句，则在创建游标以后更改了行，随后将UPDATE或DELETE命令与WHERE CURRENT OF子句一起使用将无效。

Note: 在SELECT命令中指定FOR UPDATE子句将锁定整个表，而不只是选定的行。

FOR READ ONLY

FOR READ ONLY表示光标以只读模式使用。

注解

除非指定了WITH HOLD，否则此命令创建的游标只能在当前事务中使用。因此，不带WITH HOLD的DECLARE在事务块之外是无用的：游标只能生存到语句完成。因此，如果在事务块外部使用此命令，Greenplum数据库将报告错误。使用BEGIN和COMMIT（或ROLLBACK）定义事务块。

如果指定了WITH HOLD，并且成功提交了创建游标的事务，则该游标可以继续被同一会话中的后续事务访问。（但是，如果创建事务被中止，则删除游标。）当对它发出显式CLOSE命令或会话结束时，将关闭用WITH HOLD创建的游标。在当前实现中，由保留的游标表示的行被复制到临时文件或存储区中，以便它们可用于后续事务。

如果在事务中使用DECLARE命令创建游标，则只有在使用CLOSE命令关闭游标后才能在事务中使用SET命令。

Greenplum数据库当前不支持可滚动光标。您只能使用FETCH将光标位置向前移动，而不能向后移动。

追加优化表不支持DECLARE...FOR UPDATE。

您可以通过查询pg_cursors系统视图来查看所有可用的游标。

示例

声明一个游标：

```
DECLARE mycursor CURSOR FOR SELECT * FROM mytable;
```

兼容性

SQL标准仅允许在嵌入式SQL和模块中使用游标。 Greenplum数据库允许以交互方式使用游标。

Greenplum数据库未为游标实现OPEN语句。 当游标被声明时，它被认为打开的。

SQL标准允许游标向前和向后移动。 所有Greenplum数据库游标仅向前移动（不可滚动）。

二进制游标是Greenplum数据库扩展。

另见

[CLOSE](#), [DELETE](#), [FETCH](#), [MOVE](#), [SELECT](#), [UPDATE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

从表中删除行。

概要

```
[ WITH [ RECURSIVE ] with_query [ , ... ] ]
DELETE FROM [ONLY] table [[AS] alias]
[USING usinglist]
[WHERE condition | WHERE CURRENT OF cursor_name]
[RETURNING * | output_expression [[AS] output_name]
[ , ... ]]
```

描述

`DELETE`从指定的表中删除满足`WHERE`子句的行。如果`WHERE`子句不存在，则结果是删除表中的所有行。结果是一个有效但空的表。

默认情况下，`DELETE`将删除指定表及其所有子表中的行。如果只希望从提到的特定表中删除，则必须使用`ONLY`子句。

使用数据库中其他表中包含的信息删除表中的行有两种方法：使用子选择，或在`USING`子句中指定其他表。哪种技术更合适取决于具体情况。

如果指定了`WHERE CURRENT OF`子句，则删除的行是从指定游标中最新获取的行。

复制表不支持`WHERE CURRENT OF`子句。

可选的`RETURNING`子句使`DELETE`根据实际删除的每一行计算并返回值。可以计算使用该表的列和/或`USING`中提到的其他表的列的任何表达式。`RETURNING`列表的语法与`SELECT`的输出列表的语法相同。

您必须对表具有`DELETE`特权才能从中删除。

Note: 默认情况下，Greenplum数据库为堆表上的`DELETE`操作获取表上的`EXCLUSIVE`锁。启用全局死锁检测器后，堆表上`DELETE`操作的锁定模式为`ROW EXCLUSIVE`。请参阅[全局死锁检测](#)。

输出

成功完成后， DELETE命令将返回以下形式的命令标签：

```
DELETE count
```

*count*是已删除的行数。 如果*count*为0，则查询没有删除任何行（这不视为错误）。

如果DELETE命令包含RETURNING子句，则结果将类似于SELECT语句的结果，该SELECT语句包含RETURNING列表中定义的列和值，该列和值是在该命令删除的行上计算的。

参数

with_query

WITH子句允许您指定一个或多个子查询，这些子查询可以在DELETE查询中按名称引用。

对于包含WITH子句的DELETE命令，该子句只能包含SELECT语句，而WITH子句不能包含数据修改命令
(INSERT, UPDATE或DELETE)。

请参见[WITH查询 \(公用表表达式\)](#) 和[SELECT](#)获取详细信息。

ONLY

如果指定，则仅从命名表中删除行。 如果未指定，还将处理从命名表继承的任何表。

table

现有表的名称（可以用模式指定）。

alias

目标表的替代名称。 提供别名后，它将完全隐藏表的实际名称。 例如，给定DELETE FROM foo AS f，DELETE语句的其余部分必须将此表称为f而不是foo。

usinglist

表表达式的列表，允许其他表中的列以WHERE条件出现。 这类类似于可以在[SELECT](#)语句的FROM子句中指定的表的列表。 例如，可以指定表名的别名。 除非您希望设置自连接，否则不要在usinglist中重复目标表。

condition

该表达式返回一个布尔类型的值，该值确定要删除的行。

cursor_name

在WHERE CURRENT OF条件中使用的游标名称。 要删除的行是从该游标中最近获取的行。 游标必须是对DELETE目标表的简单非分组查询。

不能与布尔条件一起指定WHERE CURRENT OF。

游标语句的DELETE... WHERE CURRENT OF游标语句只能在

服务器上执行，例如在交互式psql会话或脚本中。语言扩展（例如PL/pgSQL）不支持可更新的游标。

有关创建游标的更多信息，请参见[DECLARE](#)。

output_expression

删除每一行后，由DELETE命令计算并返回的表达式。该表达式可以使用在USING中列出的一个或多个表的任何列名。输入*以返回所有列。

output_name

用于返回的列的名称。

注解

Greenplum数据库允许您通过在USING子句中指定其他表来引用WHERE条件下其他表的列。例如，从rank表中将其命名为Hannah，可以这样做：

```
DELETE FROM rank USING names WHERE names.id = rank.id AND
name = 'Hannah';
```

这里实际上发生的是rank和names之间的联接，所有成功联接的行都标记为删除。此语法不是标准语法。但是，这种连接样式通常比更标准的子选择样式更容易编写和执行，例如：

```
DELETE FROM rank WHERE id IN (SELECT id FROM names WHERE
name
= 'Hannah');
```

当使用DELETE删除表的所有行时（例如：DELETE * FROM *table*），Greenplum数据库会添加一个隐式的TRUNCATE命令（当用户权限允许时）。添加的TRUNCATE命令释放被删除的行占用的磁盘空间，而无需对表进行VACUUM。这提高了后续查询的扫描性能，并使经常在临时表中插入和删除的ELT工作负载受益。

不支持在分区表的特定分区（子表）上直接执行UPDATE和DELETE命令。而是必须在根分区表（使用CREATE TABLE命令创建的表）上执行这些命令。

示例

删除除musicals以外的所有films：

```
DELETE FROM films WHERE kind <> 'Musical';
```

清除表films：

```
DELETE FROM films;
```

删除已完成的任务，返回已删除行的完整详细信息：

```
DELETE FROM tasks WHERE status = 'DONE' RETURNING *;
```

使用联接删除：

```
DELETE FROM rank USING names WHERE names.id = rank.id AND  
name = 'Hannah';
```

兼容性

该命令符合SQL标准，不同之处在于USING和RETURNING子句是Greenplum数据库扩展，以及将WITH和DELETE一起使用的功能。

另见

[DECLARE](#), [TRUNCATE](#)

Parent topic: [SQL Command Reference](#)

放弃会话状态。

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)

[ABORT](#)

[ALTER AGGREGATE](#)

[ALTER COLLATION](#)

[ALTER CONVERSION](#)

[ALTER DATABASE](#)

[ALTER DEFAULT PRIVILEGES](#)

[ALTER DOMAIN](#)

[ALTER EXTENSION](#)

[ALTER EXTERNAL TABLE](#)

[ALTER FOREIGN DATA WRAPPER](#)

[ALTER FOREIGN TABLE](#)

[ALTER FUNCTION](#)

[ALTER GROUP](#)

[ALTER INDEX](#)

[ALTER LANGUAGE](#)

[ALTER OPERATOR](#)

概要

```
DISCARD { ALL | PLANS | TEMPORARY | TEMP }
```

描述

DISCARD释放与数据库会话关联的内部资源。此命令对于部分或全部重置会话状态很有用。有几个子命令可以释放不同类型的资源。DISCARD ALL变体包含所有其他变体，并且还重置其他状态。

参数

PLANS

释放所有缓存的查询计划，并在下一次使用关联的预编译语句时强制进行重新计划。

SEQUENCES

丢弃所有与缓存序列相关的状态，包括尚未由nextval()返回的任何预分配序列值。（有关预分配序列值的描述，请参见CREATE SEQUENCE。）

TEMPORARY/TEMP

删除在当前会话中创建的所有临时表。

ALL

释放与当前会话关联的所有临时资源，并将会话重置为其初始状态。当前，这与执行以下语句序列具有相同的效果：

```
SET SESSION AUTHORIZATION DEFAULT;
RESET ALL;
DEALLOCATE ALL;
CLOSE ALL;
UNLISTEN *;
SELECT pg_advisory_unlock_all();
DISCARD PLANS;
DISCARD SEQUENCES;
DISCARD TEMP;
```



注解

DISCARD ALL不能在事务块内执行。

兼容性

DISCARD是Greenplum数据库扩展。

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

执行匿名代码块作为临时匿名函数。

概要

```
DO [ LANGUAGE lang_name ] code
```

描述

DO执行过程代码中的一个匿名代码块，或一个临时的匿名函数。

将代码块视为没有参数的函数的主体，并返回void。它被解析并执行一次。

可选的LANGUAGE子句可以出现在代码块之前或之后。

匿名块是过程语言结构，它提供了动态创建和执行过程代码的能力，而无需将代码作为数据库对象持久存储在系统目录中。匿名块的概念类似于UNIX Shell脚本，它使几个手动输入的命令可以被分组并作为一个步骤执行。顾名思义，匿名块没有名称，因此，不能从其他对象中引用它们。尽管匿名块是动态构建的，但可以轻松地将它们作为脚本存储在操作系统文件中以重复执行。

匿名块是标准的过程语言块。它们带有语法并遵守适用于过程语言的规则，包括变量的声明和范围，执行，异常处理和语言使用。

匿名块的编译和执行在一个步骤中组合在一起，而每次定义更改时，都必须在使用前重新定义用户定义的函数。

参数

code

要执行的过程语言代码。必须将其指定为字符串文字，就像使用CREATE FUNCTION命令一样。建议使用美元符\$将代码括起来。可选关键字无效。支持以下过程语言：PL/pgSQL (plpgsql)，PL/Python (plpythonu) 和PL/Perl (plperl和plperlu)。

lang_name

编写代码的过程语言的名称。默认值为plpgsql。该语言必须安装在Greenplum数据库系统上并在数据库中注册。

注解

PL/pgSQL语言已安装在Greenplum数据库系统上，并已在用户创建的数据库中注册。PL/Python和PL/Perl语言是默认安装的，但未注册。未安装或注册其他语言。系统目录pg_language包含有关数据库中已注册语言的信息。

用户必须对过程语言具有USAGE特权，或者如果该语言不受信任，则必须是超级用户。这与使用该语言创建函数需要的特权相同。

匿名块不支持函数易变性或EXECUTE ON属性。

示例

此PL/pgSQL示例向角色webuser授予对模式public的所有视图的所有特权：

```
DO $$DECLARE r record;
BEGIN
    FOR r IN SELECT table_schema, table_name FROM
information_schema.tables
        WHERE table_type = 'VIEW' AND table_schema =
'public'
    LOOP
        EXECUTE 'GRANT ALL ON ' ||
quote_ident(r.table_schema) || '.' ||
quote_ident(r.table_name) || ' TO webuser';
    END LOOP;
END$$;
```

该PL/pgSQL示例确定Greenplum数据库用户是否为超级用户。在该示例中，匿名块从临时表中检索输入值。

```
CREATE TEMP TABLE list AS VALUES ('gpadmin') DISTRIBUTED
RANDOMLY;

DO $$
DECLARE
    name TEXT := 'gpadmin' ;
    superuser TEXT := '' ;
    t1_row pg_authid%ROWTYPE;
```

```
BEGIN
    SELECT * INTO t1_row FROM pg_authid, list
        WHERE pg_authid.rolname = name ;
    IF t1_row.rolsuper = 'f' THEN
        superuser := 'not ' ;
    END IF ;
    RAISE NOTICE 'user % is %a superuser', t1_row.rolname,
superuser ;
END $$ LANGUAGE plpgsql ;
```

Note: 示例PL/pgSQL将SELECT与INTO子句一起使用。 它与SQL命令SELECT INTO不同。

兼容性

SQL标准中没有DO语句。

另见

CREATE LANGUAGE [Greenplum PL/pgSQL过程语言](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除聚合函数。

概要

```
DROP AGGREGATE [ IF EXISTS ] name ( type [ , ... ] ) [ CASCADE | RESTRICT ]
```

描述

`DROP AGGREGATE`将删除现有的聚合函数。要执行此命令，当前用户必须是聚合函数的所有者。

参数

IF EXISTS

如果聚合不存在，请不要引发错误。在这种情况下会发出通知。*name*

name 现有聚合函数的名称（可以由模式限定）。

type

聚合函数所基于的输入数据类型。要引用零参数聚合函数，请写*代替输入数据类型列表。

CASCADE

自动删除依赖于聚合函数的对象。

RESTRICT

如果有任何对象依赖聚合函数，则拒绝删除该函数。这是默认值。

示例

要删除整数类型的聚合函数`myavg`：

```
DROP AGGREGATE myavg(integer);
```



兼容性

SQL标准中没有DROP AGGREGATE语句。

另见

[ALTER AGGREGATE](#), [CREATE AGGREGATE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)[ABORT](#)[ALTER AGGREGATE](#)[ALTER COLLATION](#)[ALTER CONVERSION](#)[ALTER DATABASE](#)[ALTER DEFAULT PRIVILEGES](#)[ALTER DOMAIN](#)[ALTER EXTENSION](#)[ALTER EXTERNAL TABLE](#)[ALTER FOREIGN DATA WRAPPER](#)[ALTER FOREIGN TABLE](#)[ALTER FUNCTION](#)[ALTER GROUP](#)[ALTER INDEX](#)[ALTER LANGUAGE](#)[ALTER OPERATOR](#)

删除一个造型。

概要

```
DROP CAST [IF EXISTS] (sourcetype AS targettype) [CASCADE | RESTRICT]
```

描述

`DROP CAST`将删除先前定义的造型。为了能够删除造型，您必须拥有源或目标数据类型。这些与创建造型所需的特权相同。

参数

IF EXISTS

如果造型不存在，请不要抛出错误。在这种情况下会发出通知。
sourcetype

造型的源数据类型的名称。

targettype

造型的目标数据类型的名称。

CASCADE

RESTRICT

这些关键字没有任何作用，因为不依赖造型。

示例

删除从`text`类型到`int`类型的造型：

```
DROP CAST (text AS int);
```



兼容性

DROP CAST命令符合SQL标准。

另见

[CREATE CAST](#)

Parent topic: [SQL Command Reference](#)

删除以前定义的排序规则。

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

[SQL Syntax Summary](#)

[ABORT](#)

[ALTER AGGREGATE](#)

[ALTER COLLATION](#)

[ALTER CONVERSION](#)

[ALTER DATABASE](#)

[ALTER DEFAULT PRIVILEGES](#)

[ALTER DOMAIN](#)

[ALTER EXTENSION](#)

[ALTER EXTERNAL TABLE](#)

[ALTER FOREIGN DATA WRAPPER](#)

[ALTER FOREIGN TABLE](#)

[ALTER FUNCTION](#)

[ALTER GROUP](#)

[ALTER INDEX](#)

[ALTER LANGUAGE](#)

[ALTER OPERATOR](#)

概要

```
DROP COLLATION [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

参数

IF EXISTS

如果排序规则不存在，请不要抛出错误。在这种情况下会发出通知。

name

排序规则的名称。排序规则名称可以是模式限定的。

CASCADE

自动删除依赖于排序规则的对象。

RESTRICT

如果有任何对象依赖该排序规则，则拒绝删除该排序规则。这是默认值。

注解

`DROP COLLATION`删除以前定义的排序规则。要删除排序规则，您必须拥有该排序规则。

示例

删除`german`的排序规则：

```
DROP COLLATION german;
```



兼容性

除了IF EXISTS选项（这是Greenplum数据库扩展名）之外，DROP COLLATION命令符合SQL标准。

另见

[ALTER COLLATION, CREATE COLLATION](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

删除转换。

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

概要

DROP CONVERSION [IF EXISTS] *name* [CASCADE | RESTRICT]

描述

DROP CONVERSION删除以前定义的转换。为了放弃转换，您必须拥有该转换。

参数

IF EXISTS

如果转换不存在，请不要抛出错误。在这种情况下会发出通知。*name*

转换的名称。转换名称可以是模式限定的。

CASCADE

RESTRICT

这些关键字无效，因为它们不依赖转换。

示例

删除名为*myname*的转换：

DROP CONVERSION *myname*;

兼容性

SQL标准中没有DROP CONVERSION语句。该标准具有CREATE TRANSLATION和DROP TRANSLATION语句，与Greenplum数据库



的CREATE CONVERSION和DROP CONVERSION语句相似。

另见

[ALTER CONVERSION](#), [CREATE CONVERSION](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除数据库。

概要

DROP DATABASE [IF EXISTS] *name*

描述

DROP DATABASE删除数据库。它删除数据库的catalog条目，并删除包含数据的目录。它只能由数据库所有者执行。另外，当您或其他任何人连接到目标数据库时，也无法执行该命令。（连接到postgres或任何其他数据库以发出此命令。）

Warning: DROP DATABASE无法撤消。小心使用！

参数

IF EXISTS

如果数据库不存在，请不要报错。在这种情况下会发出通知。*name*

要删除的数据库的名称。

注解

无法在事务块内部执行DROP DATABASE。

连接到目标数据库时无法执行此命令。因此，改为使用程序dropdb可能更方便，该程序是该命令的包装器。

示例

删除名为testdb的数据库：



```
DROP DATABASE testdb;
```

兼容性

SQL标准中没有DROP DATABASE语句。

另见

[ALTER DATABASE](#) , [CREATE DATABASE](#)

Parent topic: [SQL Command Reference](#)

删除域。

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)

[ABORT](#)

[ALTER AGGREGATE](#)

[ALTER COLLATION](#)

[ALTER CONVERSION](#)

[ALTER DATABASE](#)

[ALTER DEFAULT PRIVILEGES](#)

[ALTER DOMAIN](#)

[ALTER EXTENSION](#)

[ALTER EXTERNAL TABLE](#)

[ALTER FOREIGN DATA WRAPPER](#)

[ALTER FOREIGN TABLE](#)

[ALTER FUNCTION](#)

[ALTER GROUP](#)

[ALTER INDEX](#)

[ALTER LANGUAGE](#)

[ALTER OPERATOR](#)

概要

```
DROP DOMAIN [ IF EXISTS ] name [ , ... ] [ CASCADE | RESTRICT ]
```

描述

`DROP DOMAIN`删除先前定义的域。 您必须是域的所有者才能删除它。

参数

IF EXISTS

如果域不存在，请不要报错。在这种情况下会发出通知。

name

现有域的名称（可以用模式指定）。

CASCADE

自动删除依赖域的对象（例如表列）。

RESTRICT

如果有任何对象依赖该域，则拒绝删除该域。这是默认值。

示例

删除名为`zipcode`的域：

```
DROP DOMAIN zipcode;
```

兼容性

此命令符合SQL标准，但`IF EXISTS`选项是Greenplum数据库扩展。



另见

[ALTER DOMAIN](#) , [CREATE DOMAIN](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

从Greenplum数据库中删除扩展。

概要

```
DROP EXTENSION [ IF EXISTS ] name [ , ... ] [ CASCADE | RESTRICT ]
```

描述

`DROP EXTENSION`从数据库中删除扩展。 删除扩展会导致其组成对象也被删除。

Note: 创建扩展所需的支持扩展文件没有被删除。 必须从Greenplum数据库主机中手动删除文件。

您必须拥有该扩展才能使用`DROP EXTENSION`。

如果数据库中正在使用任何扩展的对象，则此命令将失败。 例如，如果用扩展类型的列定义了表。 添加`CASCADE`选项以强制删除那些依赖对象。

Important: 在使用`CASCADE`关键字发出`DROP EXTENSION`之前， 您应该了解所有依赖于扩展名的对象，以避免意外的结果。

参数

IF EXISTS

如果扩展名不存在，请不要报错。 发出通知。

name

已安装扩展的名称。

CASCADE

自动删除依赖于扩展的对象，并删除依赖于这些对象的所有对



象。请参阅有关[依赖跟踪](#)的PostgreSQL信息。

RESTRICT

如果扩展成员对象以外的任何对象依赖扩展，则拒绝删除该扩展。这是默认值。

兼容性

DROP EXTENSION是Greenplum数据库扩展。

另见

[CREATE EXTENSION](#), [ALTER EXTENSION](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除外部表定义。

概要

```
DROP EXTERNAL [WEB] TABLE [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

描述

`DROP EXTERNAL TABLE`从数据库系统中删除现有的外部表定义。外部数据源或文件不会被删除。要执行此命令，您必须是外部表的所有者。

参数

WEB

用于删除外部Web表的可选关键字。

IF EXISTS

如果外部表不存在，不要报错。在这种情况下会发出通知。

name

现有外部表的名称（可以由模式指定）。

CASCADE

自动删除依赖于外部表的对象（例如视图）。

RESTRICT

如果有任何对象依赖外部表，则拒绝删除它。这是默认值。

示例

删除名为`staging`的外部表（如果存在）：

```
DROP EXTERNAL TABLE IF EXISTS staging;
```



兼容性

SQL标准中没有DROP EXTERNAL TABLE语句。

另见

[CREATE EXTERNAL TABLE](#)

Parent topic: [SQL Command Reference](#)

WRAPPER

Greenplum数据库® 6.0文档

删除外部数据包装器。

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLE

ALTER FOREIGN
DATA WRAPPER

ALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

概要

```
DROP FOREIGN DATA WRAPPER [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

描述

DROP FOREIGN DATA WRAPPER从当前数据库中删除现有的外部数据包装器。外部数据包装程序只能由其所有者删除。

参数

IF EXISTS

如果外部数据包装器不存在，不要报错。在这种情况下，Greenplum数据库会发出通知。

name

现有外部数据包装器的名称。

CASCADE

自动删除依赖于外部数据包装器的对象（例如服务器）。

RESTRICT

如果有任何对象依赖于它，则拒绝删除外部数据包装器。这是默认值。



示例

删除名为dbi的外部数据包装器：

```
DROP FOREIGN DATA WRAPPER dbi;
```

兼容性

DROP FOREIGN DATA WRAPPER符合ISO/IEC 9075-9 (SQL/MED)。IF EXISTS子句是Greenplum数据库扩展。

另见

[CREATE FOREIGN DATA WRAPPER](#), [ALTER FOREIGN DATA WRAPPER](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除外部表。

概要

```
DROP FOREIGN TABLE [ IF EXISTS ] name [ , ... ] [ CASCADE | RESTRICT ]
```

描述

`DROP FOREIGN TABLE`删除现有的外部表。只有外部表的所有者才能将其删除。

参数

IF EXISTS

如果外部表不存在，不要报错。在这种情况下，Greenplum数据库会发出通知。

name

要删除的外部表的名称（可以由模式指定）。

CASCADE

自动删除依赖于外部表的对象（例如视图）。

RESTRICT

如果有任何对象依赖外部表，则拒绝删除该表。这是默认值。

示例

删除名为`films`和`distributors`的外部表：



```
DROP FOREIGN TABLE films, distributors;
```

兼容性

DROP FOREIGN TABLE 符合 ISO/IEC 9075-9 (SQL/MED) , 但标准仅允许每个命令删除一个外部表。 IF EXISTS 子句是 Greenplum 数据库扩展。

另见

[ALTER FOREIGN TABLE](#), [CREATE FOREIGN TABLE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除函数。

概要

```
DROP FUNCTION [IF EXISTS] name ( [ [argmode] [argname]
argtype
[ , ... ] ] ) [CASCADE | RESTRICT]
```

描述

`DROP FUNCTION`删除现有函数的定义。要执行此命令，用户必须是该函数的所有者。必须指定函数的参数类型，因为可能存在多个具有相同名称和不同参数列表的不同函数。

参数

IF EXISTS

如果该函数不存在，请不要报错。在这种情况下会发出通知。

name

现有函数的名称（可以由模式指定）。

*argmode*参数的模式：`IN`, `OUT`, `INOUT`或`VARIADIC`。如果省略，则默认值为`IN`。请注意，由于仅需要输入参数来确定函数的标识，因此`DROP FUNCTION`实际上并不关注`OUT`参数。因此，列出`IN`, `INOUT`和`VARIADIC`参数就足够了。*argname*参数的名称。请注意，由于仅需要参数数据类型来确定函数的标识，因此`DROP FUNCTION`实际上并不关注参数名称。*argtype*

函数参数的数据类型（可以由模式指定）（如果有）。

CASCADE

自动删除依赖于函数的对象，例如运算符。

RESTRICT

如果有任何对象依赖该函数，则拒绝删除该函数。这是默认值。



示例

删除平方根函数：

```
DROP FUNCTION sqrt(integer);
```

兼容性

SQL标准中定义了DROP FUNCTION语句，但该命令与此命令不兼容。

另见

[CREATE FUNCTION](#) , [ALTER FUNCTION](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0 文档

□ 参考指南

□ SQL Command Reference

[SQL Syntax Summary](#)[ABORT](#)[ALTER AGGREGATE](#)[ALTER COLLATION](#)[ALTER CONVERSION](#)[ALTER DATABASE](#)[ALTER DEFAULT PRIVILEGES](#)[ALTER DOMAIN](#)[ALTER EXTENSION](#)[ALTER EXTERNAL TABLE](#)[ALTER FOREIGN DATA WRAPPER](#)[ALTER FOREIGN TABLE](#)[ALTER FUNCTION](#)[ALTER GROUP](#)[ALTER INDEX](#)[ALTER LANGUAGE](#)[ALTER OPERATOR](#)

删除数据库角色。

概要

```
DROP GROUP [ IF EXISTS ] name [ , ... ]
```

描述

DROP GROUP是DROP ROLE的别名。有关更多信息，请参见[DROP ROLE](#)。

兼容性

SQL标准中没有DROP GROUP语句。

另见

[DROP ROLE](#)

Parent topic: [SQL Command Reference](#)



Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除索引。

概要

```
DROP INDEX [ CONCURRENTLY ] [ IF EXISTS ] name [ , ... ] [ CASCADE | RESTRICT ]
```

描述

`DROP INDEX`从数据库系统删除现有索引。要执行此命令，您必须是索引的所有者。

参数

CONCURRENTLY

删除索引而不锁定索引表上的并发选择、插入、更新和删除。普通的`DROP INDEX`在表上获取排他锁，从而阻止其他访问，直到可以完成索引删除为止。使用此选项，命令将一直等到冲突的事务完成。

使用此选项时需要注意几个注意事项。只能指定一个索引名称，并且不支持`CASCADE`选项。（因此，不能以这种方式删除支持`UNIQUE`或`PRIMARY KEY`约束的索引。）而且，可以在事务块内执行常规的`DROP INDEX`命令，但不能以`DROP INDEX CONCURRENTLY`方式执行。

IF EXISTS

如果该索引不存在，请不要报错。在这种情况下会发出通知。

name

现有索引的名称（可以由模式指定）。

CASCADE

自动删除依赖于索引的对象。

RESTRICT

如果有任何对象依赖该索引，则拒绝删除该索引。

默认值。

示例

删除索引|title_idx:

```
DROP INDEX title_idx;
```

兼容性

DROP INDEX是Greenplum数据库语言的扩展。 SQL标准中没有索引的规定。

另见

[ALTER INDEX](#) , [CREATE INDEX](#) , [REINDEX](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除过程语言。

概要

```
DROP [ PROCEDURAL ] LANGUAGE [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

描述

`DROP LANGUAGE`将删除先前注册的过程语言的定义。您必须是超级用户或该语言的所有者才能删除该语言。

参数

PROCEDURAL

可选关键字-无效。

IF EXISTS

如果该语言不存在，请不要报错。在这种情况下会发出通知。

name

现有过程语言的名称。为了向后兼容，名称可以用单引号引起。

CASCADE

自动删除依赖于该语言的对象（例如用该语言编写的函数）。

RESTRICT

如果有任何对象依赖该语言，则拒绝删除该语言。这是默认值。

示例

删除过程语言`plsample`:

```
DROP LANGUAGE plsample;
```



兼容性

SQL标准中没有DROP LANGUAGE语句。

另见

[ALTER LANGUAGE](#) , [CREATE LANGUAGE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除运算符。

概要

```
DROP OPERATOR [ IF EXISTS ] name ( { lefttype | NONE } ,
{ righttype | NONE } ) [ CASCADE | RESTRICT ]
```

描述

`DROP OPERATOR`从数据库系统中删除现有的运算符。要执行此命令，您必须是运算符的所有者。

参数

IF EXISTS

如果该运算符不存在，请不要报错。在这种情况下会发出通知。*name*

lefttype

运算符的左操作数的数据类型；如果运算符没有左操作数，则写NONE。

righttype

运算符的右操作数的数据类型；如果运算符没有右操作数，则写NONE。

CASCADE

自动删除依赖于运算符的对象。

RESTRICT

如果有任何对象依赖于该运算符，请拒绝删除该运算符。这是默认值。

示例

删除integer类型的幂运算符 a^b ：



```
DROP OPERATOR ^ (integer, integer);
```

删除类型bit的左一元按位补数运算符~b:

```
DROP OPERATOR ~ (none, bit);
```

删除bigint类型的右一元阶乘运算符x!:

```
DROP OPERATOR ! (bigint, none);
```

兼容性

SQL标准中没有DROP OPERATOR语句。

另见

[ALTER OPERATOR](#) , [CREATE OPERATOR](#)

Parent topic: [SQL Command Reference](#)

删除运算符类。

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLE

ALTER FOREIGN
DATA WRAPPER

ALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

概要

```
DROP OPERATOR CLASS [ IF EXISTS ] name USING index_method  
[ CASCADE | RESTRICT ]
```

描述

`DROP OPERATOR`删除现有的运算符类。要执行此命令，您必须是运算符类的所有者。

参数

IF EXISTS

如果该运算符类不存在，请不要报错。在这种情况下会发出通知。

name

现有运算符类的名称（可以由模式指定）。

index_method

运算符类用于的索引访问方法的名称。

CASCADE

自动删除依赖于运算符类的对象。

RESTRICT

如果有任何对象依赖于该运算符类，则拒绝删除该类。这是默认值。

示例

删除B-树运算符类`widget_ops`:

```
DROP OPERATOR CLASS widget_ops USING btree;
```



如果存在使用运算符类的索引，则此命令将不会成功。添加CASCADE以删除此类索引以及运算符类。

兼容性

SQL标准中没有DROP OPERATOR CLASS语句。

另见

[ALTER OPERATOR CLASS](#) , [CREATE OPERATOR CLASS](#)

Parent topic: [SQL Command Reference](#)

FAMILY

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLE

ALTER FOREIGN
DATA WRAPPER

ALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除一个运算符族

概要

```
DROP OPERATOR FAMILY [ IF EXISTS ] name USING index_method
[ CASCADE | RESTRICT ]
```

描述

DROP OPERATOR FAMILY删除现有的运算符族。要执行此命令，您必须是运算符族的所有者。

DROP OPERATOR FAMILY包括删除该族中包含的任何运算符类，但不会删除该族引用的任何运算符或函数。如果有任何索引取决于该族中的运算符类，则需要指定CASCADE才能完成删除。

参数

IF EXISTS

如果该运算符族不存在，请不要报错。在这种情况下会发出通知。

name

现有运算符族的名称（可以由模式指定）。

index_method

运算符族用于的索引访问方法的名称。

CASCADE

自动删除依赖于运算符族的对象。

RESTRICT

如果有任何对象依赖它，请拒绝删除该运算符族。这是默认值。



示例

删除B-树运算符族float_ops:

```
DROP OPERATOR FAMILY float_ops USING btree;
```

如果存在任何使用运算符族的索引，则此命令将不会成功。添加CASCADE可以删除此类索引以及运算符族。

兼容性

SQL标准中没有DROP OPERATOR FAMILY语句。

另见

[ALTER OPERATOR FAMILY](#) , [CREATE OPERATOR FAMILY](#) ,
[ALTER OPERATOR CLASS](#) , [CREATE OPERATOR CLASS](#) , [DROP OPERATOR CLASS](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除数据库角色拥有的数据库对象。

概要

```
DROP OWNED BY name [ , ... ] [ CASCADE | RESTRICT ]
```

描述

`DROP OWNED`删除当前数据库中由指定角色之一拥有的所有对象。在当前数据库中的对象上授予给定角色的任何特权也将被撤销。

参数

name

一个角色的名称，该角色的对象将被删除，其特权将被撤销。
CASCADE

自动删除依赖于受影响对象的对象。

RESTRICT

如果任何其他数据库对象依赖于受影响的对象之一，则拒绝删除角色拥有的对象。这是默认值。

注解

`DROP OWNED`通常用于准备删除一个或多个角色。由于`DROP OWNED`仅影响当前数据库中的对象，因此通常需要在每个数据库中执行此命令，该数据库包含要删除的角色拥有的对象。

使用`CASCADE`选项可能会使命令递归到其他用户拥有的对象。

`REASSIGN OWNED`命令是一种替代方法，它重新分配一个或多个角色拥有的所有数据库对象的所有权。但是，`REASSIGN OWNED`不处理其他对象的特权。

示例

删除名称为sally的角色拥有的所有数据库对象：

```
DROP OWNED BY sally;
```

兼容性

DROP OWNED命令是Greenplum数据库扩展。

另见

[REASSIGN OWNED](#) , [DROP ROLE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

从数据库中删除外部表数据访问协议。

概要

DROP PROTOCOL [IF EXISTS] *name*

描述

DROP PROTOCOL从数据库中删除指定的协议。可以在CREATE EXTERNAL TABLE命令中指定协议名称，以从外部数据源读取数据或将数据写入外部数据源。

您必须是超级用户或协议所有者才能删除协议。

Warning: 如果删除数据访问协议，则使用该协议定义的外部表将不再能够访问外部数据源。

参数

IF EXISTS

如果该协议不存在，请不要报错。在这种情况下会发出通知。
name

现有数据访问协议的名称。

注解

如果删除数据访问协议，则不会删除数据库中与该协议相关联的调用处理程序。您必须手动删除函数。

协议使用的共享库也应从Greenplum数据库主机中删除。



兼容性

DROP PROTOCOL是Greenplum数据库扩展。

另见

[CREATE EXTERNAL TABLE](#), [CREATE PROTOCOL](#)

Parent topic: [SQL Command Reference](#)

GROUP

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除资源组。

概要

```
DROP RESOURCE GROUP group_name
```

描述

此命令从Greenplum数据库中删除资源组。只有超级用户才能删除资源组。删除资源组时，该组保留的内存和CPU资源将回收到Greenplum数据库。

要删除角色资源组，不能将该组分配给任何角色，也不能在该组中有任何待处理或正在运行的语句。如果删除为外部组件创建的资源组，则行为由外部组件决定。例如，删除分配给PL/Container运行时的资源组会杀死该组中正在运行的容器。

您不能删除预定义的admin_group和default_group资源组。

参数

group_name

要删除的资源组的名称。

注解

您不能在显式事务或子事务中提交DROP RESOURCE GROUP命令。

使用[ALTER ROLE](#)删除分配给特定用户/角色的资源组。



执行以下查询以查看所有资源组的所有当前活动查询：

```
SELECT username, current_query, waiting, procpid,  
       rsgid, rsgname, rsgqueueduration  
  FROM pg_stat_activity;
```

要查看资源组分配，请对pg_roles和pg_resgroup系统catalog表执行以下查询：

```
SELECT rolname, rsgname  
  FROM pg_roles, pg_resgroup  
 WHERE pg_roles.rolresgroup=pg_resgroup.oid;
```

示例

删除分配给角色的资源组。然后，此操作将默认资源组default_group分配给角色：

```
ALTER ROLE bob RESOURCE GROUP NONE;
```

删除名为adhoc的资源组：

```
DROP RESOURCE GROUP adhoc;
```

兼容性

DROP RESOURCE GROUP语句是Greenplum数据库扩展。

另见

[ALTER RESOURCE GROUP](#) , [CREATE RESOURCE GROUP](#) , [ALTER ROLE](#)

Parent topic: [SQL Command Reference](#)

QUEUE

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLE

ALTER FOREIGN
DATA WRAPPER

ALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除资源队列。

概要

```
DROP RESOURCE QUEUE queue_name
```

描述

此命令从Greenplum数据库中删除资源队列。要删除资源队列，队列不能分配任何角色，队列中也不能有任何等待语句。只有超级用户才能删除资源队列。

参数

queue_name

要删除的资源队列的名称。

注解

使用[ALTER ROLE](#)从资源队列中删除用户。

要查看所有资源队列的所有当前活动查询，请对与pg_roles和pg_resqueue表连接pg_locks表执行以下查询：

```
SELECT rolname, rsqname, locktype, objid, pid,
mode, granted FROM pg_roles, pg_resqueue, pg_locks WHERE
pg_roles.rolresqueue=pg_locks.objid AND
pg_locks.objid=pg_resqueue.oid;
```

要查看分配给资源队列的角色，请对pg_roles和pg_resqueue系统catalog表执行以下查询：



```
SELECT rolname, rsqname FROM pg_roles, pg_resqueue WHERE pg_roles.rolresqueue=pg_resqueue.oid;
```

示例

从资源队列中删除角色（并将角色移至默认资源队列pg_default）：

```
ALTER ROLE bob RESOURCE QUEUE NONE;
```

删除名为adhoc的资源队列：

```
DROP RESOURCE QUEUE adhoc;
```

兼容性

DROP RESOURCE QUEUE语句是Greenplum数据库扩展。

另见

[ALTER RESOURCE QUEUE](#) , [CREATE RESOURCE QUEUE](#) , [ALTER ROLE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除数据库角色。

概要

```
DROP ROLE [ IF EXISTS ] name [ , ... ]
```

描述

`DROP ROLE`删除指定的角色。要删除超级用户角色，您必须自己是超级用户。要删除非超级用户角色，您必须具有`CREATEROLE`特权。

如果仍在任何数据库中引用角色，则无法删除该角色；如果是这样，将引发错误。在删除角色之前，必须删除其拥有的所有对象（或重新分配其所有权），并撤消已授予该角色在其他对象上的所有特权。`REASSIGN OWNED`和`DROP OWNED`命令可用于此目的。

但是，不必删除涉及该角色的角色成员资格；`DROP ROLE`自动撤消其他角色中的目标角色以及目标角色中的其他角色的所有成员身份。其他角色不会丢失，也不会受到其他影响。

参数

IF EXISTS

如果该角色不存在，请不要报错。在这种情况下会发出通知。`name`

要删除的角色名称。

示例

删除名为`sally`和`bob`的角色：

```
DROP ROLE sally, bob;
```



兼容性

SQL标准定义了`DROP ROLE`, 但一次只允许删除一个角色, 并且它指定的特权要求与Greenplum数据库使用的特权要求不同。

另见

[REASSIGN OWNED](#) , [DROP OWNED](#) , [CREATE ROLE](#) , [ALTER ROLE](#) , [SET ROLE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除重写规则。

概要

```
DROP RULE [ IF EXISTS ] name ON table_name [ CASCADE | RESTRICT ]
```

描述

`DROP RULE`删除表或视图中的重写规则。

参数

IF EXISTS

如果该规则不存在，请不要报错。在这种情况下会发出通知。

name

要删除的规则的名称。

table_name

规则适用的表或视图的名称（可以由模式指定）。

CASCADE

自动删除依赖于规则的对象。

RESTRICT

如果有任何对象依赖该规则，则拒绝删除该规则。这是默认值。

示例

删除表`sales`上的重写规则`sales_2006`:

```
DROP RULE sales_2006 ON sales;
```



兼容性

DROP RULE是Greenplum数据库语言的扩展，整个查询重写系统也是如此。

另见

[ALTER RULE](#) , [CREATE RULE](#)

Parent topic: [SQL Command Reference](#)

删除模式。

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)

[ABORT](#)

[ALTER AGGREGATE](#)

[ALTER COLLATION](#)

[ALTER CONVERSION](#)

[ALTER DATABASE](#)

[ALTER DEFAULT PRIVILEGES](#)

[ALTER DOMAIN](#)

[ALTER EXTENSION](#)

[ALTER EXTERNAL TABLE](#)

[ALTER FOREIGN DATA WRAPPER](#)

[ALTER FOREIGN TABLE](#)

[ALTER FUNCTION](#)

[ALTER GROUP](#)

[ALTER INDEX](#)

[ALTER LANGUAGE](#)

[ALTER OPERATOR](#)

概要

```
DROP SCHEMA [ IF EXISTS ] name [ , ... ] [ CASCADE | RESTRICT ]
```

描述

`DROP SCHEMA`从数据库中删除模式。 模式只能由其所有者或超级用户删除。 请注意，即使所有者不拥有该模式内的某些对象，所有者也可以删除该模式（从而删除所有包含的对象）。

参数

IF EXISTS

如果该模式不存在，请不要报错。在这种情况下会发出通知。
name

要删除的模式的名称。

CASCADE

自动删除模式中包含的所有对象（表，函数等）。

RESTRICT

如果模式包含任何对象，则拒绝删除该模式。这是默认值。

示例

从数据库中删除模式`mystuff`及其包含的所有内容：

```
DROP SCHEMA mystuff CASCADE ;
```



兼容性

DROP SCHEMA完全符合SQL标准，但该标准仅允许每个命令删除一个模式。另外，IF EXISTS选项是Greenplum数据库扩展。

另见

[CREATE SCHEMA](#) , [ALTER SCHEMA](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

删除序列。

参考指南

SQL Command Reference

[SQL Syntax Summary](#)

[ABORT](#)

[ALTER AGGREGATE](#)

[ALTER COLLATION](#)

[ALTER CONVERSION](#)

[ALTER DATABASE](#)

[ALTER DEFAULT PRIVILEGES](#)

[ALTER DOMAIN](#)

[ALTER EXTENSION](#)

[ALTER EXTERNAL TABLE](#)

[ALTER FOREIGN DATA WRAPPER](#)

[ALTER FOREIGN TABLE](#)

[ALTER FUNCTION](#)

[ALTER GROUP](#)

[ALTER INDEX](#)

[ALTER LANGUAGE](#)

[ALTER OPERATOR](#)

概要

```
DROP SEQUENCE [ IF EXISTS ] name [ , ... ] [ CASCADE | RESTRICT ]
```

描述

`DROP SEQUENCE`删除序列生成器表。您必须拥有删除的序列（或成为超级用户）。

参数

IF EXISTS

如果该序列不存在，请不要报错。在这种情况下会发出通知。

name

要删除的序列的名称（可以由模式指定）。

CASCADE

自动删除依赖于序列的对象。

RESTRICT

如果有任何对象依赖该序列，则拒绝删除该序列。这是默认值。

示例

删除序列`myserial`:

```
DROP SEQUENCE myserial;
```



兼容性

DROP SEQUENCE完全符合SQL标准，但该标准仅允许每个命令删除一个序列。另外，IF EXISTS选项是Greenplum数据库扩展。

另见

[ALTER SEQUENCE](#) , [CREATE SEQUENCE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)[ABORT](#)[ALTER AGGREGATE](#)[ALTER COLLATION](#)[ALTER CONVERSION](#)[ALTER DATABASE](#)[ALTER DEFAULT PRIVILEGES](#)[ALTER DOMAIN](#)[ALTER EXTENSION](#)[ALTER EXTERNAL TABLE](#)[ALTER FOREIGN DATA WRAPPER](#)[ALTER FOREIGN TABLE](#)[ALTER FUNCTION](#)[ALTER GROUP](#)[ALTER INDEX](#)[ALTER LANGUAGE](#)[ALTER OPERATOR](#)

删除外部服务器描述符。

概要

```
DROP SERVER [ IF EXISTS ] servername [ CASCADE | RESTRICT ]
```

描述

`DROP SERVER`删除现有的外部服务器描述符。执行此命令的用户必须是服务器的所有者。

参数

IF EXISTS

如果该服务不存在，请不要报错。在这种情况下会发出通知。

servername

现有服务器的名称。

CASCADE

自动删除依赖于服务器的对象（例如用户映射）。

RESTRICT

如果有任何对象依赖于该服务器，则拒绝删除该服务器。这是默认值。

示例

删除名为`foo`的服务器（如果存在）：

```
DROP SERVER IF EXISTS foo;
```



兼容性

DROP SERVER符合ISO/IEC 9075-9 (SQL/MED)。IF EXISTS子句是Greenplum数据库扩展。

另见

[CREATE SERVER, ALTER SERVER](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除表。

概要

DROP TABLE [IF EXISTS] *name* [, ...] [CASCADE | RESTRICT]

描述

DROP TABLE从数据库中删除表。只有表所有者，模式所有者和超级用户可以删除表。要清空行表而不删除表定义，请使用DELETE或TRUNCATE。

DROP TABLE始终删除目标表存在的所有索引，规则，触发器和约束。但是，要删除视图引用的表，必须指定CASCADE。CASCADE将完全删除从属视图。

参数

IF EXISTS

如果该表不存在，请不要报错。在这种情况下会发出通知。*name*

要删除的表的名称（可以由模式指定）。

CASCADE

自动删除依赖于表的对象（例如视图）。

RESTRICT

如果有任何对象依赖表，则拒绝删除该表。这是默认值。

示例

删除表mytable:

DROP TABLE mytable;



兼容性

DROP TABLE完全符合SQL标准，但该标准仅允许每个命令删除一个表。另外，IF EXISTS选项是Greenplum数据库扩展。

另见

[CREATE TABLE](#) , [ALTER TABLE](#) , [TRUNCATE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)[ABORT](#)[ALTER AGGREGATE](#)[ALTER COLLATION](#)[ALTER CONVERSION](#)[ALTER DATABASE](#)[ALTER DEFAULT PRIVILEGES](#)[ALTER DOMAIN](#)[ALTER EXTENSION](#)[ALTER EXTERNAL TABLE](#)[ALTER FOREIGN DATA WRAPPER](#)[ALTER FOREIGN TABLE](#)[ALTER FUNCTION](#)[ALTER GROUP](#)[ALTER INDEX](#)[ALTER LANGUAGE](#)[ALTER OPERATOR](#)

删除表空间。

概要

```
DROP TABLESPACE [ IF EXISTS ] tablespacename
```

描述

`DROP TABLESPACE`从系统中删除表空间。

表空间只能由其所有者或超级用户删除。在删除表空间之前，表空间中的所有数据库对象必须为空。即使当前数据库中没有对象正在使用表空间，其他数据库中的对象也可能仍驻留在表空间中。

参数

IF EXISTS

如果该表空间不存在，请不要报错。在这种情况下会发出通知。
tablespacename

要删除的表空间的名称。

示例

删除表空间`mystuff`:

```
DROP TABLESPACE mystuff;
```

兼容性

`DROP TABLESPACE`是Greenplum数据库扩展。



另见

[CREATE TABLESPACE](#) , [ALTER TABLESPACE](#)

Parent topic: [SQL Command Reference](#)

CONFIGURATION

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

概要

```
DROP TEXT SEARCH CONFIGURATION [ IF EXISTS ] name [ CASCADE
| RESTRICT ]
```

描述

DROP TEXT SEARCH CONFIGURATION删除已有的文本搜索配置。要执行此命令，您必须是配置的所有者。

参数

IF EXISTS

如果该配置不存在，请不要报错。在这种情况下会发出通知。

name

现有文本搜索配置的名称（可以由模式指定）。

CASCADE

自动删除依赖于文本搜索配置的对象。

RESTRICT

如果有任何对象依赖它，则拒绝删除文本搜索配置。这是默认值。

示例

删除文本搜索配置my_english：

```
DROP TEXT SEARCH CONFIGURATION my_english;
```

如果有任何现有索引引用to_tsvector调用中的配置，则此命令将



不会成功。添加CASCADE以删除此类索引以及文本搜索配置。

兼容性

在SQL标准中没有DROP TEXT SEARCH CONFIGURATION语句。

另见

[ALTER TEXT SEARCH CONFIGURATION](#) , [CREATE TEXT SEARCH CONFIGURATION](#)

Parent topic: [SQL Command Reference](#)

DICTIONARY

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLE

ALTER FOREIGN
DATA WRAPPER

ALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除文本搜索字典。

概要

```
DROP TEXT SEARCH DICTIONARY [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

描述

DROP TEXT SEARCH DICTIONARY删除现有的文本搜索词典。要执行此命令，您必须是字典的所有者。

参数

IF EXISTS

如果该字典不存在，请不要报错。在这种情况下会发出通知。

name

现有文本搜索字典的名称（可以由模式指定）。

CASCADE

自动删除依赖于文本搜索字典的对象。

RESTRICT

如果有任何对象依赖它，则拒绝删除文本搜索字典。这是默认值。

示例

删除文本搜索词典english：

```
DROP TEXT SEARCH DICTIONARY english;
```

如果存在使用字典的任何现有文本搜索配置，则此命令将不会成功。



添加CASCADE可以将此类配置与字典一起删除。

兼容性

SQL标准中没有CREATE TEXT SEARCH DICTIONARY语句。

另见

[ALTER TEXT SEARCH DICTIONARY](#) , [CREATE TEXT SEARCH DICTIONARY](#)

Parent topic: [SQL Command Reference](#)

PARSER

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)

[ABORT](#)

[ALTER AGGREGATE](#)

[ALTER COLLATION](#)

[ALTER CONVERSION](#)

[ALTER DATABASE](#)

[ALTER DEFAULT
PRIVILEGES](#)

[ALTER DOMAIN](#)

[ALTER EXTENSION](#)

[ALTER EXTERNAL
TABLE](#)

[ALTER FOREIGN
DATA WRAPPER](#)

[ALTER FOREIGN
TABLE](#)

[ALTER FUNCTION](#)

[ALTER GROUP](#)

[ALTER INDEX](#)

[ALTER LANGUAGE](#)

[ALTER OPERATOR](#)

描述

删除文本搜索解析器。

概要

```
DROP TEXT SEARCH PARSER [ IF EXISTS ] name [ CASCADE |  
RESTRICT ]
```

描述

`DROP TEXT SEARCH PARSER`删除现有的文本搜索解析器。您必须是超级用户才能使用此命令。

参数

`IF EXISTS`

`name` 如果该解析器不存在，请不要报错。在这种情况下会发出通知。

现有文本搜索解析器的名称（可以由模式指定）。

`CASCADE`

自动删除依赖于文本搜索解析器的对象。

`RESTRICT`

如果有任何对象依赖它，则拒绝删除文本搜索解析器。这是默认值。

示例

删除文本搜索解析器`my_parser`:



```
DROP TEXT SEARCH PARSER my_parser;
```

如果存在使用解析器的任何现有文本搜索配置，则此命令将不会成功。添加CASCADE可以将这些配置与解析器一起删除。

兼容性

在SQL标准中没有DROP TEXT SEARCH PARSER语句。

另见

[ALTER TEXT SEARCH PARSER, CREATE TEXT SEARCH PARSER](#)

Parent topic: [SQL Command Reference](#)

TEMPLATE

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)

[ABORT](#)

[ALTER AGGREGATE](#)

[ALTER COLLATION](#)

[ALTER CONVERSION](#)

[ALTER DATABASE](#)

[ALTER DEFAULT PRIVILEGES](#)

[ALTER DOMAIN](#)

[ALTER EXTENSION](#)

[ALTER EXTERNAL TABLE](#)

[ALTER FOREIGN DATA WRAPPER](#)

[ALTER FOREIGN TABLE](#)

[ALTER FUNCTION](#)

[ALTER GROUP](#)

[ALTER INDEX](#)

[ALTER LANGUAGE](#)

[ALTER OPERATOR](#)

描述

删除文本搜索模板。

概要

```
DROP TEXT SEARCH TEMPLATE [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

描述

`DROP TEXT SEARCH TEMPLATE`删除现有的文本搜索模板。您必须是超级用户才能使用此命令。

您必须是超级用户才能使用`ALTER TEXT SEARCH TEMPLATE`。

参数

`IF EXISTS`

如果该模板不存在，请不要报错。在这种情况下会发出通知。

`name`

现有文本搜索模板的名称（可以由模式指定）。

`CASCADE`

自动删除依赖于文本搜索模板的对象。

`RESTRICT`

如果有任何对象依赖文本搜索模板，则拒绝删除它。这是默认值。



兼容性

在SQL标准中没有DROP TEXT SEARCH TEMPLATE语句。

另见

[ALTER TEXT SEARCH TEMPLATE](#), [CREATE TEXT SEARCH TEMPLATE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除数据类型。

概要

```
DROP TYPE [ IF EXISTS ] name [ , ... ] [ CASCADE | RESTRICT ]
```

描述

`DROP TYPE`将删除用户定义的数据类型。只有类型的所有者才能将其删除。

参数

IF EXISTS

如果该类型不存在，请不要报错。在这种情况下会发出通知。

name

要删除的数据类型的名称（可以由模式指定）。

CASCADE

自动删除依赖于类型的对象（例如表列，函数，运算符）。

RESTRICT

如果有任何对象依赖该类型，则拒绝删除该类型。这是默认值。

示例

删除数据类型`box`；

```
DROP TYPE box;
```



兼容性

除了IF EXISTS选项（这是Greenplum数据库扩展）之外，此命令与SQL标准中的相应命令相似。但是请注意，Greenplum数据库中的许多CREATE TYPE命令和数据类型扩展机制与SQL标准不同。

另见

[ALTER TYPE](#) , [CREATE TYPE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0 文档

□ 参考指南

□ SQL Command Reference

[SQL Syntax Summary](#)[ABORT](#)[ALTER AGGREGATE](#)[ALTER COLLATION](#)[ALTER CONVERSION](#)[ALTER DATABASE](#)[ALTER DEFAULT PRIVILEGES](#)[ALTER DOMAIN](#)[ALTER EXTENSION](#)[ALTER EXTERNAL TABLE](#)[ALTER FOREIGN DATA WRAPPER](#)[ALTER FOREIGN TABLE](#)[ALTER FUNCTION](#)[ALTER GROUP](#)[ALTER INDEX](#)[ALTER LANGUAGE](#)[ALTER OPERATOR](#)

删除数据库角色。

概要

```
DROP USER [IF EXISTS] name [, ...]
```

描述

DROP USER是[DROP ROLE](#)的别名。有关更多信息，请参见[DROP ROLE](#)。

兼容性

SQL标准中没有DROP USER语句。SQL标准将用户的定义留给实现。

另见

[DROP ROLE](#)**Parent topic:** [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

删除外部服务器的用户映射。

概要

```
DROP USER MAPPING [ IF EXISTS ] { username | USER |
CURRENT_USER | PUBLIC }
    SERVER servername
```

描述

`DROP USER MAPPING`从外部服务器中删除现有的用户映射。要执行此命令，当前用户必须是包含映射的服务器的所有者。

参数

IF EXISTS

如果该用户映射不存在，请不要报错。在这种情况下会发出通知。

username

映射的用户名。`CURRENT_USER`和`USER`与当前用户的名称匹配。`PUBLIC`用于匹配系统中所有当前和将来的用户名。

servername

用户映射的服务器名称。

示例

删除名为`bob`的用户映射，如果存在，则删除服务器`foo`

```
DROP USER MAPPING IF EXISTS FOR bob SERVER foo;
```

兼容性

`DROP SERVER`符合ISO/IEC 9075-9 (SQL/MED)。`IF EXISTS`子句是Greenplum数据库扩展。

另见

[CREATE USER MAPPING](#), [ALTER USER MAPPING](#)

Parent topic: [SQL Command Reference](#)

删除视图。

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)

[ABORT](#)

[ALTER AGGREGATE](#)

[ALTER COLLATION](#)

[ALTER CONVERSION](#)

[ALTER DATABASE](#)

[ALTER DEFAULT](#)

[PRIVILEGES](#)

[ALTER DOMAIN](#)

[ALTER EXTENSION](#)

[ALTER EXTERNAL TABLE](#)

[ALTER FOREIGN DATA WRAPPER](#)

[ALTER FOREIGN TABLE](#)

[ALTER FUNCTION](#)

[ALTER GROUP](#)

[ALTER INDEX](#)

[ALTER LANGUAGE](#)

[ALTER OPERATOR](#)

概要

```
DROP VIEW [ IF EXISTS ] name [ , ... ] [ CASCADE | RESTRICT ]
```

描述

`DROP VIEW`将删除现有视图。仅视图所有者可以将其删除。

参数

IF EXISTS

如果该视图不存在，请不要报错。在这种情况下会发出通知。
name

要删除的视图的名称（可以由模式指定）。

CASCADE

自动删除依赖于视图的对象（例如其他视图）。

RESTRICT

如果有任何对象依赖该视图，则拒绝删除该视图。这是默认值。

示例

删除视图`topten`；

```
DROP VIEW topten;
```

兼容性

`DROP VIEW`完全符合SQL标准，但该标准仅允许每个命令删除一个视



图。另外，IF EXISTS选项是Greenplum数据库扩展。

另见

[CREATE VIEW](#)

Parent topic: [SQL Command Reference](#)

提交当前事务。

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command
Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLE

ALTER FOREIGN
DATA WRAPPER

ALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

概要

END [WORK | TRANSACTION]

描述

END提交当前事务。事务所做的所有更改对其他人都可见，即使发生崩溃，也保证是持久的。此命令是Greenplum数据库扩展，等效于[COMMIT](#)。

参数

WORK

TRANSACTION

可选关键字。它们没有作用。

示例

提交当前事务：

END;

兼容性

END是Greenplum数据库扩展，它提供的功能等同于SQL标准中指定的[COMMIT](#)。



另见

[BEGIN](#) , [ROLLBACK](#) , [COMMIT](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

执行准备好的SQL语句。

概要

```
EXECUTE name [ (parameter [, ...] ) ]
```

描述

`EXECUTE`用于执行先前准备的语句。由于准备好的语句仅在会话期间存在，因此准备好的语句必须由当前会话中较早执行的`PREPARE`语句创建。

如果创建该语句的`PREPARE`语句指定了一些参数，则必须将一组兼容的参数传递给`EXECUTE`语句，否则会引发错误。请注意（与函数不同），准备好的语句不会基于其参数的类型或数量而重载。在数据库会话中，准备好的语句的名称必须唯一。

有关准备语句的创建和使用的更多信息，请参见`PREPARE`。

参数

name

要执行的准备好的语句的名称。

parameter

准备语句的参数的实际值。该表达式必须是产生与该参数的数据类型兼容的值的表达式，这是在创建准备语句时确定的。

示例

为`INSERT`语句创建一个准备语句，然后执行它：

```
PREPARE fooplan (int, text, bool, numeric) AS INSERT INTO
foo VALUES($1, $2, $3, $4);
EXECUTE fooplan(1, 'Hunter Valley', 't', 200.00);
```

兼容性

SQL标准包含一个EXECUTE语句，但仅用于嵌入式SQL。此版本的EXECUTE语句还使用了一些不同的语法。

另见

[DEALLOCATE](#) , [PREPARE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

显示语句的查询计划。

概要

```
EXPLAIN [ ( option [, ...] ) ] statement
EXPLAIN [ANALYZE] [VERBOSE] statement
```

其中*option*可以是以下之一：

```
ANALYZE [ boolean ]
VERBOSE [ boolean ]
COSTS [ boolean ]
BUFFERS [ boolean ]
TIMING [ boolean ]
FORMAT { TEXT | XML | JSON | YAML }
```

描述

EXPLAIN显示Greenplum或Postgres优化器为提供的语句生成的查询计划。查询计划是节点的查询树计划。计划中的每个节点代表一个单独的操作，例如表扫描，连接，聚合或排序。

应从下至上阅读计划，因为每个节点都会向其上方的节点中发送行。计划的最底层节点通常是表扫描操作（顺序扫描，索引扫描或位图索引扫描）。如果查询需要连接，聚集或排序（或原始行上的其他操作），则扫描节点上方将有其他节点来执行这些操作。最顶层的计划节点通常是Greenplum数据库motion节点（重新分发，显式重新分发，广播或收集motion）。这些操作负责在查询处理期间在segment实例之间移动行。

EXPLAIN的输出对于计划树中的每个节点都有一行，显示基本节点类型以及计划者为执行该计划节点而进行的以下成本估算：

- **cost** — 优化器对运行该语句要花费多长时间的猜测（以任意成本单位衡量，但通常是指磁盘页获取）。显示了两个成本编号：可以返回第一行之前的启动成本，以及返回所有行的总成本。请注意，总成本假定将检索所有行，但并非总是如此（例如，如果使用LIMIT）。

- **rows** —

此计划节点输出的总行数。这通常少于计划节点处理或扫描的实际行数，反映了任何WHERE子句条件的估计选择性。理想情况下，顶级节点估计将近似查询实际返回，更新或删除的行数。

- **width** — 此计划节点输出的所有行的总字节数。

重要的是要注意，上级节点的成本包括其所有子节点的成本。计划的最高节点具有该计划的估计总执行成本。这是计划者要尽量减少的数字。同样重要的是要意识到，成本只反映查询优化器关心的事情。特别是，成本不考虑将结果行传输到客户端所花费的时间。

`EXPLAIN ANALYZE`导致语句实际执行，而不仅仅是做计划。`EXPLAIN ANALYZE`优化器会显示实际结果以及计划者的估计。这对于查看优化器的估计是否接近实际很有用。除了`EXPLAIN`计划中显示的信息之外，`EXPLAIN ANALYZE`还将显示以下附加信息：

- 运行查询所花费的总时间（以毫秒为单位）。
- 计划节点操作中涉及的*workers* (segment) 数。仅计算返回行的segment。
- 操作产生最多行的segment所返回的最大行数。如果多个segment产生相等数量的行，则结束时间最长的一个就是选择的那个。
- 为一个操作生成最多行的segment的segment ID号。
- 对于相关操作，该操作使用的*work_mem*。如果*work_mem*不足以在内存中执行该操作，则该计划将显示有多少数据溢出到磁盘，以及最低性能segment需要多少次数据传递。例如：

```
Work_mem used: 64K bytes avg, 64K bytes max (seg0).
Work_mem wanted: 90K bytes avg, 90K bytes max (seg0) to
abate workfile
I/O affecting 2 workers.
[seg0] pass 0: 488 groups made from 488 rows; 263 rows
written to
workfile
[seg0] pass 1: 263 groups made from 263 rows
```

- 从产生最多行的segment中检索第一行所花费的时间（以毫秒为单位），以及从该segment中检索所有行所花费的总时间。如果`<time> to first row`与`<time> to end`相同，则可以省略。

Important: 请记住，使用ANALYZE时实际上会执行该语句。尽管`EXPLAIN ANALYZE`将丢弃SELECT将返回的任何输出，但是该语句的其他副作用将照常发生。如果希望在DML语句上使用`EXPLAIN ANALYZE`而不让命令影响您的数据，请使用以下方法：

```
BEGIN;
EXPLAIN ANALYZE ...;
ROLLBACK;
```

仅可以指定ANALYZE和VERBOSE选项，并且只能按该顺序指定，而不要在括号中包含选项列表。

参数

ANALYZE

执行命令并显示实际运行时间和其他统计信息。如果省略此参数，则默认为FALSE。指定ANALYZE true可以启用它。

VERBOSE

显示有关计划的其他信息。具体来说，包括计划树中每个节点的输出列列表，模式限定表和函数名称，始终在表达式中使用范围表别名标记变量，并始终打印要显示其统计信息的每个触发器的名称。如果省略此参数，则默认为FALSE；指定VERBOSE true启用它。

COSTS

包括有关每个计划节点的估计启动成本和总成本以及估计的行数和估计的每行宽度的信息。如果省略此参数，则默认为TRUE；指定COSTS false禁用它。

BUFFERS

包括有关缓冲区使用情况的信息。具体来说，包括命中，读取，弄脏和写入的共享块的数量，命中，读取，弄脏和写入的局部块的数量以及读写的临时块的数量。命中表示避免读取，因为在需要时已在高速缓存中找到该块。共享块包含来自常规表和索引的数据；本地块包含来自临时表和索引的数据；临时块包含用于排序，哈希，物化计划节点和类似情况的短期工作数据。被弄脏的块数表示此查询已更改的先前未修改的块数；而写入的块数则表示此后端在查询处理期间从缓存中逐出的先前处理的块数。上级节点显示的块数包括其所有子节点使用的块数。在文本格式中，仅打印非零值。仅当还启用了ANALYZE时，才可以使用此参数。如果省略此参数，则默认为FALSE；指定BUFFERS true启用它。

TIMING

在输出中包括实际的启动时间和在每个节点上花费的时间。重複读取系统时钟的开销可能会在某些系统上显着降低查询速度，因此，当仅需要实际的行计数而不是确切的时间时，将此参数设置为FALSE可能会很有用。即使使用此选项关闭了节点级计时，也始终会测量整个语句的运行时间。仅当还启用了ANALYZE时，才可以使用此参数。默认为TRUE。

FORMAT

指定输出格式，可以是TEXT，XML，JSON或YAML。非文本输出包含与文本输出格式相同的信息，但程序更易于解析。此参

数默认为TEXT。

boolean

指定是打开还是关闭所选选项。您可以写入TRUE, ON或1以启用该选项，而可以写入FALSE, OFF或0以禁用该选项。布尔值也可以省略，在这种情况下，假定为TRUE。

statement

您希望查看其执行计划的任何SELECT, INSERT, UPDATE, DELETE, VALUES, EXECUTE, DECLARE或CREATE TABLE AS语句。

注解

为了使查询优化器在优化查询时能够做出合理的决策，应运行ANALYZE语句以记录有关表内数据分布的统计信息。如果您尚未执行此操作（或者自上次运行ANALYZE以来，表中数据的统计分布已发生重大变化），则估计成本不太可能符合查询的实际属性，因此可能会选一个较差的查询计划。

在执行EXPLAIN ANALYZE命令期间运行的SQL语句从Greenplum数据库资源队列中排除。

有关查询分析的更多信息，请参阅*Greenplum*数据库管理员指南中的“查询分析”。有关资源队列的更多信息，请参阅*Greenplum*数据库管理员指南中的“使用资源队列进行资源管理”。

示例

为了说明如何读取EXPLAIN查询计划，请考虑一个非常简单的查询的示例：

```
EXPLAIN SELECT * FROM names WHERE name = 'Joelle';
               QUERY PLAN
-----
-----
      Gather Motion 3:1  (slice1; segments: 3)
      (cost=0.00..431.27 rows=1 width=58)
          ->  Seq Scan on names  (cost=0.00..431.27 rows=1
width=58)
                  Filter: (name = 'Joelle'::text)
      Optimizer: Pivotal Optimizer (GPORCA) version 3.23.0
      (4 rows)
```

如果我们自下而上阅读计划，则查询优化器将从对names表的顺序扫

描开始。请注意，WHERE子句被用作过滤条件。这意味着扫描操作将检查其扫描的每一行的条件，并仅输出通过条件的行。

扫描操作的结果将传递到*gather motion*操作。在Greenplum数据库中，*gather motion*是将segment行发送到master。在这种情况下，我们有3个segment实例发送到1个master实例（3: 1）。该操作在并行查询执行计划的slice1上进行。在Greenplum数据库中，查询计划分为多个切片，以便查询计划的各个部分可以由这些segment并行处理。

该计划的估计启动成本为00.00（无成本），总成本为431.27。优化器估计此查询将返回一行。

这是相同的查询，但成本估算被抑制：

```
EXPLAIN (COSTS FALSE) SELECT * FROM names WHERE name =
'Joelle';
QUERY PLAN
-----
Gather Motion 3:1 (slice1; segments: 3)
-> Seq Scan on names
  Filter: (name = 'Joelle'::text)
Optimizer: Pivotal Optimizer (GPORCA) version 3.23.0
(4 rows)
```

这是使用JSON格式的相同查询：

```
EXPLAIN (FORMAT JSON) SELECT * FROM names WHERE name =
'Joelle';
QUERY PLAN
-----
[ + 
{ + 
  "Plan": { +
    "Node Type": "Gather Motion", +
    "Senders": 3, +
    "Receivers": 1, +
    "Slice": 1, +
    "Segments": 3, +
    "Gang Type": "primary reader", +
    "Startup Cost": 0.00, +
    "Total Cost": 431.27, +
    "Plan Rows": 1, +
    "Plan Width": 58, +
    "Plans": [ +
      { +
        "Node Type": "Seq Scan", +
        "Parent Relationship": "Outer", +
        "Slice": 1, +
        "Segments": 3, +
        "Gang Type": "primary reader", +
        "Relation Name": "names", +
        "Alias": "names", +
```

```

        "Startup Cost": 0.00,
        "Total Cost": 431.27,
        "Plan Rows": 1,
        "Plan Width": 58,
        "Filter": "(name = 'Joelle')::text"
    }
]
},
"Settings": {
    "Optimizer": "Pivotal Optimizer (GPORCA) version
3.23.0"
}
]
(1 row)

```

如果存在索引，并且我们使用带有可索引WHERE条件的查询，则EXPLAIN可能会显示不同的计划。此查询使用YAML格式生成带有索引扫描的计划：

```

EXPLAIN (FORMAT YAML) SELECT * FROM NAMES WHERE
LOCATION='Sydney, Australia';
QUERY PLAN
-----
-- Plan:
+
    Node Type: "Gather Motion"
+
        Senders: 3
+
        Receivers: 1
+
        Slice: 1
+
        Segments: 3
+
        Gang Type: "primary reader"
+
        Startup Cost: 0.00
+
        Total Cost: 10.81
+
        Plan Rows: 10000
+
        Plan Width: 70
+
        Plans:
+
            - Node Type: "Index Scan"
+
                Parent Relationship: "Outer"
+
                Slice: 1
+
                Segments: 3

```

```
+          Gang Type: "primary reader"
+
+          Scan Direction: "Forward"
+
+          Index Name: "names_idx_loc"
+
+          Relation Name: "names"
+
+          Alias: "names"
+
+          Startup Cost: 0.00
+
+          Total Cost: 7.77
+
+          Plan Rows: 10000
+
+          Plan Width: 70
+
+          Index Cond: "(location = 'Sydney,
Australia'::text)"+          Settings:
+
+          Optimizer: "Pivotal Optimizer (GPORCA) version 3.23.0"
(1 row)
```

兼容性

在SQL标准中没有定义EXPLAIN语句。

另见

[ANALYZE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

使用游标从查询中检索行。

概要

```
FETCH [ forward_direction { FROM | IN } ]
      cursor_name
```

其中*forward_direction*可以为空或以下之一：

NEXT
 FIRST
 LAST
 ABSOLUTE *count*
 RELATIVE *count*
count
 ALL
 FORWARD
 FORWARD *count*
 FORWARD ALL

描述

FETCH使用先前创建的游标检索行。

Note: 本页在SQL命令级别描述游标的用法。如果试图在PL/pgSQL函数中使用游标，则规则是不同的。参见[Greenplum PL/pgSQL过程语言](#)。

游标具有关联的位置，该位置由FETCH使用。光标位置可以在查询结果的第一行之前，结果的任何特定行上，或者在结果的最后一行之后。创建后，光标将位于第一行之前。在获取了一些行之后，将光标定位在最近检索到的行上。如果FETCH在可用行的末尾运行，则光标将位于最后一行之后。FETCH ALL将始终使光标位于最后一行之后。

NEXT, FIRST, LAST, ABSOLUTE, RELATIVE形式有 移动光标后获取单行。如果没有这样的行，则返回空结果，并且光标将视情况放在第一行之前或最后一行之后。

使用FORWARD的表单将检索指示的向前行的行数，将光标留在最后返回的行上（如果计数超过可用行数，则在所有行之后）。请注意，由于不支持可滚动光标，因此无法在Greenplum数据库中后向移动光标位置。您只能使用FETCH向前移动光标。

RELATIVE 0和FORWARD 0在不移动光标的情况下请求获取当前行，即重新获取最近获取的行。除非光标位于第一行之前或最后一行之后，否则此操作将成功，在这种情况下不会返回任何行。

输出

成功完成后，FETCH命令将返回以下形式的命令标签：

```
FETCH count
```

该计数是获取的行数（可能为零）。请注意，在psql中，实际上不会显示command标记，因为psql只是显示提取的行。

参数

forward_direction

定义获取方向和要获取的行数。Greenplum数据库中仅允许前向提取。可以是以下之一：

NEXT

获取下一行。如果省略了方向，则为默认设置。

FIRST

提取查询的第一行（与ABSOLUTE 1相同）。仅当它是使用此光标的第一个FETCH操作时才允许。

LAST

获取查询的最后一行（与ABSOLUTE -1相同）。

ABSOLUTE *count*

获取查询的指定行。如果计数超出范围，则在最后一行之后定位。只有在被指定了*count*的行向前移动游标位置才能使用。

RELATIVE *count*

在当前光标位置之前获取查询*count*行的指定行。RELATIVE 0重新获取当前行（如果有）。仅当*count*向前移动光标位置时才允许。

count

获取下一个*count*行数（与FORWARD *count*相同）。

ALL

提取所有剩余的行（与FORWARD ALL相同）。

FORWARD

获取下一行（与NEXT相同）。

FORWARD count

获取下一个*count*行数。 FORWARD 0重新获取当前行。

FORWARD ALL

提取所有剩余的行。

cursor_name

打开的游标的名称。

注解

Greenplum数据库不支持可滚动光标，因此只能使用FETCH向前移动光标位置。

ABSOLUTE获取并不比通过相对移动导航到所需行更快：底层实现无论如何都必须遍历所有中间行。

DECLARE用于定义游标。使用MOVE更改光标位置而不检索数据。

示例

-- 开始事务：

```
BEGIN;
```

-- 设置游标：

```
DECLARE mycursor CURSOR FOR SELECT * FROM films;
```

-- 获取游标mycursor中的前5行：

```
FETCH FORWARD 5 FROM mycursor;
code | title | did | date_prod | kind | len
-----+-----+-----+-----+-----+-----+
-----+-----+
BL101 | The Third Man | 101 | 1949-12-23 | Drama | 01:44
BL102 | The African Queen | 101 | 1951-08-11 | Romantic | 01:43
JL201 | Une Femme est une Femme | 102 | 1961-03-12 |
Romantic | 01:25
P_301 | Vertigo | 103 | 1958-11-14 | Action | 02:08
P_302 | Becket | 103 | 1964-02-03 | Drama | 02:28
```

-- 关闭游标并提交事务：

```
CLOSE mycursor;  
COMMIT;
```

在c_films光标当前位置的行中更改表films的kind列：

```
UPDATE films SET kind = 'Dramatic' WHERE CURRENT OF  
c_films;
```

兼容性

SQL标准仅允许在嵌入式SQL和模块中使用游标。 Greenplum数据库允许以交互方式使用游标。

这里描述的FETCH的变量将数据作为SELECT结果返回，而不是将其放在主机变量中。 除此之外，FETCH与SQL标准完全向上兼容。

涉及FORWARD的FETCH形式以及其中隐含FORWARD的FETCH count和FETCH ALL形式都是Greenplum数据库扩展。 不支持BACKWARD。

SQL标准只允许在游标名称之前使用FROM。 使用IN或完全不使用它们的选项是扩展。

另见

[DECLARE](#) , [CLOSE](#) , [MOVE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定义访问权限。

概要

```

GRANT { {SELECT | INSERT | UPDATE | DELETE | REFERENCES |
TRIGGER | TRUNCATE} [, ...] | ALL [PRIVILEGES] }
ON { [TABLE] table_name [, ...]
    | ALL TABLES IN SCHEMA schema_name [, ...] }
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
OPTION ]

```



```

GRANT { { SELECT | INSERT | UPDATE | REFERENCES } (
column_name [, ...] )
[, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
ON [ TABLE ] table_name [, ...]
TO { role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

```



```

GRANT { {USAGE | SELECT | UPDATE} [, ...] | ALL
[PRIVILEGES] }
ON { SEQUENCE sequence_name [, ...]
    | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH
GRANT OPTION ]

```



```

GRANT { {CREATE | CONNECT | TEMPORARY | TEMP} [, ...] | ALL
[PRIVILEGES] }
ON DATABASE database_name [, ...]
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH
GRANT OPTION ]

```



```

GRANT { USAGE | ALL [ PRIVILEGES ] }
ON DOMAIN domain_name [, ...]
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH
GRANT OPTION ]

```



```

GRANT { USAGE | ALL [ PRIVILEGES ] }
ON FOREIGN DATA WRAPPER fdw_name [, ...]
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH
GRANT OPTION ]

```



```

GRANT { USAGE | ALL [ PRIVILEGES ] }
ON FOREIGN SERVER server_name [, ...]
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH
GRANT OPTION ]

```



```

GRANT { EXECUTE | ALL [PRIVILEGES] }
ON { FUNCTION function_name ( [ [ argmode ] | argname ]
argtype [, ...]
] ) [, ...]
    | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }

```



```
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH  
GRANT OPTION ]  
  
GRANT { USAGE | ALL [PRIVILEGES] }  
ON LANGUAGE lang_name [, ...]  
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH  
GRANT OPTION ]  
  
GRANT { { CREATE | USAGE } [, ...] | ALL [PRIVILEGES] }  
ON SCHEMA schema_name [, ...]  
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH  
GRANT OPTION ]  
  
GRANT { CREATE | ALL [PRIVILEGES] }  
ON TABLESPACE tablespace_name [, ...]  
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH  
GRANT OPTION ]  
  
GRANT { USAGE | ALL [ PRIVILEGES ] }  
ON TYPE type_name [, ...]  
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH  
GRANT OPTION ]  
  
GRANT parent_role [, ...]  
TO member_role [, ...] [WITH ADMIN OPTION]  
  
GRANT { SELECT | INSERT | ALL [PRIVILEGES] }  
ON PROTOCOL protocolname  
TO username
```

描述

Greenplum数据库将用户和组的概念统一为一种称为角色的实体。因此，不必使用关键字GROUP来标识被授予者是用户还是组。命令中仍然允许使用GROUP，但这是一个干扰词。

GRANT命令具有两种基本变体：一种对数据库对象（表，列，视图，外部表，序列，数据库，外部数据包装器，外部服务器，函数，过程语言，模式或表空间）授予特权，以及授予一个角色中的成员关系。

数据库对象上的GRANT

GRANT命令的此变体将一个数据库对象的特定特权授予一个或多个角色。这些特权将添加到已授予的特权（如果有）中。

还有一个选项可以授予一个或多个模式中所有相同类型的对象的特权。当前仅表，序列和函数支持此功能（但请注意，ALL TABLES被视为包括视图和外部表）。

关键字PUBLIC表示将特权授予所有角色，包括以后可能创建的角色。可以将PUBLIC视为始终包含所有角色的隐式定义的组级别角色。任何特定角色将具有直接授予它的特权，授予它当前所属的任何角色的特权以及授予PUBLIC的特权的总和。

如果指定了WITH GRANT OPTION，则特权的接收者可以依次将其授予其他人。没有grant option，接收者将无法做到这一点。Grant options不能授予PUBLIC。

无需向对象的所有者（通常是创建该对象的角色）授予特权，因为所有者默认情况下具有所有特权。（但是，所有者可以出于安全考虑选择撤销自己的某些特权。）

删除对象或以任何方式更改其定义的权利不视为可授予的特权；它是所有者固有的，不能被授予或撤销。（但是，通过授予或撤消拥有对象的角色的成员资格，可以获得类似的效果；请参阅下文。）所有者也隐式拥有该对象的所有授予选项。

Greenplum数据库将某些类型的对象的默认特权授予PUBLIC。默认情况下，不将在表，表列，序列，外部数据包装器，外部服务器，大型对象，模式或表空间上的权限授予PUBLIC。对于其他类型的对象，授予PUBLIC的默认特权如下：

- 数据库的CONNECT和TEMPORARY（创建临时表）特权，
- 函数的EXECUTE特权
- 语言和数据类型（包括域）的USAGE特权。

当然，对象所有者可以REVOKE默认特权和明确授予的特权。（为了获得最大的安全性，请在创建对象的同一事务中执行REVOKE；因此，没有任何时间窗口可供其他用户使用该对象。）

角色上的GRANT

GRANT命令的此变体将一个角色的成员资格授予一个或多个其他角色。角色的成员资格意义重大，因为它可以将授予角色的特权传达给每个成员。

如果指定了WITH ADMIN OPTION，则成员可以依次将角色的成员资格授予其他人，也可以撤消该角色的成员资格。没有admin选项，普通用户将无法做到这一点。角色不被认为拥有WITH ADMIN OPTION本身，但是它可以从中话用户与角色匹配的数据库会话中授予或撤消成员资格。数据库超级用户可以以任何角色向任何人授予或撤消成员资格。具有CREATEROLE特权的角色可以授予或撤消不是超级用户的任何角色的成员资格。

与具有特权的情况不同，不能将角色的成员资格授予PUBLIC。

协议上的GRANT

您还可以使用GRANT命令指定哪些用户可以访问受信任的协议。（如果该协议不受信任，则您不能授予任何其他用户权限以使用它来读取或写入数据。）

- 要允许用户使用受信任的协议创建可读的外部表：

```
GRANT SELECT ON PROTOCOL protocolname TO username
```

- 要允许用户使用受信任的协议创建可写外部表：

```
GRANT INSERT ON PROTOCOL protocolname TO username
```

- 要允许用户使用受信任的协议创建可读和可写的外部表：

```
GRANT ALL ON PROTOCOL protocolname TO username
```

您还可以使用此命令来授予用户创建和使用s3和pxf外部表的权限。但是，类型为http, https, gpfdist和gpfdists的外部表是在Greenplum数据库内部实现的，而不是作为自定义协议来实现的。对于这些类型，请使用CREATE ROLE或ALTER ROLE命令为每个用户设置CREATEEXTTABLE或NOCREATEEXTTABLE属性。有关语法和示例，请参见[CREATE ROLE](#)。

参数

SELECT

允许从指定表，视图或序列的任何列或列出的特定列中进行SELECT。还允许使用COPY TO。引用UPDATE或DELETE中现有的列值也需要此特权。

INSERT

允许将新行INSERT到指定表中。如果列出了特定的列，则只能在INSERT命令中指定那些列（其他列将接收默认值）。还允许COPY FROM。

UPDATE

允许UPDATE指定表的任何列或列出的特定列。SELECT ... FOR UPDATE和SELECT ... FOR SHARE在至少一列上也需要此特权（以及SELECT特权）。对于序列，此特权允许使用nextval()和setval()函数。

DELETE

允许从指定表中DELETE一行。

REFERENCES

尽管Greenplum数据库当前不支持外键约束，但仍可接受此关键字。要创建外键约束，必须在引用列和被引用列上都具有此特权。可以为表的所有列或仅特定列授予特权。

TRIGGER

允许在指定的表上创建触发器。

Note: Greenplum数据库不支持触发器。

TRUNCATE

允许对指定表中的所有行进行TRUNCATE。

CREATE

对于数据库，允许在数据库中创建新的schema。

对于schema，允许在schema内创建新对象。要重命名现有对象，您必须拥有该对象，并对包含的schema具有此特权。

对于表空间，允许在表空间内创建表和索引，并允许创建将表空间作为其默认表空间的数据库。（请注意，撤消此特权不会更改现有对象的位置。）

CONNECT

允许用户连接到指定的数据库。在连接启动时会检查此特权（除了检查pg_hba.conf施加的任何限制）。

TEMPORARY

TEMP

允许在使用数据库时创建临时表。

EXECUTE

允许使用指定的函数以及使用在该函数之上实现的任何运算符。

这是适用于函数的唯一特权类型。（此语法也适用于聚合函数。）

USAGE

对于过程语言，允许使用指定的语言来创建该语言的函数。这是适用于过程语言的唯一特权类型。

对于schema，允许访问指定schema中包含的对象（假设还满足对象自己的特权要求）。从本质上讲，这允许被授予者在schema中查找对象。

对于序列，此特权允许使用currval()和nextval()函数。

对于类型和域，此特权允许在创建表，函数和其他schema对象时使用类型或域。（请注意，它不控制类型的一般“用法”，例如查询中出现的类型的值。它仅防止创建依赖于类型的对象。特权的主要目的是控制哪些用户基于类型创建依赖关系，这可能会阻止所有者稍后更改类型。）

对于外部数据包装器，此特权使被授予者可以使用该外部数据包装器创建新服务器。

对于服务器，此特权使被授予者可以使用服务器创建外部表，还可以创建，更改或删除其自己与该服务器关联的用户的用户映射。

ALL PRIVILEGES

一次授予所有可用特权。 `PRIVILEGES`关键字在Greenplum数据库中是可选的，尽管严格的SQL要求使用。

PUBLIC

特殊的组级角色，表示将特权授予所有角色，包括以后可能创建的角色。

WITH GRANT OPTION

特权的接收者可以依次将其授予他人。

WITH ADMIN OPTION

角色的成员又可以将角色的成员身份授予其他人。

注解

如果用户对特定列或整个表拥有该特权，则用户可以在该列上执行`SELECT`, `INSERT`等操作。在表级别上授予特权，然后将其撤销一列，这并不像你预想的那样：表级授予不受列级操作的影响。

数据库超级用户可以访问所有对象，而不管对象特权设置如何。该对象的一个例外是视图对象。对视图中引用的表的访问权限是由视图所有者（不是当前用户，即使当前用户是超级用户）的权限决定的。

如果超级用户选择执行`GRANT`或`REVOKE`命令，则该命令的执行就像是由受影响对象的所有者执行的一样。特别是，通过此类命令授予的特权似乎已由对象所有者授予。对于角色成员资格，成员资格似乎已由包含它的角色授予。

`GRANT`和`REVOKE`也可以由不是受影响对象的所有者完成，但要求是拥有该对象的角色的成员，或者是拥有对该对象的`WITH GRANT OPTION`特权的角色的成员来完成。在这种情况下，特权将被记录为已由实际拥有对象或拥有`WITH GRANT OPTION`特权的角色授予。

授予表权限不会自动将权限扩展到该表使用的任何序列，包括与`SERIAL`列绑定的序列。序列的权限必须单独设置。

`GRANT`命令不能用于为协议`file`, `gpfdist`或`gpfdists`设置特权。这些协议在Greenplum数据库内部实现。而是使用`CREATE ROLE`或`ALTER ROLE`命令来设置角色的`CREATEEXTTABLE`属性。

使用`psql`的`\dp`元命令来获取有关表和列的现有特权的信息。您还可以使用其他`\d`元命令来显示非表对象的特权。

示例

向表mytable上的所有角色授予插入权限：

```
GRANT INSERT ON mytable TO PUBLIC;
```

将所有可用的特权授予在topten视图中的角色sally。请注意，如果上述内容的确会由超级用户或topten的所有者执行，将会授予所有特权，当由其他人执行时，它只会授予那些授予角色具有授予选项的那些权限。

```
GRANT ALL PRIVILEGES ON topten TO sally;
```

将角色admins的成员资格授予用户joe：

```
GRANT admins TO joe;
```

兼容性

PRIVILEGES关键字在SQL标准中是必需的，但在Greenplum数据库中是可选的。SQL标准不支持为每个命令在多个对象上设置特权。

Greenplum数据库允许对象所有者撤消他们自己的普通特权：例如，表所有者可以通过撤消其自己的INSERT, UPDATE, DELETE和TRUNCATE特权使该表对自己只读。根据SQL标准，这是不可能的。Greenplum数据库将所有者的特权视为由所有者授予了所有者；因此他们也可以撤销它们。在SQL标准中，所有者的特权由假定的`system`实体授予。

SQL标准为其他类型的对象提供USAGE特权：字符集，排序规则，翻译。

在SQL标准中，序列仅具有USAGE特权，该特权控制NEXT VALUE FOR表达式的使用，该表达式等效于Greenplum数据库中的`nextval`函数。序列特权SELECT和UPDATE是Greenplum数据库扩展。对`currval`函数应用序列USAGE特权也是Greenplum数据库扩展（函数本身也是）。

数据库，表空间，模式和语言的特权是Greenplum数据库扩展。

另见

[REVOKE](#) , [CREATE ROLE](#) , [ALTER ROLE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

在表中创建新行。

概要

```
[ WITH [ RECURSIVE ] with_query [ , ... ] ]
INSERT INTO table [ ( column [ , ... ] ) ]
    {DEFAULT VALUES | VALUES ( {expression | DEFAULT} [ ,
... ] ) [ , ... ] | query}
    [RETURNING * | output_expression [[AS] output_name] [ ,
... ]]
```

描述

INSERT将新行插入表中。可以插入由值表达式指定的一或多个行，或查询产生的零或多个行。

目标列名称可以按任何顺序列出。如果根本没有给出任何列名列表，则默认为表中的列按声明的顺序排列。VALUES子句或查询提供的值与从左到右的显式或隐式列列表相关联。

在显式或隐式列列表中不存在的每个列都将填充一个默认值，该默认值可以是其声明的默认值，如果没有默认值，则为null。

如果任何列的表达式的数据类型都不正确，则将尝试自动类型转换。

可选的RETURNING子句使INSERT根据实际插入的每一行计算并返回值。这对于获取默认提供的值（例如序列号）很有用。但是，允许使用表列的任何表达式。RETURNING列表的语法与SELECT的输出列表的语法相同。

您必须对表具有INSERT特权才能插入表中。指定列列表后，仅需要对列出的列具有INSERT特权。使用RETURNING子句需要RETURNING中提到的所有列都具有SELECT特权。如果提供查询以插入查询中的行，则必须对查询中引用的任何表或列具有SELECT特权。

输出

成功完成后，INSERT命令将返回以下形式的命令标记：



```
INSERT oid count
```

*count*是插入的行数。如果*count*恰好为1，并且目标表具有OID，则*oid*是分配给插入行的OID。否则，*oid*为零。

参数

with_query

WITH子句允许您指定一个或多个子查询，这些子查询可以在INSERT查询中按名称引用。

对于包含WITH子句的INSERT命令，该子句只能包含SELECT语句，而WITH子句不能包含数据修改命令（INSERT, UPDATE或DELETE）。

查询(SELECT语句)也可能包含WITH子句。在这种情况下，可以在INSERT查询中引用两套*with_query*，但是第二套优先，因为它的嵌套更紧密。

更多信息，请参考[WITH查询 \(公用表表达式\)](#) 和[SELECT](#)。

table

现有表的名称（可以由schema限定）。

column

表中列的名称。如果需要，可以使用子字段名称或数组下标来限定列名称。（仅插入到复合列的某些字段中，未指定的字段为空。）

DEFAULT VALUES

所有列均将填充其默认值。

expression

要分配给相应列的表达式或值。

DEFAULT

相应的列将填充其默认值。

query

提供要插入的行的查询（SELECT语句）。有关语法的说明，请参见SELECT语句。

output_expression

插入每行后，由INSERT命令计算并返回的表达式。该表达式可以使用表的任何列名称。写入*返回插入行的所有列。

output_name

用于返回的列的名称。

注解

要将数据插入分区表中，请指定根分区表，即使用CREATE TABLE命令创建的表。您还可以在INSERT命令中指定分区表的叶子表。如果数据对于指定的叶子表无效，则返回错误。不支持在INSERT命令中指定不是叶子表的子表。不支持在分区表的任何子表上执行其他DML命令，例如UPDATE和DELETE。这些命令必须在根分区表（使用CREATE TABLE命令创建的表）上执行。

对于追加优化表，Greenplum数据库最多可将127个并发INSERT事务插入单个追加优化表中。

对于可写的S3外部表，INSERT操作将上传到已配置的S3存储桶中的一个或多个文件，如[S3:// 协议](#)中所述。按Ctrl-c取消INSERT，并停止上传到S3。

示例

在表films中插入一行：

```
INSERT INTO films VALUES ('UA502', 'Bananas', 105,
'1971-07-13', 'Comedy', '82 minutes');
```

在此示例中，省略了length列，因此它将具有默认值：

```
INSERT INTO films (code, title, did, date_prod, kind)
VALUES
('T_601', 'Yojimbo', 106, '1961-06-16', 'Drama');
```

本示例对date_prod列使用DEFAULT子句，而不是指定值：

```
INSERT INTO films VALUES ('UA502', 'Bananas', 105, DEFAULT,
'Comedy', '82 minutes');
```

要插入完全由默认值组成的行：

```
INSERT INTO films DEFAULT VALUES;
```

要使用多行VALUES语法插入多行：

```
INSERT INTO films (code, title, did, date_prod, kind)
VALUES
('B6717', 'Tampopo', 110, '1985-02-10', 'Comedy'),
('HG120', 'The Dinner Game', 140, DEFAULT, 'Comedy');
```

本示例从表films中向表films中插入一些行，其列布局与films相同：

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

在表distributors中插入一行，返回由`DEFAULT`子句生成的序列号：

```
INSERT INTO distributors (did, dname) VALUES (DEFAULT, 'XYZ  
Widgets')  
RETURNING did;
```

兼容性

INSERT符合SQL标准。 标准不允许列名列表被省略，但不是所有的列都由VALUES子句或查询填充的情况。

SELECT记录了query子句的可能限制。

另见

[COPY, SELECT, CREATE EXTERNAL TABLE, s3:// 协议](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

加载或重新加载共享库文件。

概要

LOAD '*filename*'

描述

此命令将共享库文件加载到Greenplum数据库服务器地址空间中。如果文件先前已加载，则首先将其卸载。此命令主要用于卸载和重新加载自服务器首次加载以来已更改的共享库文件。要使用共享库，需要使用CREATE FUNCTION命令声明其中的函数。

文件名的指定方法与CREATE FUNCTION中共享库名的指定方法相同。特别是，可能依赖于搜索路径和系统标准共享库文件名扩展的自动添加。

请注意，在Greenplum数据库中，共享库文件（.so文件）必须位于Greenplum数据库阵列（master, segments和mirrors）中每个主机的相同路径。

只有数据库超级用户可以加载共享库文件。

参数

filename

共享库文件的路径和文件名。此文件必须存在于Greenplum数据库阵列中所有主机上的相同位置。

示例

加载共享库文件：

LOAD '/usr/local/greenplum-db/lib/myfuncs.so';



兼容性

LOAD是Greenplum数据库扩展。

另见

[CREATE FUNCTION](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

锁表。

概要

```
LOCK [TABLE] [ONLY] name [*] [, ...] [IN lockmode MODE]
[NOWAIT]
```

其中*lockmode*是以下之一：

```
ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE
EXCLUSIVE
| SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS
EXCLUSIVE
```

描述

LOCK TABLE获取表级别的锁，必要时等待释放任何冲突的锁。如果指定了NOWAIT，则LOCK TABLE不会等待获取所需的锁：如果无法立即获取，则命令将中止并发出错误。一旦获得，该锁将在当前事务的其余部分保持。没有UNLOCK TABLE命令；锁始终在事务结束时释放。

当自动获取引用表的命令的锁时，Greenplum数据库始终使用限制最小的锁模式。LOCK TABLE提供了您可能需要更多限制性锁定的情况。例如，假设应用程序在**Read Committed**隔离级别上运行事务，并且需要确保表中的数据在事务期间保持稳定。为此，您可以在查询之前在表上获得SHARE锁定模式。这将防止并发数据更改，并确保后续读取表时都能看到已提交数据的稳定视图，因为SHARE锁定模式与写入者获取的ROW EXCLUSIVE锁冲突，并且您的LOCK TABLE *name IN SHARE MODE*语句将等待直到任何并发持有者ROW EXCLUSIVE模式的锁将锁定提交或回滚。因此，一旦获得了锁，就不会有未提交的未完成的写操作。此外，在您释放锁之前，任何人都无法开始。

为了在REPEATABLE READ或SERIALIZABLE隔离级别上执行事务时达到类似的效果，必须在执行任何SELECT或数据修改语句之前执行LOCK TABLE语句。当REPEATABLE READ或SERIALIZABLE事务

的数据视图的第一个SELECT或数据修改语句开始时，将被冻结。事务中稍后的LOCK TABLE仍将阻止并发写入 - 但不能确保事务读取的内容与最新的提交值相对应。

如果此类事务要更改表中的数据，则应使用SHARE ROW EXCLUSIVE锁定模式而不是SHARE模式。这样可以确保一次仅运行一个这种类型的事务。否则，可能会导致死锁：两个事务可能都同时获取SHARE模式，然后又无法获取ROW EXCLUSIVE模式来实际执行其更新。请注意，事务自身的锁永远不会发生冲突，因此，当事务拥有SHARE模式时，它可以获取ROW EXCLUSIVE模式，但如果其他人拥有SHARE模式，则不行。为避免死锁，请确保所有事务以相同的顺序获取对相同对象的锁定，并且如果单个对象涉及多个锁定模式，则事务应始终首先获取限制性最强的模式。

参数

name

要锁定的现有表的名称（可以是schema限定）。如果指定ONLY，则仅锁定该表。如果未指定ONLY，则表及其所有子表（如果有）将被锁定。（可选）可以在表名称后指定*，以明确指示包括子表。

如果给出了多个表，则表将按照LOCK TABLE命令中指定的顺序一张一张地锁定。

lockmode

锁定模式指定与该锁定冲突的锁定。如果未指定锁定模式，则使用限制最大的ACCESS EXCLUSIVE模式。锁定方式如下：

- ACCESS SHARE — 仅与ACCESS EXCLUSIVE锁定模式冲突。SELECT命令在引用的表上获得此模式的锁定。通常，任何仅读取表而不修改表的查询都将获得此锁定模式。
- ROW SHARE — 与EXCLUSIVE和ACCESS EXCLUSIVE锁定模式冲突。SELECT FOR SHARE命令自动在目标表上获得此模式的锁定（除了在已引用但未选择FOR SHARE的任何其他表上的ACCESS SHARE锁定之外）。
- ROW EXCLUSIVE — 与SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE和ACCESS EXCLUSIVE锁定模式冲突。INSERT和COPY命令自动在目标表上获取此锁定模式（除了对任何其他引用表的ACCESS SHARE锁定外），请参见[注解](#)。
- SHARE UPDATE EXCLUSIVE — 与SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE和ACCESS EXCLUSIVE锁定模式冲突。

冲突。此模式可防止表发生并发schema更改和VACUUM运行。由VACUUM (无FULL) 在堆表和ANALYZE上获取。

- **SHARE** — 与ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE ROW EXCLUSIVE, EXCLUSIVE和ACCESS EXCLUSIVE锁定模式冲突。此模式可防止表发生并发数据更改。由CREATE INDEX自动获取。
- **SHARE ROW EXCLUSIVE** — 与ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE和ACCESS EXCLUSIVE锁定模式冲突。任何Greenplum数据库命令都不会自动获取此锁定模式。
- **EXCLUSIVE** — 与ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE和ACCESS EXCLUSIVE锁定模式冲突。此模式仅允许并发的ACCESS SHARE锁定，即，只有从表中读取的数据才能与持有该锁定模式的事务并行进行。对于Greenplum数据库中的UPDATE, SELECT FOR UPDATE和DELETE, 此锁定模式是自动获取的（与常规PostgreSQL相比，限制性更强）。看[注解](#)。
- **ACCESS EXCLUSIVE** — 与所有模式的锁定 (ACCESS SHARE, ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE和ACCESS EXCLUSIVE) 冲突。这种模式保证了持有者是唯一以任何方式访问表的事务。由ALTER TABLE, DROP TABLE, TRUNCATE, REINDEX, CLUSTER和VACUUM FULL命令自动获取。这是未明确指定模式的LOCK TABLE语句的默认锁定模式。VACUUM (无FULL) 也会在处理过程中在追加优化表上短暂获取此锁。

Note: 默认情况下，Greenplum数据库为堆表上的DELETE, UPDATE和SELECT FOR UPDATE操作获取表上的EXCLUSIVE锁。启用全局死锁检测器后，堆表上操作的锁定模式为ROW EXCLUSIVE。请参阅[全局死锁检测](#)。

NOWAIT

指定LOCK TABLE不等待任何冲突的锁被释放：如果不等待就无法立即获取指定的锁，则事务中止。

注解

LOCK TABLE ... IN ACCESS SHARE MODE需要对目标表具

有SELECT特权。所有其他形式的LOCK都需要表级UPDATE, DELETE或TRUNCATE特权。

LOCK TABLE在事务块之外是无用的：该锁定将仅在LOCK语句完成时就释放。因此，如果在事务块外部使用LOCK，Greenplum数据库将报告错误。使用BEGIN和END定义事务块。

LOCK TABLE仅处理表级锁，因此涉及ROW的模式名称都是错误的。这些模式名称通常应理解为指示用户获取锁定表中的行级锁定的意图。另外，ROW EXCLUSIVE模式是可共享的表锁。请记住，就LOCK TABLE而言，所有锁定模式都具有相同的语义，只是在哪些模式与哪个模式冲突的规则上有所不同。有关如何获取实际的行级锁的信息，请参见[SELECT](#)参考文档中的FOR UPDATE/FOR SHARE子句。

示例

在执行films_user_comments表中的插入操作时，在films表上获得SHARE锁定：

```
BEGIN WORK;
LOCK TABLE films IN SHARE MODE;
SELECT id FROM films
    WHERE name = 'Star Wars: Episode I - The Phantom
Menace';
-- Do ROLLBACK if record was not returned
INSERT INTO films_user_comments VALUES
    (_id_, 'GREAT! I was waiting for it for so long!');
COMMIT WORK;
```

执行删除操作时，对表进行SHARE ROW EXCLUSIVE锁定：

```
BEGIN WORK;
LOCK TABLE films IN SHARE ROW EXCLUSIVE MODE;
DELETE FROM films_user_comments WHERE id IN
    (SELECT id FROM films WHERE rating < 5);
DELETE FROM films WHERE rating < 5;
COMMIT WORK;
```

兼容性

SQL标准中没有LOCK TABLE，而是使用SET TRANSACTION来指定

事务的并发级别。 Greenplum数据库也支持这一点。

除了ACCESS SHARE, ACCESS EXCLUSIVE和SHARE UPDATE EXCLUSIVE锁定模式外， Greenplum数据库锁定模式和LOCK TABLE语法与Oracle中的兼容。

另见

[BEGIN](#), [SET TRANSACTION](#), [SELECT](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

定位游标。

概要

```
MOVE [ forward_direction [ FROM | IN ] ]
      cursor_name
```

其中*forward_direction*可以为空或以下之一：

NEXT
FIRST
LAST
ABSOLUTE *count*
RELATIVE *count*
count
ALL
FORWARD
FORWARD *count*
FORWARD ALL

描述

MOVE会重新定位游标，而不会检索任何数据。 MOVE的工作方式与[FETCH](#)命令完全相同，只不过它仅定位光标而不返回行。

请注意，由于不支持可滚动光标，因此无法在Greenplum数据库中向后移动光标位置。 您只能使用MOVE向前移动光标。

输出

成功完成后，MOVE命令将返回以下形式的命令标签：

```
MOVE count
```

count是具有相同参数的[FETCH](#)命令将返回的行数（可能~~可能~~）。

参数

forward_direction

MOVE命令的参数与FETCH命令的参数相同。有关语法和用法的详细信息，请参阅[FETCH](#)。

cursor_name

打开的游标的名称。

示例

-- 开始事务：

```
BEGIN;
```

-- 设置游标：

```
DECLARE mycursor CURSOR FOR SELECT * FROM films;
```

-- 在游标mycursor中向前移动5行：

```
MOVE FORWARD 5 IN mycursor;
MOVE 5
```

-- 之后获取下一行（第6行）：

```
FETCH 1 FROM mycursor;
code | title | did | date_prod | kind | len
-----+-----+-----+-----+-----+-----+
P_303 | 48 Hrs | 103 | 1982-10-22 | Action | 01:37
(1 row)
```

-- 关闭游标并结束事务：

```
CLOSE mycursor;
COMMIT;
```

兼容性

SQL标准中没有MOVE语句。

另见

[DECLARE](#) , [FETCH](#) , [CLOSE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)[ABORT](#)[ALTER AGGREGATE](#)[ALTER COLLATION](#)[ALTER CONVERSION](#)[ALTER DATABASE](#)[ALTER DEFAULT PRIVILEGES](#)[ALTER DOMAIN](#)[ALTER EXTENSION](#)[ALTER EXTERNAL TABLE](#)[ALTER FOREIGN DATA WRAPPER](#)[ALTER FOREIGN TABLE](#)[ALTER FUNCTION](#)[ALTER GROUP](#)[ALTER INDEX](#)[ALTER LANGUAGE](#)[ALTER OPERATOR](#)

准备要执行的语句。

概要

```
PREPARE name [ (datatype [, ...] ) ] AS statement
```

描述

PREPARE创建一个预编译语句。 预编译语句是可用于优化性能的服务器端对象。 当执行PREPARE语句时，将解析、分析和重写指定的语句。 随后发出EXECUTE命令时，将计划并执行预编译语句。 这种分工避免了重复的解析分析工作，同时允许执行计划取决于所提供的特定参数值。

预编译语句可以采用参数：执行语句时将用值替换。 创建预编译语句时，请使用\$1, \$2等按位置引用参数。 可以选择指定相应的参数数据类型列表。 如果未指定参数的数据类型或将其声明为未知，则从首次使用该参数的上下文中推断该类型（如果可能）。 执行该语句时，请在EXECUTE语句中指定这些参数的实际值。

预编译语句仅在当前数据库会话期间有效。 会话结束时，预编译语句将废弃，因此必须在重新使用之前重新创建它。 这也意味着单个预编译语句不能被多个同时的数据库客户端使用。 但是，每个客户端可以创建自己预编译语句来使用。 可以使用DEALLOCATE命令手动清除预编译语句。

当使用单个会话执行大量相似的语句时，预备语句具有最大的性能优势。 如果语句计划或重写很复杂，那么性能差异将特别显着，例如，查询涉及多个表的连接或需要应用多个规则。 如果该语句的计划和重写相对简单，但是执行成本相对较高，则预编译语句的性能优势将不太明显。

参数

name



为此特定的预编译语句提供的任意名称。它在单个会话中必须是唯一的，并随后用于执行或取消分配先前预编译语句。

datatype

预编译语句的参数的数据类型。如果未指定特定参数的数据类型或将其指定为未知，则将从首次使用该参数的上下文中推断出来。要在预编译语句本身中引用参数，请使用\$1, \$2等。

statement

任何SELECT, INSERT, UPDATE, DELETE或VALUES语句。

注解

如果预编译语句执行了足够的时间，则服务器可能最终决定保存并重新使用通用计划，而不是每次都重新计划。如果预编译语句没有参数，这将立即发生；否则，仅当通用计划看起来并不比依赖于特定参数值的计划昂贵得多时，它才会发生。通常，仅当估计查询的性能对提供的特定参数值相当不敏感时，才会选择通用计划。

要检查Greenplum数据库用于预编译语句的查询计划，请使用EXPLAIN。如果正在使用通用计划，它将包含参数符号\$*n*，而自定义计划将使用当前的实际参数值替代它。

有关查询计划和Greenplum数据库为此目的收集的统计信息的更多信息，请参阅ANALYZE文档。

尽管预编译语句的主要目的是避免重复分析和规划语句，但只要语句中使用的数据库对象自上次使用预处理语句以来发生了定义 (DDL) 更改，Greenplum都会在使用之前强制重新分析和重新规划语句。同样，如果search_path的值从一个更改为下一个，则将使用新的search_path重新解析该语句。（后一种行为自Greenplum 6起是新的。）这些规则使用预编译语句在语义上几乎等同于一次又一次地重新提交相同的查询文本，这在语义上几乎等同于使用预编译语句，但是如果不能更改对象定义，则可以提高性能，特别是如果最佳计划在不同用途之间保持不变。语义对等不完美的情况的一个示例是，如果该语句使用不合格的名称引用表，然后在较早出现在search_path中的模式中创建了一个具有相同名称的新表，则不会自动重新解析，因为语句中使用的对象没有变化。但是，如果其他更改迫使重新解析，则在随后的使用中将引用新表。

您可以通过查询pg_prepared_statements系统视图来查看会话中所有可用的预编译语句。

示例

为INSERT语句创建一个预编译语句，然后执行它：

```
PREPARE fooplan (int, text, bool, numeric) AS INSERT INTO
foo VALUES($1, $2, $3, $4);
EXECUTE fooplan(1, 'Hunter Valley', 't', 200.00);
```

为SELECT语句创建一个预编译语句，然后执行它。请注意，未指定第二个参数的数据类型，因此可以从使用\$2的上下文中推断出它：

```
PREPARE usrrptplan (int) AS SELECT * FROM users u, logs l
WHERE u.usrid=$1 AND u.usrid=l.usrid AND l.date = $2;
EXECUTE usrrptplan(1, current_date);
```

兼容性

SQL标准包括一个PREPARE语句，但仅用于嵌入式SQL。此版本的PREPARE语句还使用了一些不同的语法。

另见

[EXECUTE](#) , [DEALLOCATE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改数据库角色拥有的数据库对象的所有权。

概要

```
REASSIGN OWNED BY old_role [, ...] TO new_role
```

描述

REASSIGN OWNED将任何*old_role*拥有的数据库对象的所有权更改为*new_role*。

参数

old_role

角色名称。该角色所拥有的当前数据库中所有对象以及所有共享对象（数据库，表空间）的所有权将重新分配给*new_role*。

new_role

将成为受影响对象的新所有者的角色的名称。

注解

REASSIGN OWNED通常用于准备删除一个或多个角色。由于REASSIGN OWNED不会影响其他数据库中的对象，因此通常需要在每个数据库中执行此命令，该数据库包含要删除的角色所拥有的对象。

REASSIGN OWNED要求对源角色和目标角色都具有特权。

[DROP OWNED](#)命令是一种替代方法，它仅删除一个或多个角色拥有的所有数据库对象。DROP OWNED仅要求对源角色具有特权。

REASSIGN OWNED命令不会影响授予旧角色的不属于它们的对象的任何特权。使用DROP OWNED撤销那些特权。



示例

将由sally和bob角色拥有的所有数据库对象重新分配给admin;

```
REASSIGN OWNED BY sally, bob TO admin;
```

兼容性

REASSIGN OWNED命令是Greenplum数据库扩展。

另见

[DROP OWNED](#), [DROP ROLE](#), [ALTER DATABASE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

重建索引。

概要

```
REINDEX { INDEX | TABLE | DATABASE | SYSTEM } name
```

描述

REINDEX使用存储在索引表中的数据重建索引，替换索引的旧副本。在几种情况下可以使用REINDEX：

- 索引变得肿，也就是说，它包含许多空或几乎为空的页面。在某些罕见的访问模式下，Greenplum数据库中的B树索引可能会发生这种情况。 REINDEX提供了一种通过编写没有死页的新版本索引来减少索引空间消耗的方法。
- 您已经更改了索引的FILLFACTOR存储参数，并希望确保所做的更改完全生效。

参数

INDEX

重新创建指定的索引。

TABLE

重新创建指定表的所有索引。如果该表具有辅助TOAST表，则该表也将重新编制索引。

DATABASE

重新创建当前数据库内的所有索引。共享系统目录上的索引也会被处理。这种形式的REINDEX不能在事务块内执行。

SYSTEM

在当前数据库内的系统目录上重新创建所有索引。包括共享系统目录上的索引。用户表上的索引未处理。这种形式的REINDEX不能在事务块内执行。

name

要重新建立索引的特定索引，表或数据库的名称。索引和表名可以是模式限定的。当前，REINDEX DATABASE和REINDEX



SYSTEM只能重新索引当前数据库，因此它们的参数必须与当前数据库的名称匹配。

注解

REINDEX与删除索引和重新创建索引相似，因为索引内容是从头开始重建的。但是，锁定注意事项却大不相同。REINDEX锁定对索引的父表的写入但不锁定读取。它还对正在处理的特定索引采取排他锁，这将阻塞该索引的读取。相反，DROP INDEX会立即对父表加排他锁，从而阻止写入和读取。随后的CREATE INDEX锁定写但不锁定读；由于索引不存在，因此不会进行任何读取尝试，这意味着不会有阻塞，但读操作可能会强制进行高代价的顺序扫描。

重新索引单个索引或表要求是该索引或表的所有者。重新索引数据库需要成为数据库的所有者（请注意，所有者可以因此重建其他用户拥有的表的索引）。当然，超级用户可以随时重新索引任何内容。

REINDEX不会更新索引的reltuples和relpages统计信息。要更新这些统计信息，请在重新建立索引后在表上运行ANALYZE。

如果您怀疑共享的全局系统目录索引已损坏，则只能在Greenplum utility模式下对它们进行索引。共享索引损坏的典型症状是“索引不是btree”错误，否则服务器将在启动时由于依赖损坏的索引而立即崩溃。在这种情况下，请联系Greenplum客户支持。

示例

重建单个索引：

```
REINDEX INDEX my_index;
```

重建表my_table上的所有索引：

```
REINDEX TABLE my_table;
```

兼容性

SQL标准中没有REINDEX命令。

另见

[CREATE INDEX](#), [DROP INDEX](#), [VACUUM](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

销毁先前定义的保存点。

概要

RELEASE [SAVEPOINT] *savepoint_name*

描述

RELEASE SAVEPOINT销毁先前在当前事务中定义的保存点。

破坏保存点使其无法用作回滚点，但是它没有其他用户可见的行为。它不会撤消在建立保存点后执行的命令的影响。（为此，请参阅[ROLLBACK TO SAVEPOINT](#)。）在不再需要保存点时对其进行销毁，可能会使系统在事务结束之前回收一些资源。

RELEASE SAVEPOINT还将销毁在建立指定保存点之后建立的所有保存点。

参数

savepoint_name

要销毁的保存点的名称。

示例

建立并随后销毁一个保存点：

```
BEGIN;
    INSERT INTO table1 VALUES (3);
    SAVEPOINT my_savepoint;
    INSERT INTO table1 VALUES (4);
    RELEASE SAVEPOINT my_savepoint;
COMMIT;
```



上述事务将同时插入3和4。

兼容性

该命令符合SQL标准。 该标准指定关键字SAVEPOINT是必需的，但是Greenplum数据库允许省略它。

另见

[BEGIN](#) , [SAVEPOINT](#) , [ROLLBACK TO SAVEPOINT](#) , [COMMIT](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)[ABORT](#)[ALTER AGGREGATE](#)[ALTER COLLATION](#)[ALTER CONVERSION](#)[ALTER DATABASE](#)[ALTER DEFAULT PRIVILEGES](#)[ALTER DOMAIN](#)[ALTER EXTENSION](#)[ALTER EXTERNAL TABLE](#)[ALTER FOREIGN DATA WRAPPER](#)[ALTER FOREIGN TABLE](#)[ALTER FUNCTION](#)[ALTER GROUP](#)[ALTER INDEX](#)[ALTER LANGUAGE](#)[ALTER OPERATOR](#)

将系统配置参数的值恢复为默认值。

概要

```
RESET configuration_parameter
```

```
RESET ALL
```

描述

RESET将系统配置参数恢复为其默认值。 RESET是SET *configuration_parameter* TO DEFAULT的替代写法。

缺省值定义为该参数在当前会话中没有为其发出SET时将具有的值。该值的实际来源可能是已编译的默认值，主postgresql.conf配置文件，命令行选项或每个数据库或每个用户的默认设置。有关更多信息，请参见[服务器配置参数](#)。

参数

configuration_parameter

系统配置参数的名称。有关详细信息，请参见[服务器配置参数](#)。

ALL

将所有可设置的配置参数重置为其默认值。

示例

将statement_mem配置参数设置为其默认值：

```
RESET statement_mem;
```



兼容性

RESET是Greenplum数据库扩展。

另见

[SET](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

删除访问权限。

 参考指南 SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

概要

```

REVOKE [GRANT OPTION FOR] { {SELECT | INSERT | UPDATE |
DELETE
| REFERENCES | TRIGGER | TRUNCATE } [, ...] | ALL
[PRIVILEGES] }

ON { [TABLE] table_name [, ...]
| ALL TABLES IN SCHEMA schema_name [, ...] }
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT]

REVOKE [ GRANT OPTION FOR ] { { SELECT | INSERT | UPDATE
| REFERENCES } ( column_name [, ...] )
[, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] )
}

ON [ TABLE ] table_name [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]

REVOKE [GRANT OPTION FOR] { {USAGE | SELECT | UPDATE}
[,...]
| ALL [PRIVILEGES] }
ON { SEQUENCE sequence_name [, ...]
| ALL SEQUENCES IN SCHEMA schema_name [, ...] }
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { {CREATE | CONNECT
| TEMPORARY | TEMP} [, ...] | ALL [PRIVILEGES] }
ON DATABASE database_name [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT]

REVOKE [ GRANT OPTION FOR ]
{ USAGE | ALL [ PRIVILEGES ] }
ON DOMAIN domain_name [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
{ USAGE | ALL [ PRIVILEGES ] }
ON FOREIGN DATA WRAPPER fdw_name [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
{ USAGE | ALL [ PRIVILEGES ] }

```

```

        ON FOREIGN SERVER server_name [, ...]
        FROM { [ GROUP ] role_name | PUBLIC } [, ...]
        [ CASCADE | RESTRICT ]

REVOKE [GRANT OPTION FOR] {EXECUTE | ALL [PRIVILEGES]}
    ON { FUNCTION funcname ( [[argmode] [argname]
argtype
        [, ...]
        | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC} [, ...]
    [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] {USAGE | ALL [PRIVILEGES]}
    ON LANGUAGE langname [, ...]
    FROM { [ GROUP ] role_name | PUBLIC} [, ...]
    [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { {CREATE | USAGE} [, ...]
    | ALL [PRIVILEGES] }
    ON SCHEMA schema_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC} [, ...]
    [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { CREATE | ALL [PRIVILEGES] }
    ON TABLESPACE tablespacename [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [CASCADE | RESTRICT]

REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON TYPE type_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ADMIN OPTION FOR] parent_role [, ...]
    FROM [ GROUP ] member_role [, ...]
    [CASCADE | RESTRICT]

```

描述

REVOKE命令从一个或多个角色撤消先前授予的特权。关键字PUBLIC指所有角色的隐式定义组。

有关特权类型的含义，请参见[GRANT命令的描述](#)。

请注意，任何特定角色将具有直接授予它的特权，授予该角色当前成员的任何角色的特权以及授予PUBLIC的特权的总和。因此，例如，从PUBLIC撤消SELECT特权并不一定意味着所有角色都对该对象失去SELECT特权：直接或通过其他角色授予它的那些人仍然拥有该特权。同样，如果PUBLIC或其他成员身份角色仍然具有SELECT权限，

则从用户撤消SELECT可能不会阻止该用户使用SELECT。

如果指定了GRANT OPTION FOR，则仅撤销特权的授予选项，而不撤销特权本身。否则，特权和授予选项都将被撤销。

如果某个角色拥有带有“授予”选项的特权并将其授予其他角色，则这些其他角色所拥有的特权称为“从属特权”。如果第一个角色持有的特权或授予选项被撤销，并且存在从属特权，则如果指定了CASCADE，那些从属特权也将被撤销，否则撤销操作将失败。此递归撤销仅影响通过可追溯到此REVOKE命令对象的角色的角色链授予的特权。因此，如果还通过其他角色授予了特权，则受影响的角色可以有效地保留特权。

当您撤消对表的特权时，Greenplum数据库也会撤消对表的每一列的相应列特权（如果有）。另一方面，如果已授予角色某个表的特权，则从单独列中撤消相同的特权将无效。

撤销角色成员身份时，GRANT OPTION改为ADMIN OPTION，但行为类似。

参数

参见[GRANT](#)。

注解

用户只能撤消该用户直接授予的那些特权。例如，如果用户A向用户B授予了具有授予选项的特权，而用户B又将其授予了用户C，则用户A无法直接从C撤消该特权。而是，用户A可以撤消用户B的授予选项，并使用CASCADE选项，以便依次从用户C撤消特权。例如，如果A和B都向C授予相同的特权，则A可以撤消他自己的授予，但不能撤消B的授予，因此C实际上仍然有特权。

当对象的非所有者尝试REVOKE对对象的特权时，如果用户对对象没有任何特权，则该命令将彻底失败。只要有某些特权，该命令就会继续执行，但是它将仅撤销用户具有授予选项的那些特权。如果未保留任何授予选项，则REVOKE ALL PRIVILEGES形式将发出警告消息，而如果未保留针对命令中明确指定的任何特权的授予选项，则其他形式将发出警告。（原则上，这些声明也适用于对象所有者，但是由于Greenplum数据库始终将所有者视为拥有所有授予选项的对象，因此永远不会发生这种情况。）

如果超级用户选择发出GRANT或REVOKE命令，则Greenplum数据库将执行该命令，就像它是由受影响对象的所有者发出的一样。由于所有特权最终都来自对象所有者（可能间接地通过授予选项链），因此超级用户可以撤消所有特权，但是如上所述，这可能需要使用CASCADE。

REVOKE也可以由不是受影响对象的所有者，但是拥有该对象的角色的成员，或者是持有对该对象具有WITH GRANT OPTION特权的角色的成员来调用。在这种情况下，Greenplum数据库将执行该命令，就像它是由实际拥有该对象或拥有WITH GRANT OPTION特权的用户所包含的角色发出的那样。例如，如果表t1由角色g1拥有，而角色u1是它的成员，则u1可以撤销t1上记录为由g1授予的特权。这包括u1以及角色g1的其他成员所做的授权。

如果执行REVOKE的角色通过多个角色成员路径间接持有特权，则未指定将使用哪个包含角色来执行命令。在这种情况下，最佳实践是使用SET ROLE来成为要REVOKE的特定角色。否则，可能会导致撤销您想要的特权以外的特权，或者根本不撤销任何特权。

使用psql的\dp元命令来获取有关表和列的现有特权的信息。您还可以使用其他\d元命令来显示非表对象的特权。

示例

撤销public对表films的插入特权：

```
REVOKE INSERT ON films FROM PUBLIC;
```

从topten视图中的角色sally撤消所有特权。请注意，这实际上意味着撤消当前角色（如果不是超级用户）授予的所有特权。

```
REVOKE ALL PRIVILEGES ON topten FROM sally;
```

从用户joe撤消角色admins的成员资格：

```
REVOKE admins FROM joe;
```

兼容性

[GRANT](#)命令的兼容性说明也适用于REVOKE。

根据标准，需要RESTRICT或CASCADE，但是默认情况下Greenplum数据库假定RESTRICT。

另见

[GRANT](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

[SQL Syntax Summary](#)[ABORT](#)[ALTER AGGREGATE](#)[ALTER COLLATION](#)[ALTER CONVERSION](#)[ALTER DATABASE](#)[ALTER DEFAULT PRIVILEGES](#)[ALTER DOMAIN](#)[ALTER EXTENSION](#)[ALTER EXTERNAL TABLE](#)[ALTER FOREIGN DATA WRAPPER](#)[ALTER FOREIGN TABLE](#)[ALTER FUNCTION](#)[ALTER GROUP](#)[ALTER INDEX](#)[ALTER LANGUAGE](#)[ALTER OPERATOR](#)

中止当前事务。

概要

ROLLBACK [WORK | TRANSACTION]

描述

ROLLBACK回滚当前事务，并使该事务进行的所有更新都被丢弃。

参数

WORK

TRANSACTION

可选关键字。它们没有作用。

注解

使用COMMIT成功提交当前事务。

如果不在事务内部，则发出ROLLBACK不会造成任何危害，但是会引发警告消息。

示例

要舍弃当前事务中的所有更改：

ROLLBACK ;



兼容性

SQL标准仅指定两种形式ROLLBACK和ROLLBACK WORK。否则，此命令完全符合要求。

另见

[BEGIN , COMMIT , SAVEPOINT , ROLLBACK TO SAVEPOINT](#)

Parent topic: [SQL Command Reference](#)

SAVEPOINT

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

将当前事务回滚到保存点。

概要

```
ROLLBACK [ WORK | TRANSACTION ] TO [ SAVEPOINT ]
          savepoint_name
```

描述

此命令将回滚在建立保存点之后执行的所有命令。该保存点保持有效，可以稍后在需要时回滚。

ROLLBACK TO SAVEPOINT隐式销毁在指定保存点之后建立的所有保存点。

参数

WORK

TRANSACTION

可选关键字。它们没有作用。

savepoint_name

要回滚的保存点的名称。

注解

使用RELEASE SAVEPOINT销毁一个保存点，而不会丢弃建立该保存点后执行的命令的效果。

指定尚未建立的保存点名称是错误的。

游标相对于保存点具有某种非事务性的行为。当回滚保存点时，在保



存点内部打开的所有游标都将关闭。如果先前打开的游标受保存点内的FETCH命令影响，该保存点随后被回滚，则该游标仍将保持在FETCH指向的位置（即，由FETCH引起的光标运动不会回滚）。关闭游标也不会通过回滚来撤消。但是，如果游标的查询所引起的其他副作用（例如查询所调用的volatile函数的副作用）在保存点期间发生，然后又回滚，则这些副作用也会被回滚。由于其执行导致事务中止的游标处于无法执行状态，因此尽管可以使用ROLLBACK TO SAVEPOINT还原该事务，但不能再使用该游标。

示例

要撤消在建立my_savepoint之后执行的命令的影响：

```
ROLLBACK TO SAVEPOINT my_savepoint;
```

光标位置不受保存点回滚的影响：

```
BEGIN;
DECLARE foo CURSOR FOR SELECT 1 UNION SELECT 2;
SAVEPOINT foo;
FETCH 1 FROM foo;
column
-----
1
ROLLBACK TO SAVEPOINT foo;
FETCH 1 FROM foo;
column
-----
2
COMMIT;
```

兼容性

SQL标准指定关键字SAVEPOINT是必需的，但是Greenplum数据库（和Oracle）允许省略它。SQL在ROLLBACK之后只允许WORK，而不允许TRANSACTION作为干扰词。此外，SQL具有可选子句AND [NO] CHAIN，Greenplum数据库当前不支持孩子句。在其他方面，此命令符合SQL标准。

另见

[BEGIN](#) , [COMMIT](#) , [SAVEPOINT](#) , [RELEASE SAVEPOINT](#) , [ROLLBACK](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

在当前事务中定义一个新的保存点。

概要

SAVEPOINT *savepoint_name*

描述

SAVEPOINT 在当前事务中建立一个新的保存点。

保存点是事务内部的特殊标记，它允许回滚在建立事务后执行的所有命令，从而将事务状态恢复到保存点时的状态。

参数

savepoint_name

新保存点的名称。

注解

使用 `ROLLBACK TO SAVEPOINT` 回滚到保存点。使用 `RELEASE SAVEPOINT` 销毁一个保存点，并在建立保存点后保持命令执行的效果。

只能在事务块内部建立保存点。事务中可以定义多个保存点。

示例

要建立一个保存点，然后撤消建立保存点后执行的所有 的效果：

```
BEGIN;
    INSERT INTO table1 VALUES (1);
```

```
SAVEPOINT my_savepoint;
INSERT INTO table1 VALUES (2);
ROLLBACK TO SAVEPOINT my_savepoint;
INSERT INTO table1 VALUES (3);

COMMIT;
```

上面的事务将插入值1和3，但不插入2。

要建立并随后销毁一个保存点：

```
BEGIN;

        INSERT INTO table1 VALUES (3);
        SAVEPOINT my_savepoint;
        INSERT INTO table1 VALUES (4);
        RELEASE SAVEPOINT my_savepoint;

COMMIT;
```

上述事务将同时插入3和4。

兼容性

建立另一个具有相同名称的保存点时，SQL要求自动删除一个保存点。在Greenplum数据库中，保留了旧的保存点，尽管在回滚或释放时仅使用较新的保存点。（释放较新的保存点将使较旧的保存点再次可用于`ROLLBACK TO SAVEPOINT`和`RELEASE SAVEPOINT`。）否则，`SAVEPOINT`完全符合SQL。

另见

[BEGIN , COMMIT , ROLLBACK , RELEASE SAVEPOINT , ROLLBACK TO SAVEPOINT](#)

[Parent topic: SQL Command Reference](#)

从表或视图中检索行。

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

[SQL Syntax Summary](#)

[ABORT](#)

[ALTER AGGREGATE](#)

[ALTER COLLATION](#)

[ALTER CONVERSION](#)

[ALTER DATABASE](#)

[ALTER DEFAULT PRIVILEGES](#)

[ALTER DOMAIN](#)

[ALTER EXTENSION](#)

[ALTER EXTERNAL TABLE](#)

[ALTER FOREIGN DATA WRAPPER](#)

[ALTER FOREIGN TABLE](#)

[ALTER FUNCTION](#)

[ALTER GROUP](#)

[ALTER INDEX](#)

[ALTER LANGUAGE](#)

[ALTER OPERATOR](#)

概要

```
[ WITH [ RECURSIVE1 ] with_query [, ...] ]
SELECT [ALL | DISTINCT [ON (expression [, ...])]]
      * | expression[[AS] output_name] [, ...]
      [FROM from_item [, ...]]
      [WHERE condition]
      [GROUP BY grouping_element [, ...]]
      [HAVING condition [, ...]]
      [WINDOW window_name AS (window_definition) [, ...]]
      [{UNION | INTERSECT | EXCEPT} [ALL | DISTINCT] select]
      [ORDER BY expression [ASC | DESC | USING operator] [NULLS
{FIRST | LAST}] [, ...]]
      [LIMIT {count | ALL}]
      [OFFSET start [ ROW | ROWS ] ]
      [FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY]
      [FOR {UPDATE | NO KEY UPDATE | SHARE | KEY SHARE} [OF
table_name [, ...]] [NOWAIT] [...]]
TABLE { [ ONLY ] table_name [ * ] | with_query_name }
```

其中*with_query*是：

```
with_query_name [ ( column_name [, ...] ) ] AS ( select | values |
insert | update | delete )
```

其中*from_item*可以是以下之一：

```
[ONLY] table_name [ * ] [ [ AS ] alias [ ( column_alias [, ...] ) ]
]
( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
function_name ( [ argument [, ...] ] )
      [ WITH ORDINALITY ] [ [ AS ] alias [ ( column_alias
[, ...] ) ] ]
function_name ( [ argument [, ...] ] ) [ AS ] alias (
column_definition [, ...] )
function_name ( [ argument [, ...] ] ) AS ( column_definition [, ...
...]
)
ROWS FROM( function_name ( [ argument [, ...] ] ) [ AS (
column_definition [, ...] ) ] [, ...] )
      [ WITH ORDINALITY ] [ [ AS ] alias [ ( column_alias
[, ...] ) ] ]
from_item [ NATURAL ] join_type from_item
      [ ON join_condition | USING ( join_column [, ...] ) ]
]
```

其中*grouping_element*可以是以下之一：

```
( )
expression
```

```
ROLLUP (expression [, ...])
CUBE (expression [, ...])
GROUPING SETS ((grouping_element [, ...]))
```

其中*window_definition*是：

```
[existing_window_name]
[PARTITION BY expression [, ...]]
[ORDER BY expression [ASC | DESC | USING operator]
[NULLS {FIRST | LAST}] [, ...]]
[ { RANGE | ROWS} frame_start
  | {RANGE | ROWS} BETWEEN frame_start AND frame_end]
```

其中*frame_start*和*frame_end*可以是以下之一：

```
UNBOUNDED PRECEDING
value
PRECEDING
CURRENT ROW
value
FOLLOWING
UNBOUNDED FOLLOWING
```

注解：¹RECURSIVE关键字是一个Beta功能。

描述

SELECT从零个或多个表中检索行。 SELECT的一般处理如下：

1. WITH子句中的所有查询均被计算。这些可以有效地用作可以在FROM列表中引用的临时表。
2. 将计算FROM列表中的所有元素。（FROM列表中的每个元素都是真实表或虚拟表。）如果FROM列表中指定了多个元素，则它们将交叉连接在一起。
3. 如果指定了WHERE子句，则从输出中排除所有不满足该条件的行。
4. 如果指定了GROUP BY子句，或者存在聚合函数调用，则将输出组合为与一个或多个值匹配的行组，并计算聚合函数的结果。如果存在HAVING子句，它将排除不满足给定条件的组。
5. 实际的输出行是使用SELECT输出表达式为每个选定的行或行组计算的。
6. SELECT DISTINCT从结果中消除重复的行。SELECT DISTINCT ON消除与所有指定表达式匹配的行。SELECT ALL（默认）将返回所有候选行，包括重复项。
7. 如果指定了窗口表达式（和可选的WINDOW子句），则根据位置（行）或基于值的（范围）窗口框架来组织输出。
8. 使用每个选定行的SELECT输出表达式来计算实际的输出行。
9. 使用运算符UNION, INTERSECT和EXCEPT，可以组合多个SELECT语句的UNION

输出以形成单个结果集。 运算符返回一个或两个结果集中的所有行。 `INTERSECT`运算符返回完全位于两个结果集中的所有行。 `EXCEPT`运算符返回第一个结果集中的行，而不是第二个结果集中的行。 在所有三种情况下，除非指定了`ALL`，否则将消除重复的行。 可以添加噪声字`DISTINCT`来明确指定消除重复行。 请注意，`DISTINCT`是此处的默认行为，即使`ALL`是`SELECT`本身默认行为。

10. 如果指定了`ORDER BY`子句，则返回的行将以指定的顺序排序。如果未给出`ORDER BY`，则以系统最先找到的顺序返回行。
11. 如果指定了`LIMIT`（或`FETCH FIRST`）或`OFFSET`子句，则`SELECT`语句仅返回结果行的子集。
12. 如果指定了`FOR UPDATE`, `FOR NO KEY UPDATE`, `FOR SHARE`或`FOR KEY SHARE`，则`SELECT`语句将锁定整个表以防止并发更新。

您必须对`SELECT`命令中使用的每一列都具有`SELECT`特权。 使用`FOR NO KEY UPDATE`, `FOR UPDATE`, `FOR SHARE`或`FOR KEY SHARE`也需要`UPDATE`特权（对于这样选择的每个表的至少一列）。

参数

`WITH`子句

可选的`WITH`子句允许您指定一个或多个子查询，这些子查询可以在主查询中按名称引用。 子查询在主查询期间有效地充当临时表或视图。 每个子查询可以是`SELECT`, `INSERT`, `UPDATE`或`DELETE`语句。 在用`WITH`编写数据修改语句（`INSERT`, `UPDATE`或`DELETE`）时，通常会包含`RETURNING`子句。 它是`RETURNING`的输出，而不是该语句修改的基础表，形成了由主查询读取的临时表。 如果省略`RETURNING` 则该语句仍将执行，但是不会产生任何输出，因此主查询无法将其引用为表。

对于包含`WITH`子句的`SELECT`命令，该子句最多可以包含一个用于修改表数据的子句（`INSERT`, `UPDATE`或`DELETE`命令）。

必须为`WITH`子句中的每个查询指定一个无schema限定的`with_query_name`。
(可选) 可以指定列名列表；如果省略列名列表，则从子查询中推断出名称。 主查询和`WITH`查询都(理论上)同时执行。

如果指定了`RECURSIVE`，则它允许`SELECT`子查询按名称引用自己。这样的子查询具有一般形式

```
non_recursive_term UNION [ALL | DISTINCT] recursive_term
```

递归自引用出现在`UNION`的右侧。 每个查询仅允许一个递归自引用。 不支持递归数据修改语句，但是您可以在数据修改语句中使用递归`SELECT`查询的结果。

如果指定了`RECURSIVE`关键字，则无需对`WITH`查询进行排序：一个查询可以引用列表中后面的另一个查询。但是，不支持循环引用或相互递归。

如果没有`RECURSIVE`关键字，则`WITH`查询只能引用`WITH`列表中较早的同

级WITH查询。

WITH RECURSIVE限制。不支持这些项目：

- 一个递归WITH子句，在`recursive_term`中包含以下内容。

- 具有自引用的子查询
- `DISTINCT`子句
- `GROUP BY`子句
- 窗口函数

- 递归WITH子句，其中`with_query_name`是集合操作的一部分。

以下是集合操作限制的示例。该查询返回错误，因为集合操作`UNION`包含对表`foo`的引用。

```
WITH RECURSIVE foo(i) AS (
    SELECT 1
    UNION ALL
    SELECT i+1 FROM (SELECT * FROM foo UNION SELECT 0) bar
)
SELECT * FROM foo LIMIT 5;
```

由于集合操作`UNION`没有对CTE `foo`的引用，因此允许此递归CTE。

```
WITH RECURSIVE foo(i) AS (
    SELECT 1
    UNION ALL
    SELECT i+1 FROM (SELECT * FROM bar UNION SELECT 0) bar, foo
    WHERE foo.i = bar.a
)
SELECT * FROM foo LIMIT 5;
```

WITH查询的一个关键属性是，即使主查询多次引用它们，一次执行主查询也只会对它们进行一次评估。特别是，无论主查询是读取全部输出还是输出任何内容，都保证数据修改语句仅执行一次。

主查询和WITH查询都（理论上）同时执行。这意味着，除了通过读取其`RETURNING`输出之外，不能从查询的其他部分看到WITH中的数据修改语句的效果。如果两个这样的数据修改语句试图修改同一行，则结果不确定。

有关更多信息，请参见[WITH查询 \(公用表表达式\)](#)。[WITH查询 \(公用表表达式\)](#)...

SELECT列表

SELECT列表（在关键字`SELECT`和`FROM`之间）指定用于形成SELECT语句输出行的表达式。表达式可以（通常）引用`FROM`子句中计算的列。

SELECT列表中的`expression`可以是常量值、列引用、运算符调用、函数调用、聚合表达式、窗口表达式、标量子查询等。可以将许多结构体归类为表达式，但不遵循任何常规语法规则。这些通常具有函数或运算符的语义。有关SQL值表达式和函数调用的信息，请参阅Greenplum数据库管理员指南中的“[查询数据](#)”。

就像在表中一样，SELECT的每个输出列都有一个名称。在简单的SELECT中，

此名称仅用于标记要显示的列，但是当SELECT是较大查询的子查询时，该名称在较大查询中被视为由该子查询生成的虚拟表的列名。要指定用于输出列的名称，请在该列的表达式之后写入AS *output_name*。（您可以省略AS，但只有在所需的输出名称与任何SQL关键字都不匹配时才可以使用。为防止将来可能再添加关键字，您始终可以写AS或在输出名称中用双引号。）如果不指定列名称，Greenplum数据库自动选择一个名称。如果列的表达式是简单的列引用，则所选名称与该列的名称相同。在更复杂的情况下，可以使用函数或类型名称，或者系统可能会依赖生成的名称（例如?column?或column *N*）。

输出列的名称可用于引用ORDER BY和GROUP BY子句中的列的值，但不能用于WHERE或HAVING子句中。在那里，您必须写出表达式。

可以将*而不是表达式写到输出列表，作为所选行的所有列的简写。另外，您可以编写.*作为仅来自该表的列的简写。在这种情况下，无法使用AS指定新名称。输出列名称将与表列名称相同。

DISTINCT子句

如果指定了SELECT DISTINCT，则将从结果集中删除所有重复的行（每组重复项中保留一行）。SELECT ALL则相反：所有行都保留；这是默认值。

`SELECT DISTINCT ON (expression [, ...])` 仅保留给定表达式等于的每组行的第一行。使用与ORDER BY相同的规则来解释DISTINCT ON 表达式（请参见上文）。请注意，除非使用ORDER BY来确保所需的行最先出现，否则每个集合的“第一行”都是不可预测的。例如：

```
SELECT DISTINCT ON (location) location, time, report
  FROM weather_reports
 ORDER BY location, time DESC;
```

检索每个位置的最新天气报告。但是，如果我们没有使用ORDER BY强制每个位置的时间值按降序排列，那么我们将从每个位置不可预测的时间获得报告。

DISTINCT ON 表达式必须与最左边的ORDER BY表达式匹配。ORDER BY子句通常将包含附加表达式，这些表达式确定每个DISTINCT ON 组中所需的行优先级。

FROM子句

FROM子句为SELECT指定一个或多个源表。如果指定了多个源，则结果为所有源的笛卡尔乘积（交叉连接）。但通常会添加限定条件（通过WHERE），以将返回的行限制为笛卡尔乘积的一小部分。FROM子句可以包含以下元素：

table_name

现有表或视图的名称（可以有schema修饰）。如果指定ONLY，则仅扫描该表。如果未指定ONLY，则扫描该表及其所有子表（如果有）。

alias

包含别名的FROM项目的替代名称。别名用于简洁起见或用于消除自连接的歧义（多次扫描同一张表）。提供别名后，它将完全隐藏表或函数的实际名称。例如，在给定FROM foo AS f的情况下，SELECT的其余部分必须将此FROM项目引用为f而不是foo。如果写入了别名，则还可以写入列别名列表以为表的一个或多个列提供替代名称。

select

子SELECT可以出现在FROM子句中。这就像在单个SELECT命令期间将其输出创建为临时表一样。请注意，子SELECT必须用括号括起来，并且必须为其提供别名。[VALUES](#)命令也可以在这里使用。有关在Greenplum数据库中使用相关子选择的限制，请参阅[兼容性](#)部分中的“非标准子句”。

with_query_name

通过指定其*with_query_name*在FROM子句中引用*with_query*，就像该名称是表名一样。*with_query_name*不能包含schema限定符。可以使用与表相同的方式提供别名。

*with_query*隐藏用于主查询目的的同名表。如有必要，可以通过使用schema限定表名来引用相同名称的表。

function_name

函数调用可以出现在FROM子句中。（这对于返回结果集的函数特别有用，但是可以使用任何函数。）这就像在单个SELECT命令期间将其输出创建为临时表一样。也可以使用别名。如果编写了别名，则还可以编写列别名列表，以为函数的复合返回类型的一个或多个属性提供替代名称。如果已将函数定义为返回记录数据类型，则必须存在别名或关键字AS，然后是形式为(column_name data_type [, ...])的列定义列表。列定义列表必须匹配该函数返回的实际列数和类型。

join_type

以下之一：

- **[INNER] JOIN**
- **LEFT [OUTER] JOIN**
- **RIGHT [OUTER] JOIN**
- **FULL [OUTER] JOIN**
- **CROSS JOIN**

对于INNER和OUTER连接类型，必须指定一个连接条件，即恰好是NATURAL ON *join_condition*或USING(*join_column*[, ...])之一。含义如下。对于CROSS JOIN，这些子句都不会出现。

JOIN子句结合了两个FROM项，为方便起见，我们将其称为“表”，尽管实际上它们可以是任何类型的FROM项。如有必要，请使用括号确定嵌套顺序。在没有括号的情况下，JOIN从左到右嵌套。无论如何，JOIN的绑定比逗号分隔FROM列表项的绑定更紧密。

CROSS JOIN 和 INNER JOIN 产生一个简单的笛卡尔积，与在FROM的顶层列出两个表所获得的结果相同，但受连接条件（如果有）的限制。

CROSS JOIN 等效于INNER JOIN ON (TRUE)，即，没有行被限定删除。这些连接类型只是一种符号上的方便，因为它们无法执行普通FROM和WHERE无法完成的工作。

LEFT OUTER JOIN 返回合格的笛卡尔乘积中的所有行（即，所有通过其连接条件的组合行），加上左表中没有通过右表连接条件的每行的一个副本。通过为右侧列插入空值，此左侧行将扩展为连接表的整个宽度。请注意，在确定哪些行具有匹配项时，仅考虑JOIN子句自身的条件。之后应用外部条件。

相反，RIGHT OUTER JOIN 返回所有已连接的行，并为每个不匹配的右手指上一行（在左边扩展为空）。这只是一个符号上的便利，因为您可以切换左右表将其转换为LEFT OUTER JOIN。

FULL OUTER JOIN 返回所有连接的行，再加上一行，用于每行不匹配的左表行（在右边扩展为空），再加上每行不匹配的右表行（扩展在左边为空）。

ON join_condition

*join_condition*是一个表达式，其结果为boolean 类型的值（类似于WHERE子句），该值指定连接中的哪些行被视为匹配。

USING (*join_column* [, ...])

USING (a, b, ...) 形式的子句是ON left_table.a = right_table.a AND left_table.b = right_table.b ...的简写。同样，USING意味着连接输出中将仅包括每对等效列中的一对，而不是两者。

NATURAL

NATURAL是USING列表的简写形式，该列表提到两个表中具有相同名称的所有列。如果没有公用的列名，则NATURAL等效于ON TRUE。

WHERE子句

可选的WHERE子句具有一般形式：

```
WHERE condition
```

*condition*是任何计算结果为boolean 型结果的表达式。任何不满足此条件的行将从输出中删除。如果用一行的实际值替换其中的变量引用后，该表达式返回真，则该行符合条件。

GROUP BY子句

可选的GROUP BY子句具有一般形式：

```
GROUP BY grouping_element[, ...]
```

其中*grouping_element*可以是以下之一：

```
()  
expression  
ROLLUP (expression [,....])  
CUBE (expression [,....])  
GROUPING SETS ((grouping_element [, ...]))
```

GROUP BY将所有共享有相同值的分组表达式的所有选定行压缩为单行。

*expression*可以是输入列名称，也可以是输出列（SELECT列表项）的名称或序号，或者是由输入列值组成的任意表达式。如有歧义，GROUP BY名称将被解释为输入列名称，而不是输出列名称。

汇总函数（如果有的话）在组成每个组的所有行中进行计算，从而为每个组生成单独的值。（如果有聚合函数但没有GROUP BY子句，则该查询被视为具有包含所有选定行的单个组。）可以通过将FILTER 子句附加到聚合函数调用来进一步过滤提供给每个聚合函数的行集。如果存在FILTER 子句，则只有与之匹配的那些行才包含在该聚合函数的输入中。

如果存在GROUP BY或任何聚合函数，则SELECT列表表达式不能引用未分组的列，除非是在聚合函数内，或者当未分组的列在函数上依赖于分组的列时，否则对于未分组的列将会返回不止一个的可能的值。如果分组的列（或其子集）是包含未分组的列的表的主键，则存在函数依赖。

请记住，在评估HAVING子句或SELECT列表中的任何“标量”表达式之前，将评估所有聚合函数。这意味着，例如，CASE表达式不能用于跳过对聚合函数的求值；见_____。

表达式计算规则

Greenplum数据库具有以下附加的OLAP分组扩展（通常称为`supergroups`）：

ROLLUP

`ROLLUP`分组是对`GROUP BY`子句的扩展，该子句创建聚合部分和，该部分和从最详细的级别汇总到总计，并遵循分组列（或表达式）的列表。`ROLLUP`提取分组列的有序列表，计算在`GROUP BY`子句中指定的标准聚合值，然后逐步创建更高级别的部分和，从列表的右向左移动。最后，它创建了一个总计。可以将`ROLLUP`分组视为一系列分组集。例如：

```
GROUP BY ROLLUP (a,b,c)
```

等效于：

```
GROUP BY GROUPING SETS( (a,b,c), (a,b), (a), () )
```

请注意，`ROLLUP`的 n 个元素转换为 $n+1$ 个分组集。同样，在`ROLLUP`中指定分组表达式的顺序也很重要。

CUBE

`CUBE`分组是对`GROUP BY`子句的扩展，它为分组列（或表达式）的给定列表的所有可能组合创建部分和。在多维分析方面，`CUBE`生成可以为具有指定维的多维数据集计算的所有部分和。例如：

```
GROUP BY CUBE (a,b,c)
```

等效于：

```
GROUP BY GROUPING SETS( (a,b,c), (a,b), (a,c), (b,c), (a), (b), (c), () )
```

注意，`CUBE`的 n 个元素转换为 $2n$ 个分组集。考虑在需要交叉表报告的任何情况下使用`CUBE`。`CUBE`通常最适合使用多个维度列而不是表示单个维度不同级别的列的查询。例如，通常要求的交叉列表可能需要月份，州和产品的所有组合的部分和。

GROUPING SETS

您可以使用`GROUP BY`子句中的`GROUPING SETS`表达式有选择地指定要创建的组集。这样就可以在多个维度上进行精确指定，而无需计算整个`ROLLUP`或`CUBE`。例如：

```
GROUP BY GROUPING SETS( (a,c), (a,b) )
```

如果使用分组扩展子句`ROLLUP`、`CUBE`或`GROUPING SETS`，则要面临两个问题。首先，如何确定哪些结果行是部分和，然后确定给定部分和的确切聚合级别。或者，如何区分包含存储的`NULL`值和由`ROLLUP`或`CUBE`创建的“`NULL`”值的结果行。其次，当在`GROUP BY`子句中指定重复的分组集时，如何确定哪些结果行是重复的？您可以在`SELECT`列表中使用两个附加的分组函数来帮助解决此问题：

- **grouping(column [, ...])** — 可以将`grouping`函数应用于一个或多个分组属性，以将超聚合的行与常规的分组的行区分开。这有助于将代表超级汇总行中所有值的集合的“`NULL`”与常规行中的`NULL`值区分开。此函数中的每个参数都产生一个位 - 1或0，其中1表示结果行是

超级聚合的，而0表示结果行来自常规分组。`grouping`函数通过将这些位视为二进制数然后将其转换为以10为基的整数来返回整数。

- **group_id()** — 对于包含重复分组集的分组扩展查询，`group_id`函数用于标识输出中的重复行。所有唯一分组集输出的唯一行的`group_id`值为0。对于每个检测到的重复分组集，`group_id`函数分配的`group_id`编号大于0。特定重复分组集中的所有输出行均由相同的`group_id`编号标识。

WINDOW子句

可选的WINDOW句指定出现在查询的SELECT列表或ORDER BY子句中的窗口函数的行为。这些函数可以在其OVER子句中按名称引用WINDOW句条目。但是，WINDOW句条目不必在任何地方引用。如果查询中未使用它，则将其忽略。可以根本不使用任何WINDOW句来使用窗口函数，因为窗口函数调用可以直接在其OVER子句中指定其窗口定义。但是，当多个窗口函数需要相同的窗口定义时，WINDOW句将保存键入内容。

例如：

```
SELECT vendor, rank() OVER (mywindow) FROM sale
GROUP BY vendor
WINDOW mywindow AS (ORDER BY sum(prc*qty));
```

WINDOW句具有以下一般形式：

```
WINDOW window_name AS (window_definition)
```

其中*window_name*是可以从OVER子句或后续窗口定义中引用的名称，而*window_definition*为：

```
[existing_window_name]
[PARTITION BY expression [, ...]]
[ORDER BY expression [ASC | DESC | USING operator] [NULLS {FIRST | LAST}] [, ...]]
[frame_clause]
```

existing_window_name

如果指定了*existing_window_name*，则它必须引用WINDOW句表中的较早条目；新窗口将复制该条目的分区子句以及其排序子句（如果有）。新窗口不能指定自己的PARTITION BY子句，只有复制的窗口没有孩子句时，它才能指定ORDER BY。新窗口始终使用其自己的frame子句；复制的窗口不得指定frame子句。

PARTITION BY

PARTITION BY子句根据指定表达式的唯一值将结果集组织为逻辑组。

PARTITION BY子句的元素的解释方式与GROUP BY子句的元素几乎相同，不同之处在于它们始终是简单表达式，而不是输出列的名称或编号。另一个区别是这些表达式可以包含聚合函数调用，这在常规GROUP BY子句中是不允许的。在此处允许使用它们，因为窗口在分组和聚合之后发生。与窗口函数一起使用时，这些函数将分别应用于每个分区。例如，如果在PARTITION BY之后加上列名，则结果集将按该列的不同值进行分区。如果省略，则将整个结果集视为一个分区。

同样，ORDER BY列表中的元素的解释方式与ORDER BY子句中的元素几

乎相同，不同之处在于，始终将表达式视为简单表达式，而不使用输出列的名称或编号。

ORDER BY

ORDER BY子句的元素定义如何对结果集的每个分区中的行进行排序。

如果省略，则以最有效的顺序返回行，并且行可能会有所不同。注

意：缺少一致顺序（例如time）的数据类型的列不是在窗口规范

的ORDER BY子句中使用的良好候选对象。带有或不带有时区的时间缺乏连贯的排序，因为加法和减法没有预期的效果。例如，以下条件通常不成立：`x::time < x::time + '2 hour'::interval`

frame_clause

可选的*frame_clause*定义依赖于框架的窗口函数的窗口框架（并非全部如此）。窗口框架是查询的每一行（称为当前行）的一组相关行。

*frame_clause*可以是以下之一

```
{ RANGE | ROWS } frame_start
{ RANGE | ROWS } BETWEEN frame_start AND frame_end
```

其中*frame_start*和*frame_end*可以是以下之一

- UNBOUNDED PRECEDING
- *value* PRECEDING
- CURRENT ROW
- *value* FOLLOWING
- UNBOUNDED FOLLOWING

如果省略*frame_end*，则默认为CURRENT ROW 限制条件

是*frame_start*不能为UNBOUNDED FOLLOWING *frame_end*不能为UNBOUNDED PRECEDING 并且*frame_end*选择不能在上述列表中出现在*frame_start*选择之前—例如，RANGE BETWEEN CURRENT ROW AND *value* PRECEDING是不允许的。

默认的框架选项是RANGE UNBOUNDED PRECEDING与RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW之间的范围相同；它将框架设置为从分区开始到当前行的最后一个对等方的所有行（ORDER BY的行认为与当前行等效，如果没有ORDER BY则为所有行）。通常，UNBOUNDED PRECEDING表示框架以分区的第一行开始，类似地UNBOUNDED FOLLOWING表示框架以分区的最后一行结束（与RANGE模式无关）。在ROW模式下，CURRENT ROW表示框架以当前行开始或结束；但是在RANGE模式下，这意味着框架以ORDER BY顺序中当前行的第一个或最后一个对等点开始或结束。当前仅在ROW模式下允许*value* PRECEDING和*value* FOLLOWING情况。它们指示框架以当前行之前或之后许多行的行开始或结束。*value*必须是不包含任何变量，聚合函数或窗口函数的整数表达式。该值不能为null或负数；但它可以为零，从而选择当前行本身。

请注意，如果ORDER BY排序不能唯一地对行进行排序，则ROW选项可能会产生不可预测的结果。RANGE选项旨在确保以相同的方式对待在ORDER BY顺序中为对等的行；所有对等行将在同一框架中。

使用ROWS或RANGE子句来表示窗口的边界。窗口边界可以是一个分区的一行，多行或所有行。您可以用偏移当前行值的数据值范围（RANGE）或偏移当前行的行数（ROWS）来表示窗口的边界。使用RANGE子句时，还必须使用ORDER BY子句。这是因为生成窗口而执行的计算需要对值进行排序。此外，ORDER BY子句不能包含多个表达式，并且该表达

式必须导致日期或数字值。使用ROW或RANGE子句时，如果仅指定起始行，则当前行将用作窗口中的最后一行。

PRECEDING— PRECEDING子句使用当前行作为参考点定义窗口的第一行。起始行以当前行之前的行数表示。例如，对于ROW框架，5 PRECEDING设置窗口从当前行之前的第五行开始。对于RANGE框架，它将窗口设置为从第一行开始，其排序列值在给定顺序比当前行的顺序高⁵。如果指定的顺序按日期升序，则它将是当前行之前5天内的第一行。UNBOUNDED PRECEDING将窗口中的第一行设置为分区中的第一行。

BETWEEN— BETWEEN子句使用当前行作为参考点定义窗口的第一行和最后一行。第一行和最后一行分别用当前行之前和之后的行数表示。例如，BETWEEN 3 PRECEDING AND 5 FOLLOWING将窗口设置为从当前行之前的第三行开始，到当前行之后的第五行结束。使用BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING，可以将窗口中的第一行和最后一行分别设置为分区的第一行和最后一行。如果未指定ROW或RANGE子句，则等效于默认行为。

FOLLOWING— FOLLOWING子句使用当前行作为参考点定义窗口的最后一行。最后一行以当前行之后的行数表示。例如，对于ROW框架，5 FOLLOWING将窗口设置为在当前行之后的第五行结束。在RANGE框架的情况下，它将窗口设置为最后一行的末尾，其排序列值在给定顺序的当前行后5个。如果指定的顺序按日期升序，则它将是当前行之后5天内的最后一行。使用UNBOUNDED FOLLOWING将窗口中的最后一行设置为分区中的最后一行。

如果未指定ROW或RANGE子句，并且使用ORDER BY，则窗口绑定从分区的第一行（UNBOUNDED PRECEDING）开始，到当前行（CURRENT ROW）结束。如果未指定ORDER BY，则窗口从分区的第一行开始（UNBOUNDED PRECEDING），然后从分区的最后一行结束（UNBOUNDED FOLLOWING）。

HAVING子句

可选的HAVING子句具有以下一般形式：

```
HAVING condition
```

其中*condition*与为WHERE子句指定的条件相同。HAVING去除不满足条件的组行。HAVING与WHERE不同：WHERE在应用GROUP BY之前过滤单个行，而HAVING过滤GROUP BY创建的组行。*condition*中引用的每个列都必须明确引用一个分组列，除非该引用出现在聚合函数中或未分组的列在函数上依赖于分组列。

即使没有GROUP BY子句，HAVING的存在也会将查询转换为分组查询。这与查询包含聚合函数但不包含GROUP BY子句的情况相同。所有选定的行都被视为形成一个单一的组，并且SELECT列表和HAVING子句只能引用聚合函数中的表列。如果HAVING条件为true，则此类查询将返回单行；如果条件不是true，则将返回零行。

UNION子句

UNION子句具有以下一般形式：

```
select_statement UNION [ALL | DISTINCT] select_statement
```

其中*select_statement*是不带ORDER BY, LIMIT, FOR NO KEY UPDATE, FOR UPDATE, FOR SHARE或FOR KEY SHARE子句的任何SELECT语句。（如果将ORDER BY和LIMIT括在圆括号中，则可以将其附加到子查询表达式。不带圆括号，这些子句将应用于UNION的结果，而不是其右侧输入表达式。）

UNION运算符计算所涉及的SELECT语句返回的行的集合并集。如果行出现在至少一个结果集中，则在两个结果集中的行并集中。表示UNION的直接操作数的两个SELECT语句必须产生相同数量的列，并且对应的列必须具有兼容的数据类型。

除非指定了ALL选项，否则UNION的结果不包含任何重复的行。ALL防止重复项的消除。（因此，UNION ALL通常比UNION快得多；请尽可能使用ALL。）可以编写DISTINCT来明确指定消除重复行的默认行为。

除非括号中另有说明，否则同一SELECT语句中的多个UNION运算符从左到右求值。

当前，不能为UNION结果或UNION的任何输入指定FOR NO KEY UPDATE, FOR UPDATE, FOR SHARE和FOR KEY SHARE。

INTERSECT子句

INTERSECT子句具有以下一般形式：

```
select_statement INTERSECT [ALL | DISTINCT] select_statement
```

其中*select_statement*是不带ORDER BY, LIMIT, FOR NO KEY UPDATE, FOR UPDATE, FOR SHARE或FOR KEY SHARE子句的任何SELECT语句。

INTERSECT运算符计算所涉及的SELECT语句返回的行的交集。如果一行出现在两个结果集中，则该行位于两个结果集中的交集。

除非指定ALL选项，否则INTERSECT的结果不包含任何重复的行。使用ALL，在左表中具有 m 个重复项目在右表中具有 n 个重复项的行将在结果集中出现 $\min(m, n)$ 次。可以编写DISTINCT来明确指定消除重复行的默认行为。

除非括号中另有规定，否则同一SELECT语句中的多个INTERSECT运算符从左到右求值。INTERSECT的优先级比UNION高。即，A UNION B INTERSECT C等同于A UNION (B INTERSECT C)。

当前，不能为INTERSECT结果或INTERSECT的任何输入指定FOR NO KEY UPDATE, FOR UPDATE, FOR SHARE和FOR KEY SHARE。

EXCEPT子句

EXCEPT子句具有以下一般形式：

```
select_statement EXCEPT [ALL | DISTINCT] select_statement
```

其中*select_statement*是不带ORDER BY, LIMIT, FOR NO KEY UPDATE, FOR UPDATE, FOR SHARE或FOR KEY SHARE子句的任何SELECT语句。

EXCEPT运算符计算在左SELECT语句的结果但不在右SELECT的结果的行集。

除非指定了ALL选项，否则EXCEPT的结果将不包含任何重复的行。使用ALL，在左表中具有 m 个重复项目在右表中具有 n 个重复项的行将在结果集中出现 $\max(m-n, 0)$ 次。可以编写DISTINCT来明确指定消除重复行的默认行为。

除非括号中另有规定，否则同一SELECT语句中的多个EXCEPT运算符从左到右求值。EXCEPT与UNION优先级相同。

当前，不能为EXCEPT结果或EXCEPT的任何输入指定FOR NO KEY UPDATE, FOR UPDATE, FOR SHARE和FOR KEY SHARE。

ORDER BY子句

可选的ORDER BY子句具有以下一般形式：

```
ORDER BY expression [ASC | DESC | USING operator] [NULLS {FIRST | LAST}] [, ...]
```

其中*expression*可以是输出列（SELECT列表项）的名称或序号，也可以是由输入列值组成的任意表达式。

ORDER BY子句使结果行根据指定的表达式排序。如果两行最左边的表达式相等，则根据下一个表达式对它们进行比较，依此类推。如果所有指定的表达式相等，则将根据实现的顺序返回它们。

序号是指输出列的序数位置（从左到右）。通过此特性，可以基于没有唯一名称的列来定义顺序。这绝对不是必须的，因为始终可以使用AS子句为输出列分配名称。

还可以在ORDER BY子句中使用任意表达式，包括未出现在SELECT输出列表中的列。因此，以下语句是有效的：

```
SELECT name FROM distributors ORDER BY code;
```

此特性的局限性在于，应用于UNION, INTERSECT或EXCEPT子句结果的ORDER BY子句只能指定输出列名称或序号，而不能指定表达式。

如果ORDER BY表达式是一个与输出列名和输入列名都匹配的简单名称，则ORDER BY将其解释为输出列名。这与GROUP BY在相同情况下所做的选择相反。这两个语句的不一致行为与SQL标准兼容。

可以选择在ORDER BY子句中的任何表达式之后添加关键字ASC（升序）或DESC（降序）。如果未指定，则默认采用ASC。或者，可以在USING子句中指定特定的排序运算符名称。ASC通常等效于USING <，而DESC通常等效于USING >。（但是，用户定义数据类型的创建者可以准确定义默认的排序顺序，并且它可能与其他名称的运算符相对应。）

如果指定了NULLS LAST，则空值将在所有非空值之后排序；如果指定了NULLS FIRST，则空值将在所有非空值之前排序。如果都未指定，则默认行为是在指定或隐含ASC时为NULLS LAST，而在指定DESC时为NULLS FIRST（因此，默认值表示空值大于非空值）。指定USING时，默认的空值排序取决于运算符是小于运算符还是大于运算符。

请注意，排序选项仅适用于它们遵循的表达式。例如，`ORDER BY x, y DESC`的含义与`ORDER BY x DESC, y DESC`的含义不同。

字符串数据是根据创建数据库时建立的特定于语言环境的排序顺序进行排序的。

字符串数据是根据应用于要排序的列的排序规则进行排序的。可以根据需要通过在*expression*中包含COLLATE子句来覆盖它，例如`ORDER BY mycolumn COLLATE "en_US"`。有关定义排序规则的信息，请参见[CREATE COLLATION](#)。

LIMIT子句

LIMIT 子句包含两个独立的子句：

```
LIMIT {count | ALL}
OFFSET start
```

其中*count*指定要返回的最大行数，而*start*指定在开始返回行之前要跳过的行数。如果同时指定了两者，则在开始对要返回的*count* 行进行计数之前，将跳过起始行。

如果*count* 表达式的计算结果为NULL，则将其视为LIMIT ALL，即无限制。如果*start* 计算为NULL，则将其与OFFSET 0 相同。

SQL: 2008引入了不同的语法来实现相同的结果，Greenplum数据库也支持该语法。它是：

```
OFFSET start [ ROW | ROWS ]
FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY
```

在这种语法中，标准要求*start*或*count*值是文字常量，参数或变量名；作为Greenplum数据库扩展，可以使用其他表达式，但通常需要将其括在括号中以避免歧义。如果在FETCH子句中省略了*count*，则其默认值为1。

ROW和ROWS以及FIRST 和NEXT是不影响这些子句效果的干扰词。根据标准，如果两者都存在，则OFFSET子句必须位于FETCH子句之前。但Greenplum数据库允许使用任何顺序。

使用LIMIT 时，最好使用ORDER BY子句将结果行强制为唯一的顺序。否则，您将获得查询行的不可预测的子集 - 您可能会要求以第十到第二十行，但以什么顺序要求第十到第二十行？除非您指定ORDER BY，否则您不知道该如何排序。

查询优化器在生成查询计划时会考虑LIMIT，因此根据LIMIT 和OFFSET的使用方式，您很可能会获得不同的计划（产生不同的行顺序）。因此，除非使用ORDER BY强制执行可预测的结果顺序，否则使用不同的LIMIT/OFFSET 值选择查询结果的不同子集将产生不一致的结果。这不是缺陷；这是一个正常的

结果，即除非使用ORDER BY强制该顺序，否则SQL不会保证以任何特定顺序传递查询结果。

Locking子句

FOR UPDATE, FOR NO KEY UPDATE, FOR SHARE和FOR KEY SHARE是锁定子句；它们会影响SELECT如何锁定从表中获取的行。

锁定子句具有一般形式

```
FOR lock_strength [OF table_name [ , ... ] ] [ NOWAIT ]
```

其中`lock_strength`可以是以下之一

```
UPDATE
NO KEY UPDATE
SHARE
KEY SHARE
```

Note: 默认情况下，Greenplum数据库获取对表的EXCLUSIVE锁，以对堆表进行DELETE和UPDATE操作。启用全局死锁检测器后，堆表上DELETE和UPDATE操作的锁定模式为ROW EXCLUSIVE 通过将`gp_enable_global_deadlock_detector` 配置参数设置为true来启用全局死锁检测器。有关全局死锁检测器的信息，请参见Greenplum数据库管理员指南中的[全局死锁检测器](#)。

有关每种行级锁定模式的更多信息，请参见PostgreSQL文档中的[显式锁定](#)。

为防止该操作等待其他事务提交，请使用NOWAIT选项。使用NOWAIT 如果无法立即锁定选定的行，该语句将报告错误，而不是等待。请注意，NOWAIT仅适用于行级锁 - 所需的ROW SHARE表级锁仍以常规方式获取。如果需要不等待就获取表级锁，则可以先将LOCK与NOWAIT选项一起使用。

如果在锁定子句中命名了特定的表，则仅锁定来自那些表的行；其他使用SELECT的表则像往常一样简单地读取。没有表列表的锁定子句会影响该语句中使用的所有表。如果将锁定子句应用于视图或子查询，则它将影响视图或子查询中使用的所有表。但是，这些子句不适用于主查询引用的WITH查询。如果希望在WITH查询中发生行锁定，请在WITH查询中指定一个锁定子句。

如果有必要为不同的表指定不同的锁定行为，则可以编写多个锁定子句。如果同一表被两个以上的锁定子句提及（或隐式影响），则将其视为仅由最强的一个子表指定。同样，如果在影响该表的任何子句中指定了表，则该表将作为NOWAIT处理。

在无法用单独的表行清楚地标识返回的行的上下文中，不能使用锁定子句；例如，它们不能与聚合并一起使用。

当锁定子句出现在SELECT查询的顶层时，被锁定的行与查询返回的行完全相同。对于连接查询，锁定的行是那些促成返回连接行的行。此外，查询快照中满足查询条件的行将被锁定，尽管这些行在快照后对其进行更新并且不再满足查询条件不会返回。如果使用LIMIT，则一旦返回了足够的行以满足限制，锁定就会停止（但请注意，被OFFSET跳过的行将被锁定）。同样，如果在游标的查询中使用锁定子句，则仅锁定实际被游标读取或跳过的行。

当sub-SELECT中出现locking子句时，被锁定的行就是该子查询返回给外部查询的行。这可能涉及的行数少于单独检查子查询所建议的行数，因为外部查询的条件可用于优化子查询的执行。例如，

```
SELECT * FROM (SELECT * FROM mytable FOR UPDATE) ss WHERE col1 = 5;
```

将仅锁定`col1 = 5` 的行，即使该条件在文本上不在子查询中。

`SELECT`命令可能以`READ COMMITTED`事务隔离级别运行，并且使用`ORDER BY`和锁定子句可以使行无序返回。这是因为`ORDER BY`首先被应用。该命令对结果进行排序，但随后可能会阻止尝试获得对一个或多个行的锁定。一旦`SELECT`解除阻塞，某些排序列值可能已被修改，从而导致这些行看起来是乱序的（尽管就原始列值而言，它们是有序的）。例如，可以通过在子查询中放置`FOR UPDATE/SHARE`子句来解决此问题。

```
SELECT * FROM (SELECT * FROM mytable FOR UPDATE) ss ORDER BY column1;
```

请注意，这将导致锁定`mytable` 的所有行，而顶层的`FOR UPDATE`将仅锁定实际返回的行。这可能会导致明显的性能差异，特别是如果将`ORDER BY`与`LIMIT`或其他限制结合使用。因此，仅当期望并发更新排序列并且需要严格排序的结果时才建议使用此技术。

在`REPEATABLE READ`或`SERIALIZABLE`事务隔离级别，这将导致序列化失败（SQLSTATE为40001），因此在这些隔离级别下，不可能乱序接收行。

TABLE命令

命令

```
TABLE name
```

完全等同于

```
SELECT * FROM name
```

在部分复杂查询中，它可用作顶级命令或节省空间的语法变体。

示例

连接表`films` 与表`distributors`：

```
SELECT f.title, f.did, d.name, f.date_prod, f.kind FROM
distributors d, films f WHERE f.did = d.did
```

将表`films`的`length` 列求和并将结果按`kind` 分组：

```
SELECT kind, sum(length) AS total FROM films GROUP BY kind;
```

将表films的length 列求和并将结果按kind 分组并显示总和小于5小时的组:

```
SELECT kind, sum(length) AS total FROM films GROUP BY kind
HAVING sum(length) < interval '5 hours';
```

计算电影kind 和distributor 的所有销售部分和与总和。

```
SELECT kind, distributor, sum(prc*qty) FROM sales
GROUP BY ROLLUP(kind, distributor)
ORDER BY 1,2,3;
```

根据总销量计算电影发行商的排名:

```
SELECT distributor, sum(prc*qty),
       rank() OVER (ORDER BY sum(prc*qty) DESC)
FROM sale
GROUP BY distributor ORDER BY 2 DESC;
```

以下两个示例是根据第二列 (name) 的内容对单个结果进行排序的相同方法:

```
SELECT * FROM distributors ORDER BY name;
SELECT * FROM distributors ORDER BY 2;
```

下一个示例说明如何获取表distributors 和actors 的并集, 将结果限制为每个表中以字母W开头的行。只需要不同的行, 因此关键字ALL被省略:

```
SELECT distributors.name FROM distributors WHERE
distributors.name LIKE 'W%' UNION SELECT actors.name FROM
actors WHERE actors.name LIKE 'W%';
```

此示例说明如何在FROM子句中使用函数, 无论是否包含列定义列表:

```
CREATE FUNCTION distributors(int) RETURNS SETOF distributors
AS $$ SELECT * FROM distributors WHERE did = $1; $$ LANGUAGE
SQL;
SELECT * FROM distributors(111);

CREATE FUNCTION distributors_2(int) RETURNS SETOF record AS
$$ SELECT * FROM distributors WHERE did = $1; $$ LANGUAGE
SQL;
SELECT * FROM distributors_2(111) AS (dist_id int, dist_name
text);
```

此示例使用一个简单的WITH子句:

```
WITH test AS (
    SELECT random() as x FROM generate_series(1, 3)
)
SELECT * FROM test
UNION ALL
SELECT * FROM test;
```

本示例使用WITH子句仅显示最高销售区域中的每产品销售总额。

```

WITH regional_sales AS
  SELECT region, SUM(amount) AS total_sales
  FROM orders
  GROUP BY region
), top_regions AS (
  SELECT region
  FROM regional_sales
  WHERE total_sales > (SELECT SUM(total_sales) FROM
    regional_sales)
)
SELECT region, product, SUM(quantity) AS product_units,
  SUM(amount) AS product_sales
FROM orders
WHERE region IN (SELECT region FROM top_regions)
GROUP BY region, product;

```

该示例可能编写不包含WITH子句，但需要两级嵌套的sub-SELECT语句。

本示例使用WITH RECURSIVE子句从仅显示直接下属的表中查找员工Mary的所有下属（直接或间接）及其间接级别：

```

WITH RECURSIVE employee_recursive(distance, employee_name,
manager_name) AS (
  SELECT 1, employee_name, manager_name
  FROM employee
  WHERE manager_name = 'Mary'
UNION ALL
  SELECT er.distance + 1, e.employee_name, e.manager_name
  FROM employee_recursive er, employee e
  WHERE er.employee_name = e.manager_name
)
SELECT distance, employee_name FROM employee_recursive;

```

递归查询的典型形式：初始条件，后跟UNION [ALL]，然后是查询的递归部分。确保查询的递归部分最终不会返回任何元组，否则查询将无限期地循环。有关更多示例，请参见[WITH查询 \(公用表表达式\)](#)。

兼容性

SELECT语句与SQL标准兼容，但是有一些扩展和某些缺少的功能。

省略FROM子句

Greenplum数据库允许省略FROM子句。它可以直接用于计算简单表达式的結果。例如：

```
SELECT 2+2;
```

其他一些SQL数据库无法做到这一点，除非引入一个虚拟的单行表来执行SELECT。

请注意，如果未指定FROM子句，则查询无法引用任何数据库表。例如，以下查询无效：

```
SELECT distributors.* WHERE distributors.name = 'Westward';
```

在较早的版本中，将服务器配置参数 `add_missing_from` 设置为 `true` 允许 Greenplum 数据库为查询所引用的每个表向查询的 `FROM` 子句添加隐式条目。这个不再允许。
省略 **AS** 关键字

在 SQL 标准中，只要新列名是有效的列名（即与任何保留关键字不同），就可以在输出列名之前省略可选关键字 **AS**。Greenplum 数据库的限制更为严格：如果新列名完全匹配任何关键字（保留与否），则要求使用 **AS**。推荐的做法是使用 **AS** 或双引号输出列名，以防止与将来添加关键字的任何可能的冲突。

在 `FROM` 项中，SQL 标准和 Greenplum 数据库都允许在作为未保留关键字的别名之前省略 **AS**。但是由于语法上的歧义，这对于输出列名称是不切实际的。

ONLY 和 **Inheritance**

使用 **ONLY** 时，SQL 标准要求在表名前后加上括号，例如：

```
SELECT * FROM ONLY (tab1), ONLY (tab2) WHERE ...
```

Greenplum 数据库认为这些括号是可选的。

Greenplum 数据库允许编写尾随 * 来明确指定包括子表的 non-**ONLY** 行为。该标准不允许这样做。

（这些要点同样适用于所有支持 **ONLY** 选项的 SQL 命令。）

Namespace 可以用于 **GROUP BY** 和 **ORDER BY**

在 SQL-92 标准中，**ORDER BY** 子句只能使用输出列名称或序号，而 **GROUP BY** 子句只能使用基于输入列名称的表达式。Greenplum 数据库扩展了这些子句中的每一个，以允许其他选择（但是如果有歧义，它将使用标准的解释）。Greenplum 数据库还允许两个子句都指定任意表达式。请注意，出现在表达式中的名称始终被视为输入列名称，而不是输出列名称。

SQL: 1999 及更高版本使用的定义略有不同，但并不完全与 SQL-92 向上兼容。但是，在大多数情况下，Greenplum 数据库以与 SQL: 1999 相同的方式解释 **ORDER BY** 或 **GROUP BY** 表达式。

函数依赖

仅当表的主键包含在 **GROUP BY** 列表中时，Greenplum 数据库才能识别函数依赖（允许从 **GROUP BY** 省略列）。SQL 标准指定了应识别的其他条件。

LIMIT 和 **OFFSET**

子句 **LIMIT** 和 **OFFSET** 是 Greenplum 数据库特定的语法，也由 MySQL 使用。如上所述，SQL: 2008 标准引入了子句 `OFFSET .. FETCH {FIRST | NEXT} ...` 来实现相同的功能。IBM DB2 也使用此语法。（Oracle 应用程序经常使用一种变通办法来实现这些子句的效果，该变通办法涉及自动生成的 `rownum` 列，Greenplum 数据库中不提供该列。）

FOR NO KEY UPDATE, **FOR UPDATE**, **FOR SHARE** 和 **FOR KEY SHARE**

尽管 **FOR UPDATE** 出现在 SQL 标准中，但该标准仅允许将它作为 **DECLARE**

CURSOR的选项。Greenplum数据库允许在任何SELECT查询以及sub-SELECT中使用它，但这是一个扩展。FOR NO KEY UPDATE, FOR SHARE和FOR KEY SHARE变体以及NOWAIT选项未出现在标准中。

WITH中的数据修改语句

Greenplum数据库允许将INSERT, UPDATE和DELETE用作WITH查询。在SQL标准中不允许。

非标准子句

在SQL标准中未定义DISTINCT ON子句。

STABLE和VOLATILE函数的限制

为防止数据在Greenplum数据库中的各个segment之间变得不同步，如果分类为STABLE或VOLATILE的任何函数包含SQL或以任何方式修改了数据库，则不能在segment数据库级别执行该函数。有关更多信息，请参见[CREATE FUNCTION](#)。

另见

[EXPLAIN](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

根据查询结果定义一个新表。

概要

```
[ WITH [ RECURSIVE ] with_query [ , ... ] ]
SELECT [ALL | DISTINCT [ON ( expression [ , ... ] )]]
      * | expression [AS output_name] [ , ... ]
      INTO [TEMPORARY | TEMP | UNLOGGED] [TABLE] new_table
      [FROM from_item [ , ... ]]
      [WHERE condition]
      [GROUP BY expression [ , ... ]]
      [HAVING condition [ , ... ]]
      [ {UNION | INTERSECT | EXCEPT} [ALL | DISTINCT] select ]
      [ORDER BY expression [ASC | DESC | USING operator]
      [NULLS {FIRST | LAST}] [ , ... ]]
      [LIMIT {count | ALL}]
      [OFFSET start [ ROW | ROWS ] ]
      [FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
      [FOR {UPDATE | SHARE} [OF table_name [ , ... ]]] [NOWAIT]
      [ ... ]]
```

描述

SELECT INTO创建一个新表，并用查询计算的数据填充该表。数据不会像常规SELECT一样返回给客户端。新表的列具有与SELECT的输出列关联的名称和数据类型。

参数

SELECT INTO的大多数参数与[SELECT](#)相同。

TEMPORARY

TEMP

如果指定，该表将创建为临时表。

UNLOGGED

如果指定，该表将创建为不计入日志表。写入不计入日志表的数据不会写入预写 (WAL) 日志，这使它们比普通表快得多。

但是，不计入日志表的内容不会复制到mirror实例。同样，不计入日志表也不是崩溃安全的。segment实例崩溃或异常关闭后，

该segment上不计入日志表的数据将被截断。在不计入日志表上创建的所有索引也会自动取消计入日志。

new_table

要创建的表的名称（可以用schema修饰）。

示例

创建一个新表films_recent，该表仅包含表films中的最新条目：

```
SELECT * INTO films_recent FROM films WHERE date_prod >=
'2016-01-01';
```

兼容性

SQL标准使用SELECT INTO表示将值选择到主机程序的标量变量中，而不是创建新表。SELECT INTO在Greenplum数据库中用于表示表创建的用法是历史性的。为此，最好在新应用程序中使用[CREATE TABLE AS](#)。

另见

[SELECT, CREATE TABLE AS](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更改Greenplum数据库配置参数的值。

概要

```
SET [SESSION | LOCAL] configuration_parameter {TO | =}
value | 'value' | DEFAULT
SET [SESSION | LOCAL] TIME ZONE {timezone | LOCAL |
DEFAULT}
```

描述

SET命令更改服务器配置参数。任何分类为*session*参数的配置参数都可以使用SET即时更改。SET仅影响当前会话使用的值。

如果在稍后中止的事务中发出SET或SET SESSION，则回滚该事务时SET命令的效果会消失。一旦提交了事务，效果将一直持续到会话结束，除非被另一个SET覆盖。

SET LOCAL的影响仅持续到当前事务结束为止，无论是否提交。特殊情况是在单个事务中SET LOCAL紧随SET其后：SET LOCAL值将一直显示到事务结束为止，但是此后（如果提交了事务）SET值将生效。

如果在包含针对同一配置参数的SET选项的函数中使用SET LOCAL（请参见[CREATE FUNCTION](#)），则SET LOCAL命令的效果在函数退出时消失；无论如何，将恢复调用该函数时有效的值。这允许将SET LOCAL用于函数中参数的动态或重复更改，同时保留使用SET选项保存和恢复调用者值的便利。请注意，常规的SET命令会覆盖周围函数的SET选项。除非回滚，否则其影响持续存在。

如果在事务中使用DECLARE命令创建游标，则只有在使用CLOSE命令关闭游标后才能在事务中使用SET命令。

有关服务器参数的信息，请参阅[服务器配置参数](#)。



参数

SESSION

指定该命令对当前会话生效。这是默认值。

LOCAL

指定该命令仅对当前事务生效。在COMMIT或ROLLBACK之后，会话级设置将再次生效。请注意，如果SET LOCAL在事务外部执行，则似乎无效。

configuration_parameter

Greenplum数据库配置参数的名称。使用SET只能更改分类为*session*的参数。有关详细信息，请参见[服务器配置参数](#)。

value

参数的新值。可以将值指定为字符串常量，标识符，数字或以逗号分隔的列表。DEFAULT可用于指定将参数重置为其默认值。如果指定内存大小或时间单位，则将值用单引号引起。

TIME ZONE

SET TIME ZONE值是SET timezone TO *value*。语法SET TIME ZONE允许时区指定使用特殊语法。以下是有效值的示例：

'PST8PDT'

'Europe/Rome'

-7 (UTC以西7小时时区)

INTERVAL '-08:00' HOUR TO MINUTE (UTC以西8小时时区)。

LOCAL

DEFAULT

将时区设置为您的本地时区（即服务器的默认*timezone*）。有关Greenplum数据库中时区的更多信息，请参见[PostgreSQL文档的时区部分](#)。

示例

设置schema搜索路径：

```
SET search_path TO my_schema, public;
```

将每个查询的segment主机内存增加到200 MB：

```
SET statement_mem TO '200MB';
```

把日期风格设置为传统POSTGRES的"日在月之前"的输入习惯：

```
SET datestyle TO postgres, dmy;
```

设置加利福尼亚州圣马特奥市的时区（太平洋时间）：

```
SET TIME ZONE 'PST8PDT';
```

设置意大利的时区：

```
SET TIME ZONE 'Europe/Rome';
```

兼容性

`SET TIME ZONE`扩展了SQL标准中定义的语法。该标准仅允许数字时区偏移，而Greenplum数据库允许更灵活的时区规范。`SET`的所有其他功能都是Greenplum数据库扩展。

另见

[RESET](#), [SHOW](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

设置当前事务的约束检查时间。

Note: 引用完整性语法（外键约束）被接受但未强制执行。

概要

```
SET CONSTRAINTS { ALL | name [, ...] } { DEFERRED | IMMEDIATE }
```

描述

SET CONSTRAINTS设置当前事务中的约束检查行为。在每个语句的末尾检查IMMEDIATE约束。在提交事务之前，不检查DEFERRED约束。每个约束都有其自己的IMMEDIATE或DEFERRED模式。

创建后，将为约束提供以下三个特征之一：DEFERRABLE INITIALLY DEFERRED, DEFERRABLE INITIALLY IMMEDIATE或NOT DEFERRABLE。第三类始终为IMMEDIATE，不受SET CONSTRAINTS命令的影响。前两个类以指示的模式启动每个事务，但是可以通过SET CONSTRAINTS在事务内更改其行为。

带有约束名称列表的SET CONSTRAINTS会更改那些约束的模式（必须全部推迟）。每个约束名称可以是schema限定的。如果未指定schema名称，则使用当前schema搜索路径查找第一个匹配名称。SET CONSTRAINTS ALL更改所有可延缓约束的模式。

当SET CONSTRAINTS将约束的模式从DEFERRED更改位IMMEDIATE时，新模式将追溯生效：在执行SET CONSTRAINTS命令时，将检查在事务结束时会被检查的所有未完成的数据修改。如果违反了任何此类约束，则SET CONSTRAINTS会失败（并且不会更改约束模式）。因此，SET CONSTRAINTS可以用于强制检查约束以在事务的特定点发生。

当前，此设置仅影响UNIQUE, PRIMARY KEY, REFERENTIAL (外键) 和EXCLUDE约束。当插入或修改一行时（不在语句`ON UPDATE`），始终会立即检查NOT NULL和CHECK约束。还可以立即检查尚未声明为DEFERRABLE的唯一性和排除约束。

声明为“约束触发器”的触发器的触发也由此设置控制-它们在应检查相关约束的同时触发。

注解

因为Greenplum数据库不要求约束名称在schema中唯一（而仅按表），所以指定约束名称可能有多个匹配项。在这种情况下，SET CONSTRAINTS将对所有匹配起作用。对于非schema限定的名称，一旦在搜索路径中的某些schema中找到一个或多个匹配项，就不会搜索路径中稍后出现的schema。

此命令仅更改当前事务中约束的行为。在事务块外部发出此命令将发出警告，否则无效。

兼容性

该命令符合SQL标准中定义的行为，除了以下限制：在Greenplum数据库中，该命令不适用于NOT NULL和CHECK约束。另外，Greenplum数据库会立即检查不可延展的唯一性约束，而不是像标准建议那样在声明结束时进行检查。

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT
PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL
TABLEALTER FOREIGN
DATA WRAPPERALTER FOREIGN
TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

设置当前会话的当前角色标识符。

概要

SET [SESSION | LOCAL] ROLE *rolename*

SET [SESSION | LOCAL] ROLE NONE

RESET ROLE

描述

此命令将当前SQL会话上下文的当前角色标识符设置为*rolename*。 角色名称可以写为标识符或字符串文字。 在SET ROLE之后，将执行SQL命令的权限检查，就好像新角色是最初登录的角色一样。

指定的*rolename*必须是当前会话用户所属的角色。 如果会话用户是超级用户，则可以选择任何角色。

NONE和RESET表单将当前角色标识符重置为当前会话角色标识符。 这些表格可以由任何用户执行。

参数

SESSION

指定该命令对当前会话生效。这是默认值。

LOCAL

指定该命令仅对当前事务生效。在COMMIT或ROLLBACK之后，会话级设置将再次生效。请注意，如果SET LOCAL在事务外部执行，则似乎无效。

rolename

在此会话中用于权限检查的角色名称。

NONE

RESET

将当前角色标识符重置为当前会话角色标识符（用于登录的角色的标识符）。



注解

使用此命令，可以添加特权或限制特权。如果会话用户角色具有INHERITS属性，则它将自动拥有可以SET ROLE的每个角色的所有特权；在这种情况下，SET ROLE有效地放弃直接分配给会话用户及其成员的其他角色的所有特权，仅保留命名角色可用的特权。另一方面，如果会话用户角色具有NOINHERITS属性，则SET ROLE会删除直接分配给会话用户的特权，而是获取可用于命名角色的特权。

特别是，当超级用户选择将SET ROLE设置为非超级用户角色时，她将失去其超级用户特权。

SET ROLE具有与SET SESSION AUTHORIZATION相同的效果，但是所涉及的特权检查却大不相同。同样，SET SESSION AUTHORIZATION确定以后的SET ROLE命令允许哪些角色，而使用SET ROLE不会更改后续SET ROLE可以更改的角色集。

SET ROLE不处理由角色的ALTER ROLE设置指定的会话变量。会话变量仅在登录期间处理。

示例

```
SELECT SESSION_USER, CURRENT_USER;
session_user | current_user
-----+-----
peter | peter

SET ROLE 'paul';

SELECT SESSION_USER, CURRENT_USER;
session_user | current_user
-----+-----
peter | paul
```

兼容性

Greenplum数据库允许使用标识符语法 (*rolename*)，而SQL标准要求将角色名称写为字符串文字。SQL在事务期间不允许使用此命令。Greenplum数据库没有进行此限制。SESSION和LOCAL修饰符是Greenplum数据库的扩展名，RESET语法也是如此。

另见

[SET SESSION AUTHORIZATION](#)

Parent topic: [SQL Command Reference](#)

AUTHORIZATION

Greenplum数据库® 6.0文档

设置会话角色标识符和当前会话的当前角色标识符。

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

概要

```
SET [SESSION | LOCAL] SESSION AUTHORIZATION rolename
```

```
SET [SESSION | LOCAL] SESSION AUTHORIZATION DEFAULT
```

```
RESET SESSION AUTHORIZATION
```

描述

此命令将会话角色标识符和当前SQL会话上下文的当前角色标识符设置为*rolename*。角色名称可以写为标识符或字符串文字。使用此命令，例如，可以暂时成为非特权用户，然后再切换回为超级用户。

会话角色标识符最初设置为客户端提供的（可能经过身份验证的）角色名称。当前角色标识符通常等于会话用户标识符，但是在setuid函数和类似机制的上下文中可能会暂时更改；也可以通过[SET ROLE](#)进行更改。当前用户标识符与权限检查相关。

仅当初始会话用户（已认证用户）具有超级用户特权时，才可以更改会话用户标识符。否则，仅当命令指定了经过身份验证的用户名时，该命令才会被接受。

DEFAULT和RESET表单将会话和当前用户标识符重置为原始身份验证的用户名。这些表格可以由任何用户执行。

参数

SESSION

指定该命令对当前会话生效。这是默认值。



LOCAL

指定该命令仅对当前事务生效。在COMMIT或ROLLBACK之后，

SET LOCAL

会话级设置将再次生效。请注意，如果执行，则似乎无效。

在事务外部

rolename

指定的角色的名称。

NONE

RESET

将会话和当前角色标识符重置为用于登录的角色的标识符。

示例

```
SELECT SESSION_USER, CURRENT_USER;
session_user | current_user
-----+-----
peter | peter

SET SESSION AUTHORIZATION 'paul';

SELECT SESSION_USER, CURRENT_USER;
session_user | current_user
-----+-----
paul | paul
```

兼容性

SQL标准允许其他一些表达式代替文字*rolename*出现，但是这些选项在实践中并不重要。Greenplum数据库允许使用标识符语法(*rolename*)，而SQL则不允许。SQL在事务期间不允许使用此命令。Greenplum数据库没有进行此限制。SESSION和LOCAL修饰符是Greenplum数据库的扩展名，RESET语法也是如此。

另见

[SET ROLE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

设置当前事务的特性。

概要

```
SET TRANSACTION [transaction_mode] [READ ONLY | READ WRITE]
```

```
SET TRANSACTION SNAPSHOT snapshot_id
```

```
SET SESSION CHARACTERISTICS AS TRANSACTION transaction_mode
[READ ONLY | READ WRITE]
[NOT] DEFERRABLE
```

其中*transaction_mode*为下列之一：

```
ISOLATION LEVEL {SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED}
```

描述

SET TRANSACTION命令设置当前事务的特性。 它对任何后续交易都没有影响。

可用的事务特性是事务隔离级别，事务访问模式（读/写或只读）和可延迟的模式。

Note: 可延迟事务要求该事务为可序列化。 Greenplum数据库不支持可序列化的事务，因此包含DEFERRABLE子句无效。

Greenplum数据库不支持SET TRANSACTION SNAPSHOT命令。

事务的隔离级别确定当其他事务同时运行时，该事务可以看到哪些数据。

- **READ COMMITTED** — 一条语句只能看到在开始之前提交的行。这是默认值。

- **REPEATABLE READ** — 当前事务中的所有语句只能在事务中执行的第一个查询语句之前提交的或该事务中数据修改语句修改的行。

SQL标准定义了两个附加级别，即READ UNCOMMITTED和SERIALIZABLE。在Greenplum数据库中，READ UNCOMMITTED被视为READ COMMITTED。如果指定SERIALIZABLE，则Greenplum数据库将退回到REPEATABLE READ。

在执行事务的第一个查询或数据修改语句

(SELECT, INSERT, DELETE, UPDATE, FETCH或COPY) 之后，不能更改事务隔离级别。

事务访问模式确定事务是读/写还是只读。读/写是默认设置。当事务为只读时，不允许使用以下SQL命令：INSERT, UPDATE, DELETE和COPY FROM (如果要写入的表不是临时表)；所有的CREATE, ALTER和DROP命令；GRANT, REVOKE, TRUNCATE；如果EXPLAIN ANALYZE和EXECUTE将要执行的命令在上述命令之中，也不被允许。这是只读的高级概念，不会阻止所有对磁盘的写入。

除非事务既是SERIALIZABLE又是READ ONLY，否则DEFERRABLE事务属性无效。当在事务上设置了所有这些属性后，该事务在首次获取其快照时可能会阻塞，此后它可以在没有SERIALIZABLE事务的正常开销的情况下运行，并且没有任何导致序列化失败或被序列化失败取消的风险。由于Greenplum数据库不支持可序列化的事务，因此DEFERRABLE事务属性在Greenplum数据库中无效。

参数

SESSION CHARACTERISTICS

为会话的后续事务设置默认事务特性。

READ UNCOMMITTED

READ COMMITTED

REPEATABLE READ

SERIALIZABLE

SQL标准定义了四个事务隔离级别：READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ和SERIALIZABLE。

READ UNCOMMITTED允许事务查看未提交的并发事务所做的更改。在Greenplum数据库中这是不可能的，因此READ UNCOMMITTED与READ COMMITTED相同。

READ COMMITTED是Greenplum数据库中的默认隔离级别，可确保语句只能看到在开始之前提交的行。如果在第一次执行该语

句后又提交了另一个并发事务，则在一个事务中执行两次的相同语句可能会产生不同的结果。

`REPEATABLE READ`隔离级别确保事务只能看到在事务开始之前提交的行。`REPEATABLE READ`是Greenplum数据库支持的最严格的事务隔离级别。由于可串行化失败，使用`REPEATABLE READ`隔离级别的应用程序，必须准备重试事务。

`SERIALIZABLE`事务隔离级别确保当前事务的所有语句只能看到在此事务中执行第一个查询之前或此事务内数据修改语句提交的行。如果并发可序列化事务之间的读取和写入模式会导致这种事务的任何串行（一次一个）执行都不可能发生的情况，则其中一个事务将回滚，并出现`serialization_failure`错误。Greenplum数据库不完全支持标准定义的`SERIALIZABLE`，因此，如果指定`SERIALIZABLE`，则Greenplum数据库将退回到`REPEATABLE READ`。有关Greenplum数据库中事务可串行化的更多信息，请参见[兼容性](#)。

READ WRITE

READ ONLY

确定事务是读/写还是只读。读/写是默认设置。当事务为只读时，不允许使用以下SQL命令：

`INSERT, UPDATE, DELETE`和`COPY FROM`（如果要写入的表不是临时表）；所有的`CREATE, ALTER`和`DROP`命令；`GRANT, REVOKE, TRUNCATE`；如果`EXPLAIN ANALYZE`和`EXECUTE`将要执行的命令在上述命令之中，也不被允许。

[NOT] DEFERRABLE

因为不支持`SERIALIZABLE`事务，所以`DEFERRABLE`事务属性在Greenplum数据库中无效。如果指定了`DEFERRABLE`且该事务既是`SERIALIZABLE`也是`READ ONLY`的，则该事务在首次获取其快照时可能会阻塞，此后它可以在没有`SERIALIZABLE`事务的正常开销的情况下运行，并且不存在构成或被取消可串行化失败风险。会造成任何贡献或被其取消序列化失败。此模式非常适合长时间运行的报告或备份。

注解

如果在没有事先进行`START TRANSACTION`或`BEGIN`的情况下执行`SET TRANSACTION`，则会发出警告，并且该命令无效。

通过在`BEGIN`或`START TRANSACTION`中指定所需的事务模式，可以省去`SET TRANSACTION`。

也可以通过设置配置参数[`default_transaction_isolation`](#)，

`default_transaction_read_only`和`default_transaction_deferrable` 来设置会话默认事务模式。

示例

设置当前事务的事务隔离级别：

```
BEGIN;  
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

兼容性

这两个命令均在SQL标准中定义。SERIALIZABLE是标准中的默认事务隔离级别。在Greenplum数据库中，默认值为READ COMMITTED。由于缺少谓词锁定，Greenplum数据库不完全支持SERIALIZABLE级别，因此当指定SERIALIZABLE时，它退回到REPEATABLE READ级别。本质上，谓词锁定系统通过限制写的内容来防止幻像读取，而Greenplum数据库中使用的多版本并发控制模型（MVCC）通过限制读取的内容来防止幻像读取。

PostgreSQL提供了一个真正的可序列化隔离级别，称为可序列化快照隔离（SSI），它可以监视并发事务并回滚可能引入序列化异常的事务。Greenplum数据库未实现此隔离模式。

在SQL标准中，可以使用以下命令设置其他事务特性：诊断区域的大小。此概念是嵌入式SQL特有的，因此未在Greenplum数据库服务器中实现。

DEFERRABLE事务模式是Greenplum数据库语言的扩展。

SQL标准要求连续的`transaction_modes`之间使用逗号，但是由于历史原因，Greenplum数据库允许省略逗号。

另见

[BEGIN, LOCK](#)

[Parent topic: SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

显示系统配置参数的值。

概要

SHOW *configuration_parameter*

SHOW ALL

描述

SHOW 显示Greenplum数据库系统配置参数的当前设置。 您可以使用SET语句或通过编辑Greenplum数据库master的`postgresql.conf`配置文件来设置这些参数。 请注意，SHOW可以查看的某些参数是只读的 - 可以查看但不能设置它们的值。 有关详细信息，请参见Greenplum数据库参考指南。

参数

configuration_parameter

系统配置参数的名称。

ALL

显示所有配置参数的当前值。

示例

显示参数DateStyle的当前设置：

```
SHOW DateStyle;
DateStyle
-----
ISO, MDY
(1 row)
```

显示参数geqo的当前设置：



```
SHOW geqo;
geqo
-----
off
(1 row)
```

显示所有参数的当前设置：

```
SHOW ALL;
      name      | setting | description
-----+-----+-----+
application_name | psql | Sets the application name to be
reported in sta...
.
.
.
xmlbinary       | base64 | Sets how binary values are to
be encoded in XML.
xmloption       | content | Sets whether XML data in
implicit parsing and s...
(331 rows)
```

兼容性

SHOW是Greenplum数据库扩展。

另见

[SET](#) , [RESET](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

启动事务块。

概要

```
START TRANSACTION [transaction_mode] [READ WRITE | READ ONLY]
```

其中*transaction_mode*是：

```
ISOLATION LEVEL {SERIALIZABLE | READ COMMITTED | READ UNCOMMITTED}
```

描述

START TRANSACTION开始一个新的事务块。如果指定了隔离级别或读/写模式，则新事务具有那些特征，就像执行[SET TRANSACTION](#)一样。这与BEGIN命令相同。

参数

READ UNCOMMITTED

READ COMMITTED

REPEATABLE READ

SERIALIZABLE

SQL标准定义了四个事务隔离级别：READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ和SERIALIZABLE。

READ UNCOMMITTED允许事务查看未提交的并发事务所做的更改。在Greenplum数据库中这是不可能的，因此READ UNCOMMITTED与READ COMMITTED相同。

READ COMMITTED是Greenplum数据库中的默认隔离级别，可确保语句只能看到在其开始之前提交的行。如果在
执行该语句后又提交了另一个并发事务，则在一个事务中执行两次的同一条语句可能会产生不同的结果。

REPEATABLE READ隔离级别确保事务只能看到在事务开始之

前提交的行。 REPEATABLE READ是Greenplum数据库支持的最严格的事务隔离级别。由于可串行化失败，必须准备使用REPEATABLE READ隔离级别的应用程序以重试事务。

SERIALIZABLE事务隔离级别确保执行多个并发事务与串行运行这些事务产生相同的效果。如果指定SERIALIZABLE，则Greenplum数据库将退回到REPEATABLE READ。

READ WRITE

READ ONLY

确定事务是读/写还是只读。读/写是默认设置。当事务为只读时，不允许使用以下SQL命令：

INSERT, UPDATE, DELETE和COPY FROM（如果要写入的表不是临时表）；所有的CREATE, ALTER和DROP命令；GRANT, REVOKE, TRUNCATE；如果EXPLAIN ANALYZE和EXECUTE将要执行的命令在上述命令之中，也不被允许。和EXPLAIN ANALYZE和EXECUTE，如果它们将要执行的命令在列出的命令之中。

示例

启动一个事务块：

```
START TRANSACTION;
```

兼容性

在标准中，不必发出START TRANSACTION来启动事务块：任何SQL命令都隐式地开始一个块。Greenplum数据库行为可以看作是在每个不遵循START TRANSACTION（或BEGIN）的命令之后隐式发出COMMIT，因此通常被称为“自动提交”。其他关系数据库系统可能会提供便利的自动提交功能。

SQL标准要求连续的*transaction_modes*之间使用逗号，但是由于历史原因，Greenplum数据库允许省略逗号。

另请参阅[SET TRANSACTION](#)的兼容性部分。

另见

[BEGIN, SET TRANSACTION](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

清空表的所有行。

概要

```
TRUNCATE [ TABLE ] [ ONLY ] name [ * ] [, ...]
          [ RESTART IDENTITY | CONTINUE IDENTITY ] [ CASCADE |
          RESTRICT]
```

描述

TRUNCATE快速删除一个表或一组表中的所有行。 它与在每个表上进行不限定[DELETE](#)的效果相同，但是由于它实际上并不扫描表，因此速度更快。 这在大型表上最有用。

您必须在表上具有TRUNCATE特权才能清空表行。

TRUNCATE在对其操作的表上获取访问排他锁，这将阻止该表上的所有其他并发操作。 如果指定了RESTART IDENTITY，则将要重新计数的所有序列都将被排他地锁定。 如果需要并发访问表，则应改用[DELETE](#)命令。

参数

name

要清空的表的名称（可以用schema修饰）。 如果在表名之前指定了ONLY，则仅该表被清空。 如果未指定ONLY，则该表及其所有子表（如果有）将被清空。（可选）可以在表名称后指定*，以显式指示包括子表。

CASCADE

因为此关键字适用于外键引用（Greenplum数据库中不支持），所以它无效。

RESTART IDENTITY

自动重新启动被清空表的列所拥有的序列。

CONTINUE IDENTITY

不要更改序列的值。 这是默认值。



RESTRICT

因为此关键字适用于外键引用（Greenplum数据库中不支持），所以它无效。

注解

TRUNCATE将不会运行表可能存在的任何用户定义的ON DELETE触发器。

TRUNCATE不会清空从指定表继承的任何表。仅指定的表被清空，而不是其子表。

TRUNCATE不会清空分区表的任何子表。如果指定分区表的子表，则TRUNCATE不会从该表及其子表中删除行。

TRUNCATE不是MVCC安全的。清空后，如果并发事务使用清空发生之前获取的快照，则该表将对并发事务显示为空。

对于表中的数据，TRUNCATE是事务安全的：如果所属的事务未提交，则清空将被安全地回滚。

TRUNCATE在其操作的每个表上获取ACCESS EXCLUSIVE锁，该锁将阻止该表上的所有其他并发操作。如果需要并发访问表，则应改用DELETE命令。

当指定了RESTART IDENTITY时，隐式的ALTER SEQUENCE RESTART操作也将以事务方式完成；也就是说，如果所在的事务没有提交，它们将被回滚。这不同于ALTER SEQUENCE RESTART的正常行为。请注意，如果在事务回滚之前对重新启动的序列执行了任何其他序列操作，则这些操作对序列的影响将被回滚，但不会对currval()产生影响。也就是说，在事务currval()之后，它将继续反映在失败的事务中获得的最后一个序列值，即使序列本身可能不再与此一致。这类似于事务失败后currval()的通常行为。

示例

清空表films和distributors：

```
TRUNCATE films, distributors;
```

相同，并重置所有关联的序列生成器：

```
TRUNCATE films, distributors RESTART IDENTITY;
```

兼容性

SQL: 2008标准包括带有语法`TRUNCATE TABLE tablename`的`TRUNCATE`命令。子句`CONTINUE IDENTITY/RESTART IDENTITY`也出现在该标准中，但是尽管具有相关的含义，但略有不同。该命令的一些并发行为由标准实现定义，因此应考虑上述注意事项，并在必要时与其他实现进行比较。

另见

[DELETE](#) , [DROP TABLE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

更新表的行。

概要

```
[ WITH [ RECURSIVE ] with_query [ , ... ]
      UPDATE [ONLY] table [[AS] alias]
                  SET {column = {expression | DEFAULT} |
                      (column [, ...]) = ({expression |
                      DEFAULT} [, ...])} [, ...]
                  [FROM fromlist]
                  [WHERE condition | WHERE CURRENT OF
cursor_name ]]
```

描述

UPDATE更改所有满足条件的行中指定列的值。只需在SET子句中提及要修改的列；未显式修改的列将保留其先前的值。

默认情况下，UPDATE将更新指定表及其所有子表中的行。如果只希望更新提到的特定表，则必须使用ONLY子句。

有两种方法可以使用数据库中其他表中包含的信息来修改表：使用子选择，或在FROM子句中指定其他表。哪种技术更合适取决于具体情况。

如果指定了WHERE CURRENT OF子句，则更新的行是从指定游标中最新获取的行。

复制表不支持WHERE CURRENT OF子句。

您必须在表上或至少在要更新的列上具有UPDATE特权。您还必须拥有读取*expression*或*condition*中的任何列的SELECT特权。

Note: 默认情况下，Greenplum数据库为堆表的UPDATE操作获取表的EXCLUSIVE锁。启用全局死锁检测器后，堆表上UPDATE操作的锁定模式为ROW EXCLUSIVE。请参阅[全局死锁检测](#)。

输出

成功完成后，UPDATE命令将返回以下格式的命令标记：

```
UPDATE count
```

其中*count*是更新的行数。如果*count*为0，则没有符合条件的行（这不视为错误）。

参数

with_query

WITH子句允许您指定一个或多个子查询，这些子查询可以在UPDATE查询中按名称进行引用。

对于包含WITH子句的UPDATE命令，该子句只能包含SELECT命令，而WITH子句不能包含数据修改命令（INSERT, UPDATE或DELETE）。

查询（SELECT语句）也可能包含WITH子句。在这种情况下，可以在UPDATE查询中引用两组*with_query*，但是第二组优先，因为它的嵌套更紧密。

更多信息，请参考[WITH查询（公用表表达式）](#)和[SELECT](#)。

ONLY

如果指定，则仅更新指定表中的行。如果未指定，还将处理从指定表继承的任何表。

table

现有表的名称（可以用schema修饰）。

alias

目标表的替代名称。提供别名后，它将完全隐藏表的实际名称。例如，在给定UPDATE *foo* AS *f*的情况下，UPDATE语句的其余部分必须将此表称为*f*而不是*foo*。

column

表中列的名称。如果需要，可以使用子字段名称或数组下标来限定列名称。在目标列的规范中不要包含表的名称。

expression

分配给该列的表达式。该表达式可以使用表中此列和其他列的旧值。

DEFAULT

将列设置为其默认值（如果未分配任何特定的默认表达式，则为NULL）。

fromlist

表表达式的列表，允许其他表中的列出现在WHERE条件和更新表达式中。这类似于可以在SELECT语句的FROM子句中指定的表的列表。请注意，除非您打算进行自连接，否则目标表不得出现在*fromlist*中（在这种情况下，目标表必须带有别名出现在*fromlist*中）。

condition

该表达式返回boolean类型的值。仅此表达式返回true的行将被更新。

cursor_name

在WHERE CURRENT OF条件中使用的游标名称。要更新的行是从游标最近获取的行。游标必须是UPDATE命令目标表上的非分组查询。有关创建游标的更多信息，请参见[DECLARE](#)。

不能与布尔条件一起指定WHERE CURRENT OF。

注意，不能将WHERE CURRENT OF与布尔条件一起指定。

UPDATE...WHERE CURRENT OF语句只能在服务器上执行，

例如在交互式psql会话或脚本中。语言扩展（例如PL / pgSQL）不支持可更新的游标。

有关创建游标的更多信息，请参见[DECLARE](#)。

output_expression

每行更新后，由UPDATE命令计算并返回表达式。该表达式可以使用FROM中列出的一个或多个表的任何列名。输入*以返回所有列。

output_name

用于返回的列的名称。

注解

在表的Greenplum分布键列上不允许使用SET。

当存在FROM子句时，本质上是将目标表连接到from列表中提到的表，并且连接的每个输出行都代表目标表的更新操作。使用FROM时，应确保该连接为要修改的每一行最多产生一个输出行。换句话说，目标行不应与其他表中的一行连接。如果是这样，那么将仅使用连接行之一来更新目标行，但是将很难预测将使用哪一行。

由于存在这种不确定性，因此仅在子选择内引用其他表会更安全，尽管与使用连接相比，通常更难阅读，也更慢。

不支持在分区表的特定分区（子表）上直接执行UPDATE和DELETE命令。而是在根分区表（使用CREATE TABLE命令创建的表）上执行这些命令。

示例

将表films中的kind列，从Drama改为Dramatic：

```
UPDATE films SET kind = 'Dramatic' WHERE kind = 'Drama';
```

调整温度条目并将表weather的一行中的降水重置为默认值：

```
UPDATE weather SET temp_lo = temp_lo+1, temp_hi =
temp_lo+15, prcp = DEFAULT
WHERE city = 'San Francisco' AND date =
'2016-07-03';
```

使用替代的列列表语法进行相同的更新：

```
UPDATE weather SET (temp_lo, temp_hi, prcp) = (temp_lo+1,
temp_lo+15, DEFAULT)
WHERE city = 'San Francisco' AND date =
'2016-07-03';
```

使用FROM子句语法增加管理Acme Corporation帐户的销售人员的销售数量（假设两个被连接的表在Greenplum数据库中都基于id列分布）：

```
UPDATE employees SET sales_count = sales_count + 1 FROM
accounts
WHERE accounts.name = 'Acme Corporation'
AND employees.id = accounts.id;
```

使用WHERE子句中的子选择来执行相同的操作：

```
UPDATE employees SET sales_count = sales_count + 1 WHERE id
=
(SELECT id FROM accounts WHERE name = 'Acme
Corporation');
```

尝试插入新的库存商品以及库存数量。如果物料已经存在，请更新现有物料的库存数量。要在不使整个事务失败的情况下执行此操作，请使用保存点。

```
BEGIN;
-- other operations
SAVEPOINT spl;
INSERT INTO wines VALUES('Chateau Lafite 2003', '24');
-- Assume the above fails because of a unique key
violation,
-- so now we issue these commands:
```

```
ROLLBACK TO sp1;
UPDATE wines SET stock = stock + 24 WHERE winename =
'Chateau
Lafite 2003';
-- continue with other operations, and eventually
COMMIT;
```

兼容性

该命令符合SQL标准，但FROM子句是Greenplum数据库扩展。

根据标准，列列表语法应允许从单个行值表达式（例如子选择）分配列列表：

```
UPDATE accounts SET (contact_last_name, contact_first_name)
=
  (SELECT last_name, first_name FROM salesmen
  WHERE salesmen.id = accounts.sales_id);
```

当前尚未实现-源必须是独立表达式的列表。

其他一些数据库系统提供FROM选项，该目标表应该在FROM中再次列出。那不是Greenplum数据库解释FROM的方式。移植使用此扩展名的应用程序时请小心。

另见

[DECLARE](#) , [DELETE](#) , [SELECT](#) , [INSERT](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

垃圾收集并可选地分析数据库。

概要

```
VACUUM [({ FULL | FREEZE | VERBOSE | ANALYZE } [, ...])]  
[table [(column [, ...])]]
```

```
VACUUM [FULL] [FREEZE] [VERBOSE] [table]
```

```
VACUUM [FULL] [FREEZE] [VERBOSE] ANALYZE  
[table [(column [, ...])]]
```

描述

VACUUM回收已删除的元组占用的存储。在正常的Greenplum数据库操作中，被更新删除或废弃的元组不会从表中物理删除；它们将保留在磁盘上，直到完成VACUUM。因此，有必要定期进行VACUUM，尤其是在频繁更新的表上。

如果没有参数，VACUUM将处理当前数据库中的每个表。使用参数，VACUUM仅处理该表。

VACUUM ANALYZE对每个选定的表执行VACUUM，然后执行ANALYZE。这是日常维护脚本的便捷组合形式。有关处理的更多详细信息，请参见[ANALYZE](#)。

VACUUM（无FULL）在表和索引中标记已删除和过时的数据以供将来重用，并且仅当该空间位于表的末尾并且可以轻松获得独占表锁时，才可以回收该空间以供重用。表开始或中间的未使用空间保持不变。对于堆表，这种形式的命令可以与表的正常读取和写入并行运行，因为不会获得排他锁。但是，在大多数情况下，多余的空间不会返回给操作系统。它只是可以在同一张表中重复使用。VACUUM FULL将表的全部内容重写为一个没有额外空间的新磁盘文件，从而允许将未使用的空间返回给操作系统。这种形式要慢得多，并且在处理过程中需要在每个表上获得排他锁。

使用追加优化表，VACUUM首先清理索引，然后依次压缩每个段文件，最后清理辅助表并更新统计信息，从而压缩表。在每个段上，将可见行从当前段文件复制到新的段文件，然后计划删除当前段文件。



并使新的段文件可用。追加优化表的普通VACUUM允许在压缩段文件的同时扫描，插入，删除和更新表。但是，将使用Access Exclusive锁短暂删除当前段文件并激活新的段文件。

VACUUM FULL进行更广泛的处理，包括在块之间移动元组以尝试将表压缩到最小数量的磁盘块。这种形式要慢得多，并且在处理每个表时都需要申请Access Exclusive锁。Access Exclusive锁可确保所有者是以任何方式访问表的唯一事务。

当选项列表用括号括起来时，可以按任何顺序写入选项。不带括号的选项必须严格按照上面显示的顺序指定。括号中的语法是在Greenplum数据库6.0中添加的；不带括号的语法已弃用。

Important: 有关使用VACUUM, VACUUM FULL和VACUUM ANALYZE的信息，请参阅[注解](#)。

输出

当指定了VERBOSE时，VACUUM发出进度消息以指示当前正在处理哪个表。还将打印有关表格的各种统计信息。

参数

FULL

选择full vacuum，这可以回收更多空间，但是需要更长的时间排他锁定表。此方法还需要额外的磁盘空间，因为它会写入表的新副本，并且在操作完成之前不会释放旧副本。通常，仅在需要从表中回收大量空间时才应使用此选项。

FREEZE

指定FREEZE等效于将vacuum_freeze_min_age服务器配置参数设置为零来执行VACUUM。请参阅[服务器配置参数](#)以获取有关vacuum_freeze_min_age的信息。

VERBOSE

为每个表打印详细的vacuum活动报告。

ANALYZE

更新优化器使用的统计信息，以确定执行查询的最快方法。

table

要vacuum的表的名称（可以用schema修饰）。默认为当前数据库中的所有表。

column

要分析的特定列的名称。默认为所有列。如果指定了列列表，则意味着ANALYZE。

注解

VACUUM无法在事务块内执行。

频繁（至少每晚一次）vacuum活跃数据库，以便删除过期的行。添加或删除大量行后，对受影响的表运行VACUUM ANALYZE命令可能会很有用。这将使用所有最近更改的结果来更新系统catalog，并使Greenplum数据库查询优化器可以在计划查询中做出更好的选择。

Important: PostgreSQL有一个单独的可选服务器进程，称为*autovacuum daemon*，其目的是自动执行VACUUM和ANALYZE命令。Greenplum数据库开启autovacuum守护程序仅在Greenplum数据库模板数据库template0上执行VACUUM操作。为template0启用了autovacuum，因为不允许连接到template0。autovacuum守护程序在template0上执行VACUUM操作以管理事务ID（XID），并帮助避免template0中的事务ID环绕问题。

必须在用户定义的数据库中执行手动VACUUM操作，以管理这些数据库中的事务ID（XID）。

VACUUM导致I/O流量大幅增加，这可能导致其他活动会话的性能下降。因此，建议在低使用率时vacuum数据库。

VACUUM命令跳过外部表。

VACUUM FULL回收所有过期的行空间，但是它需要对每个正在处理的表进行独占锁定，这是一项非常昂贵的操作，并且可能需要很长时间才能完成大型分布式Greenplum数据库表。在数据库维护期间执行VACUUM FULL操作。

不建议例行使用FULL选项，但在特殊情况下可能有用。例如，当您删除或更新了表中的大多数行，并希望该表在物理上缩小以占用更少的磁盘空间并允许更快的表扫描时。VACUUM FULL通常比普通VACUUM将表缩小更多。

作为VACUUM FULL的替代方法，您可以使用CREATE TABLE AS语句重新创建表并删除旧表。

对于附加优化表，VACUUM需要足够的可用磁盘空间以在VACUUM过程中容纳新的段文件。如果段文件中隐藏行与总行的比率小于阈值（默认为10），则不会压缩段文件。可以使用gp_appendonly_compaction_threshold服务器配置参数来配置阈值。VACUUM FULL忽略阈值并重写段文件，而不考虑比率。可以使用gp_appendonly_compaction服务器配置参数为附加优化表禁用VACUUM。有关服务器配置参数的信息，请参阅[服务器配置参数](#)。

数。

如果在清理附加优化表时检测到并发可序列化事务，则不会压缩当前和后续段文件。如果已压缩段文件，但是在删除原始段文件的事务中检测到并发可序列化事务，则将忽略该删除。清理完成后，这可能会使一个或两个段文件处于“等待删除”状态。

有关Greenplum数据库中并发控制的更多信息，请参阅*Greenplum*数据库管理员指南中的“例行系统维护任务”。

示例

要清理单个表onek，请对它进行优化分析并打印详细的vacuum活动报告：

```
VACUUM (VERBOSE, ANALYZE) onek;
```

清理当前数据库中的所有表：

```
VACUUM;
```

仅清理特定表：

```
VACUUM (VERBOSE, ANALYZE) mytable;
```

清理当前数据库中的所有表并收集查询优化器的统计信息：

```
VACUUM ANALYZE;
```

兼容性

SQL标准中没有VACUUM语句。

另见

[ANALYZE](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL Syntax Summary

ABORT

ALTER AGGREGATE

ALTER COLLATION

ALTER CONVERSION

ALTER DATABASE

ALTER DEFAULT PRIVILEGES

ALTER DOMAIN

ALTER EXTENSION

ALTER EXTERNAL TABLE

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FUNCTION

ALTER GROUP

ALTER INDEX

ALTER LANGUAGE

ALTER OPERATOR

计算一组行。

概要

```
VALUES ( expression [ , ... ] ) [ , ... ]
        [ ORDER BY sort_expression [ ASC | DESC |
USING operator ] [ , ... ]
        ]
        [ LIMIT { count | ALL } ]
        [ OFFSET start [ ROW | ROWS ] ]
        [ FETCH { FIRST | NEXT } [ count ] { ROW |
ROWS } ONLY ]
```

描述

VALUES计算值表达式指定的行值或行值集。它最常用于在较大的命令中生成“常数表”，但可以单独使用。

指定多于一行时，所有行必须具有相同数量的元素。通过使用与UNION相同的规则，组合出现在该列中的表达式的显式或推断类型，来确定结果表的列的数据类型。

在较大的命令中，语法上允许SELECT所在的任何位置使用VALUES。由于语法将其视为SELECT，因此可以将VALUES命令与ORDER BY, LIMIT (或等效的FETCH FIRST) 和OFFSET子句一起使用。

参数

expression

要计算并插入到结果表中指定位置（行集）的常量或表达式。在出现在INSERT顶层的VALUES列表中，可以将表达式替换为DEFAULT以指示应插入目标列的默认值。当VALUES出现在其他上下文中时，不能使用DEFAULT。

sort_expression

表示如何对结果行进行排序的表达式或整数常量。该表达式可以将VALUES结果的列称为column1, column2等。有关更多



详细信息，请参见[SELECT参数中的“ORDER BY子句”。](#)

operator

排序运算符。有关更多详细信息，请参见[SELECT参数中的“ORDER BY子句”。](#)

LIMIT count

OFFSET start

要返回的最大行数。有关更多详细信息，请参见[SELECT参数中的“LIMIT子句”。](#)

注解

应该避免使用具有大量行的VALUES列表，因为您可能会遇到内存不足的故障或性能不佳的情况。出现在INSERT中的VALUES是一种特殊情况（因为所需的列类型是从INSERT的目标表中知道的，不需要通过扫描VALUES列表来推断），因此它可以处理比其他情况下实际更大的列表。

示例

一条简单的VALUES命令：

```
VALUES (1, 'one'), (2, 'two'), (3, 'three');
```

这将返回一个两列三行的表。它实际上等效于：

```
SELECT 1 AS column1, 'one' AS column2
UNION ALL
SELECT 2, 'two'
UNION ALL
SELECT 3, 'three';
```

通常，在较大的SQL命令中使用VALUES。最常见的用法是在INSERT中：

```
INSERT INTO films (code, title, did, date_prod, kind)
    VALUES ('T_601', 'Yojimbo', 106, '1961-06-16', 'Drama');
```

在INSERT的上下文中，VALUES列表的条目可以为DEFAULT，以指示此处应使用列默认值而不是指定值：

```
INSERT INTO films VALUES
    ('UA502', 'Bananas', 105, DEFAULT,
     'Comedy', '82
               minutes'),
    ('T_601', 'Yojimbo', 106, DEFAULT, 'Drama',
     DEFAULT);
```

VALUES也可以用于可能写入sub-SELECT的地方，例如在FROM子句中：

```
SELECT f.* FROM films f, (VALUES('MGM', 'Horror'), ('UA',
'Sci-Fi')) AS t (studio, kind) WHERE f.studio = t.studio
AND
f.kind = t.kind;
UPDATE employees SET salary = salary * v.increase FROM
(VALUES(1, 200000, 1.2), (2, 400000, 1.4)) AS v (depno,
target, increase) WHERE employees.depno = v.depno AND
employees.sales >= v.target;
```

请注意，在FROM子句中使用VALUES时，需要AS子句，就像SELECT一样。不需要AS子句为所有列指定名称，但是这样做是一种好习惯。在Greenplum数据库中，VALUES的默认列名称为column1, column2等，但是在其他数据库系统中，这些名称可能不同。

在INSERT中使用VALUES时，所有值都将自动强制转换为相应目标列的数据类型。在其他上下文中使用它时，可能有必要指定正确的数据类型。如果所有条目都是用引号括起来的文字常量，则强制第一个足以确定所有假定的类型：

```
SELECT * FROM machines WHERE ip_address IN
(VALUES('192.168.0.1'::inet), ('192.168.0.10'),
('192.0.2.43'));
```

Note: 对于简单的IN测试，最好依靠IN的标量列表形式，而不要像上面所示编写VALUES查询。标量方法列表需要较少的编写，并且通常效率更高。

兼容性

VALUES符合SQL标准。LIMIT和OFFSET是Greenplum数据库扩展；另请参见[SELECT](#)下的内容。

另见

[INSERT , SELECT](#)

Parent topic: [SQL Command Reference](#)

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

使用外部表装载数据

□ 装载和写入非HDFS自
定义数据

□ 处理装载错误

用gupload装载数据

使用PXF访问外部数据

使用gpfdist和gupload转
换外部数据

用COPY装载数据

在单行错误隔离模式中
运行COPY优化数据装载和查询性
能

有两种保留字符对于Greenplum数据库具有特殊含义：

- 指派的用来在数据文件中分隔列或者域的分隔字符。
- 在数据文件中指定一个新行的新行字符。

如果用户的数据包含这些字符，用户必须转义它们这样Greenplum才会把它们当作数据来对待而不是一个域定界符或者新行。默认情况下，文本格式文件的转义字符是一个\（反斜线）而CSV格式文件的转义字符是一个双引号（"）。

- [在文本格式的文件中转义](#)
- [在CSV格式的文件中转义](#)

Parent topic: [格式化数据文件](#)



Greenplum数据库® 6.0 文档

□ 参考指南

Greenplum database 5 管理员指南

Greenplum database 4 管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

- [gp_configuration_history](#)
- [gp_distribution_policy](#)
- [gp_fastsequence](#)
- [gp_global_sequence](#)
- [gp_id](#)
- [gp_stat_replication](#)
- [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 \(not supported\)](#)
- [pg_attrdef](#)
- [pg_attribute](#)
- [pg_auth_members](#)
- [pg_authid](#)
- [pg_autovacuum \(not supported\)](#)
- [pg_cast](#)
- [pg_class](#)
- [pg_constraint](#)
- [pg_conversion](#)
- [pg_database](#)
- [pg_db_role_setting](#)
- [pg_depend](#)
- [pg_description](#)
- [pg_exttable](#)
- [pg_foreign_data_wrapper](#)



- [pg_foreign_server](#)
- [pg_foreign_table](#)
- [pg_index](#)
- [pg_inherits](#)
- [pg_language](#)
- [pg_largeobject](#)
- [pg_listener](#)
- [pg_namespace](#)
- [pg_opclass](#)
- [pg_operator](#)
- [pg_opfamily](#)
- [pg_partition](#)
- [pg_partition_rule](#)
- [pg_pltemplate](#)
- [pg_proc](#)
- [pg_resgroup](#)
- [pg_resgroupcapability](#)
- [pg_resourcetype](#)
- [pg_resqueue](#)
- [pg_resqueuecapability](#)
- [pg_rewrite](#)
- [pg_shdepend](#)
- [pg_shdescription](#)
- [pg_stat_last_operation](#)
- [pg_stat_last_shoperation](#)
- [pg_stat_replication](#)
- [pg_statistic](#)
- [pg_tablespace](#)
- [pg_trigger](#)
- [pg_type](#)
- [pg_user_mapping](#)

Parent topic: [系统目录参考](#)

Greenplum数据库® 6.0文档

□ 参考指南

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

Greenplum数据库提供了如下在PostgreSQL中不存在的系统视图。

- [gp_distributed_log](#)
- [gp_distributed_xacts](#)
- [gp_pgdatabase](#)
- [gp_resgroup_config](#)
- [gp_resgroup_status](#)
- [gp_resgroup_status_per_host](#)
- [gp_resgroup_status_per_segment](#)
- [gp_resqueue_status](#)
- [gp_transaction_log](#)
- [gpexpand.expansion_progress](#)
- [pg_max_external_files](#)
- [pg_partition_columns](#)
- [pg_partition_templates](#)
- [pg_partitions](#)
- [pg_resqueue_attributes](#)
- [pg_resqueue_status](#) (弃用。使用[gp_toolkit.gp_resqueue_status](#)。)
- [pg_stat_activity](#)
- [pg_stat_replication](#)
- [pg_stat_resqueues](#)
- [session_level_memory_consumption](#)(见*Greenplum*数据库管理员指南中的“查看会话内存使用信息”。)

有关PostgreSQL和Greenplum数据库支持的标准系统视图的更多信息，请参阅PostgreSQL文档的以下部分：

- [系统视图](#)
- [统计收集器视图](#)
- [信息模式](#)

Parent topic: [系统目录参考](#)



Greenplum数据库® 6.0文档

□ 参考指南

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

系统目录表和视图按字母顺序定义。

- [**foreign_data_wrapper_options**](#)
- [**foreign_data_wrappers**](#)
- [**foreign_server_options**](#)
- [**foreign_servers**](#)
- [**foreign_table_options**](#)
- [**foreign_tables**](#)
- [**gp_configuration_history**](#)
- [**gp_distributed_log**](#)
- [**gp_distributed_xacts**](#)
- [**gp_distribution_policy**](#)
- [**gpexpand.expansion_progress**](#)
- [**gpexpand.status**](#)
- [**gpexpand.status_detail**](#)
- [**gp_fastsequence**](#)
- [**gp_id**](#)
- [**gp_pgdatabase**](#)
- [**gp_resgroup_config**](#)
- [**gp_resgroup_status**](#)
- [**gp_resgroup_status_per_host**](#)
- [**gp_resgroup_status_per_segment**](#)
- [**gp_resqueue_status**](#)
- [**gp_stat_replication**](#)
- [**gp_segment_configuration**](#)
- [**gp_transaction_log**](#)
- [**gp_version_at_initdb**](#)
- [**pg_aggregate**](#)
- [**pg_am**](#)
- [**pg_amop**](#)
- [**pg_amproc**](#)
- [**pg_appendonly**](#)



- [pg_attrdef](#)
- [pg_attribute](#)
- [pg_attribute_encoding](#)
- [pg_auth_members](#)
- [pg_authid](#)
- [pg_available_extension_versions](#)
- [pg_available_extensions](#)
- [pg_cast](#)
- [pg_class](#)
- [pg_compression](#)
- [pg_constraint](#)
- [pg_conversion](#)
- [pg_database](#)
- [pg_db_role_setting](#)
- [pg_depend](#)
- [pg_description](#)
- [pg_enum](#)
- [pg_extension](#)
- [pg_exttable](#)
- [pg_foreign_data_wrapper](#)
- [pg_foreign_server](#)
- [pg_foreign_table](#)
- [pg_index](#)
- [pg_inherits](#)
- [pg_language](#)
- [pg_largeobject](#)
- [pg_listener](#)
- [pg_locks](#)
- [pg_max_external_files](#)
- [pg_namespace](#)
- [pg_opclass](#)
- [pg_operator](#)

- [pg_opfamily](#)
- [pg_partition](#)
- [pg_partition_columns](#)
- [pg_partition_encoding](#)
- [pg_partition_rule](#)
- [pg_partition_templates](#)
- [pg_partitions](#)
- [pg_pltemplate](#)
- [pg_proc](#)
- [pg_resgroup](#)
- [pg_resgroupcapability](#)
- [pg_resourcetype](#)
- [pg_resqueue](#)
- [pg_resqueue_attributes](#)
- [pg_resqueuecapability](#)
- [pg_rewrite](#)
- [pg_roles](#)
- [pg_shdepend](#)
- [pg_shdescription](#)
- [pg_stat_activity](#)
- [pg_stat_last_operation](#)
- [pg_stat_last_shoperation](#)
- [pg_stat_operations](#)
- [pg_stat_partition_operations](#)
- [pg_stat_replication](#)
- [pg_statistic](#)
- [pg_stat_resqueues](#)
- [pg_tablespace](#)
- [pg_trigger](#)
- [pg_type](#)
- [pg_type_encoding](#)
- [pg_user_mapping](#)

[pg_user_mappings](#)

- [user_mapping_options](#)
- [user_mappings](#)

Parent topic: [系统目录参考](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign_data_wrapper_options

foreign_data_wrappers

foreign_server_options

foreign_servers

foreign_table_options

foreign_tables

gp_configuration_history

gp_distributed_log

gp_distributed_xacts

gp_distribution_policy

gpexpand.expansion_progress

gpexpand.status

`gp_toolkit.gp_resgroup_status_per_host`视图允许管理员基于每个主机查看每个资源组的当前内存和CPU使用率和分配。

内存量以MB为单位指定。

Note: 仅当基于资源组的资源管理处于活动状态时, `gp_resgroup_status_per_host`视图才有效。

Table 1. `gp_toolkit.gp_resgroup_status_per_host`

列	类型	参考	描述
rsgname	name	<code>pg_resgroup.rsgname</code>	资源组的名称。
groupid	oid	<code>pg_resgroup.oid</code>	资源组的ID。
hostname	text	<code>gp_segment_configuration.hostname</code>	segment主机的主机名。
cpu	numeric		主机上资源组的实时CPU使用情况。
memory_used	integer		主机上资源组的实时内存使用情况。此总数包括资源组固定和共享内存。它还包括资源组使用的全局共享内存。
memory_available	integer		主机上可用的资源组的未使用的固定和共享内存。此总计不包括可用资源组全局共享内存。
memory_quota_used	integer		主机上资源组的实时固定内存使用情况。
memory_quota_available	integer		主机上资源组可用的固定内存。
memory_quota_proposed	integer		Greenplum数据库分配给主机上资源组的固定内存总量。
memory_shared_used	integer		该组共享主机上资源组使用的内存。如果资源组使用任

			何全局共享内存，则此数量也包含在总计中。
memory_shared_available	integer		主机上资源组可用的组共享内存量。资源组全局共享内存不包含在此总计中。
memory_shared_granted	integer		Greenplum数据库分配给主机上资源组的组共享内存部分。资源组全局共享内存不包含在此值中。
memory_shared_proposed	integer		主机上资源组请求的组共享内存总量。

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign_data_wrapper_options

foreign_data_wrappers

foreign_server_options

foreign_servers

foreign_table_options

foreign_tables

gp_configuration_history

gp_distributed_log

gp_distributed_xacts

gp_distribution_policy

gpexpand.expansion_progress

gpexpand.status

`gp_toolkit.gp_resgroup_status_per_segment`视图允许管理员基于每个主机和每个segment查看每个资源组的当前内存和CPU使用率和分配。

内存量以MB为单位指定。

Note: 仅当基于资源组的资源管理处于活动状态时, `gp_resgroup_status_per_segment`视图才有效。

Table 1. `gp_toolkit.gp_resgroup_status_per_segment`

列	类型	参考	描述
rsgname	name	<code>pg_resgroup.rsgname</code>	资源组的名称。
groupid	oid	<code>pg_resgroup.oid</code>	资源组的ID。
hostname	text	<code>gp_segment_configuration.hostname</code>	segment主机的主机名。
segment_id	smallint	<code>gp_segment_configuration.content</code>	segment主机上的segment实例的内容ID。
cpu	numeric		主机上的segment实例的资源组的实时CPU使用情况。
memory_used	integer		主机上的segment实例的资源组的实时内存使用情况。此总数包括资源组固定和共享内存。它还包括资源组使用的全局共享内存。
memory_available	integer		主机上的segment实例的资源组的未使用的固定和共享内存。
memory_quota_used	integer		主机上的segment实例的资源组的实时固定内存使用情况。
memory_quota_available	integer		主机上的segment实例的资源组可用的固定内存。
memory_quota_proposed	integer		Greenplum数据库分配给主机上的segment的资源组的固定内存总量。对于主机上的所有资源组和segment实例组合, 此值将相同。
memory_shared_used	integer		该组共享资源组用于主机上segment

			的实例的内存。
memory_shared_available	integer		主机上的segment实例可用的组共享内存量。资源组全局共享内存不包含在此总计中。
memory_shared_granted	integer		组共享内存的部分，Greenplum数据库已分配给主机上的segment实例的资源组。资源组全局共享内存不包含在此值中。
memory_shared_proposed	integer		主机上资源组请求的组共享内存总量。对于主机上的所有资源组和segment实例组合，此值将相同。

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

database_* 表

diskspace_*

interface_stats_*

log_alert_*

queries_*

segment_*

socket_stats_*

system_*

dynamic_memory_info
视图

memory_info 视图

□ Greenplum 数据库数据类型

字符集支持

database_* 表存储一个 Greenplum 数据库实例的工作负载信息。这里一共有三张表，每张表都具有相同的结构(列):

- database_now 是一个外部表，其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 从数据采集代理程序获得数据以后，自动提交到 database_history 表之前，当前查询工作负载数据存储在 database_now 表中。
- database_tail 是一个外部表，其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 它是一个过渡表，当数据已经从 database_now 中清除，但还没有提交到 database_history 表中时，暂存在这里。它通常仅包含数据几分钟时间。
- database_history 是一个常规表，用于存储历史查询工作负载数据。它已预先设置为按月分区。分区会根据需要以两个月为增量自动添加。

列	类型	说明
ctime	timestamp	该行的创建时间.
queries_total	int	采集数据时，Greenplum 数据库中的查询总数量.
queries_running	int	采集数据时，活动的查询数量.
queries_queued	int	采集数据时，资源组或资源队列中处于等待状态的查询数量(与当前使用的资源管理器相关).

Parent topic: [gpperfmon 数据库](#)



Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

database_* 表

diskspace_* 表

interface_stats_* 表

log_alert_* 表

queries_* 表

segment_* 表

socket_stats_* 表

system_* 表

dynamic_memory_info
视图

memory_info 视图

□ Greenplum 数据库数据类型

字符集支持

diskspace_* 表存储磁盘空间指标。

- diskspace_now 是一个外部表, 其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 在数据从 gpperfmon 数据采集代理程序获得数据以后, 自动提交到 diskspace_history 表之前, 当前磁盘空间指标数据存储在 database_now 表中。
- diskspace_tail 是一个外部表, 其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 它是一个过渡表, 当数据已经从 diskspace_now 中清除, 但还没有提交到 diskspace_history 表中时, 暂存在这里。它通常仅包含数据几分钟时间。
- diskspace_history 是一个常规表, 用于存储历史磁盘空间指标。它已预先设置为按月分区。分区会根据需要以两个月为增量自动添加。

列	类型	说明
ctime	timestamp(0) without time zone	磁盘空间测量时间.
hostname	varchar(64)	磁盘空间测量对应的主机名称.
Filesystem	text	磁盘空间测量对应的文件系统名称.
total_bytes	bigint	文件系统的总字节数.
bytes_used	bigint	文件系统中已使用字节数.
bytes_available	bigint	文件系统中可用的字节数.

Parent topic: [gpperfmon 数据库](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

database_* 表

diskspace_* 表

interface_stats_* 表

log_alert_* 表

queries_* 表

segment_* 表

socket_stats_* 表

system_* 表

dynamic_memory_info

视图

memory_info 视图

□ Greenplum 数据库数据类型

字符集支持

log_alert_* 表存储 pg_log 错误和警报。

参考 [警报日志处理和日志轮换](#) 了解关于配置 gpperfmon 系统 logger 的更多信息。

一共有三张 log_alert 表, 所有这些表具有相同的列:

- log_alert_now 是一个外部表, 它的数据存储在 \$MASTER_DATA_DIRECTORY/gpperfmon/logs 目录下的 .csv 文件中。在数据从 gpperfmon 代理自动提交到 log_alert_history 表期间, 当前的 pg_log 错误和警报数据在 log_alert_now 中。.
- log_alert_tail 是一个外部表, 它的数据存储在 \$MASTER_DATA_DIRECTORY/gpperfmon/logs/alert_log_stage 中。这是一个过渡表, 当数据已经从 log_alert_now 中清除, 但还没有提交到 log_alert_history 中时, 暂存在这里。该表包括所有警报日志中的记录, 但最新的除外。它通常仅包含数据几分钟时间。
- log_alert_history 是一个常规表, 用于存储数据库范围内的历史错误和警告数据。它已预先设置为按月分区。分区会根据需要以两个月为增量自动添加。

列	类型	说明
logtime	timestamp with time zone	此日志的时间戳
loguser	text	查询的用户
logdatabase	text	查询的用户
logpid	text	进程 ID
logthread	text	线程号
loghost	text	主机名或 IP 地址
logport	text	端口号
logsessiontime	timestamp with time zone	会话时间戳
logtransaction	integer	事务 ID
logsession	text	会话 ID
logcmdcount	text	命令数量
logsegment	text	Segment 号
logslice	text	Slice 编号
logdistxact	text	分布式事务
loglocalxact	text	本地事务
logsubxact	text	子事务

logseverity	text	日志级别
logstate	text	日志状态
logmessage	text	日志信息
logdetail	text	详细信息
loghint	text	提示信息
logquery	text	查询内容
logquerypos	text	查询位置
logcontext	text	上下文信息
logdebug	text	调试
logcursorpos	text	光标位置
logfunction	text	函数信息
logfile	text	源代码文件
logline	text	源代码行
logstack	text	堆栈信息

Parent topic: [gpperfmon 数据库](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

database_* 表

diskspace_* 表

interface_stats_* 表

log_alert_* 表

queries_* 表

segment_* 表

socket_stats_* 表

system_* 表

dynamic_memory_info
视图

memory_info 视图

□ Greenplum 数据库数据类型

字符集支持

queries_* 系列表存储高级查询状态信息.

tmid, ssid 和 ccnt 列是唯一标识特定查询的组合键.

一共有三张查询表，所有查询表都具有相同的列:

- queries_now 是一个外部表, 其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 在数据从 gpperfmon 代理自动提交到 queries_history 表期间, 当前查询状态存储在 queries_now 中。
- queries_tail 是一个外部表, 其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 它是一个过渡表。当数据已经从 queries_now 中清除, 但还没有提交到 queries_history 时, 查询状态数据保存在这里。它通常仅包含数据几分钟时间.
- queries_history 是存储历史查询状态数据的常规表。它被预定义为按月进行分区. 分区会根据需要以两个月为增量自动添加。

列	类型	说明
ctime	timestamp	该行的创建时间.
tmid	int	特定查询的时间标识符. 与特定查询关联的所有记录具有相同的 tmid .
ssid	int	与 gp_session_id 关联的会话 ID. 与特定查询关联的所有记录具有相同的 ssid .
ccnt	int	与 gp_command_count 关联的命令号. 与特定查询关联的所有记录具有相同的 ccnt .
username	varchar(64)	发出查询的 Greenplum 角色名称.
db	varchar(64)	查询的数据库名称.
cost	int	在此版本中未实现
tsubmit	timestamp	查询的提交时间.
tstart	timestamp	查询的开始时间.

tfinish	timestamp	查询的结束时间.
status	varchar(64)	查询状态 -- start , done , or abort .
rows_out	bigint	查询输出的行数.
cpu_elapsed	bigint	<p>执行该查询的全部 segments 的全部进程的 CPU 时间(单位: 秒). 它是从数据库系统中所有活动的主 segments 获取的 CPU 时间总和。</p> <p>请注意, 如果查询运行时间短于 quantum 的值, 则该值记录为 0。即使查询运行时间大于 min_query_time 值, 但是低于 quantum 的值, 也会记录为 0。</p>
cpu_currpct	float	<p>当前执行此查询的所有进程的 CPU 平均百分比。对在每个 segment 上运行的所有进程的百分比求平均值, 然后计算所有这些值的平均值来获得该指标。</p> <p>当前的 CPU 平均百分比在 history 数据和 tail 数据中始终为零。</p>
skew_cpu	float	显示该查询在系统中的处理偏斜量。当一个 segment 对查询执行的处理量不成比例时, 就会发生处理 / CPU 偏斜。此值是此查询所有 segment 中 CPU% 度量的方差系数乘以 100。例如, .95 的值表示为 95。
skew_rows	float	显示系统中的行偏斜量。当一个 segment 产生的查询行数不成比例时, 就会发生行偏斜。该值是该查询所有

		segment 的 rows_in 指标的方差系数乘以100。例如，.95 的值显示为 95。
query_hash	bigint	在此版本中未实现.
query_text	text	该查询的 SQL 文本.
query_plan	text	查询计划文本. 在此版本中未实现.
application_name	varchar(64)	应用程序的名称.
rsqname	varchar(64)	如果基于资源队列的资源管理方案被使用，则此列为资源队列的名称.
rqppriority	varchar(64)	如果基于资源队列的资源管理方案被使用，则此列为查询优先级 -- max, high, med, low, 或 min.

Parent topic: [gpperfmon 数据库](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

database_* 表

diskspace_* 表

interface_stats_* 表

log_alert_* 表

queries_* 表

segment_* 表

socket_stats_* 表

system_* 表

dynamic_memory_info 视图

memory_info 视图

□ Greenplum 数据库数据类型

字符集支持

The `segment_*` 包含 Greenplum 数据库 segment 实例的内存分配统计信息。它跟踪每个特定 segment 实例上全部 postgres 进程的内存消费数量, 以及由当前资源管理方案(基于资源组或基于资源队列)设置的每个 segment 可用的剩余内存数量. 参见 *Greenplum* 数据库管理员指南 了解更多关于资源管理方案的详细信息.

一共有三张 segment 表, 每张表都具有相同的结构(列):

- `segment_now` 是一个外部表, 其数据文件位于 `$MASTER_DATA_DIRECTORY/gpperfmon/data`. 在数据从 gpperfmon 数据采集代理程序获得以后, 自动提交到 `segment_history` 表之前, 当前系统使用指标数据存储在 `segment_now` 表中。
- `segment_tail` 是一个外部表, 其数据文件位于 `$MASTER_DATA_DIRECTORY/gpperfmon/data`. 它是一个内存分配信息的过渡表, 当数据已经从 `segment_now` 中清除, 但还没有提交到 `segment_history` 表中时, 暂存在这里。它通常仅包含数据几分钟时间。
- `segment_history` 是一个常规表, 用于存储历史内存分配指标。它已预先设置为按月分区。分区会根据需要以两个月为增量自动添加。

每个特定 segment 实例通过 `hostname` 和 `dbid` (`gp_segment_configuration` 系统目录表中的 segment 唯一标识) 定义.

列	类型	说明
<code>ctime</code>	<code>timestamp(0)</code> (without time zone)	该行的创建时间.
<code>dbid</code>	<code>int</code>	segment ID (<code>gp_segment_configuration</code> 中的 <code>dbid</code>).
<code>hostname</code>	<code>charvar(64)</code>	segment 主机名称.
<code>dynamic_memory_used</code>	<code>bigint</code>	当前 segment 上分配给查询进程的动态内存数量(单位: 字节).
<code>dynamic_memory_available</code>	<code>bigint</code>	在到达当前资源管理方案(基于资源组或基于资源队列)设置限制前, segment 上还可分配的动态内存数量(单位: 字节).

参见 `memory_info` 视图和 `dynamic_memory_info` 视图了解更多主机上内存分配汇总和使用信息.

Parent topic: [gpperfmon 数据库](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

database_* 表

diskspace_* 表

interface_stats_* 表

log_alert_* 表

queries_* 表

segment_* 表

socket_stats_* 表

system_* 表

dynamic_memory_info
视图

memory_info 视图

□ Greenplum 数据库数据类型

字符集支持

socket_stats_* 表存储一个 Greenplum 数据库实例中 socket 使用统计指标。一共有三张表，所有查询表都具有相同的列：

这些表为将来使用保留，当前没有填充信息。

- `socket_stats_now` 是一个外部表，它的数据存储在 `$MASTER_DATA_DIRECTORY/gpperfmon/data`.
- `socket_stats_tail` 是一个外部表，它的数据存储在 `$MASTER_DATA_DIRECTORY/gpperfmon/data`. 这是一个过渡表，当数据已经从 `socket_stats_now` 中清除，但还没有提交到 `socket_stats_history` 中时，暂存在这里。它通常仅包含数据几分钟时间。
- `socket_stats_history` 是一个常规表，用于存储 socket 历史统计指标。它已预先设置为按月分区。分区会根据需要以两个月为增量自动添加。

列	类型	说明
<code>total_sockets_used</code>	int	系统中的 socket 总数.
<code>tcp_sockets_inuse</code>	int	使用中的 TCP socket 数量.
<code>tcp_sockets_orphan</code>	int	孤儿 TCP socket 数量.
<code>tcp_sockets_timewait</code>	int	Time-Wait 状态的 TCP socket 数量.
<code>tcp_sockets_alloc</code>	int	已分配的 TCP socket 数量.
<code>tcp_sockets_memusage_inbytes</code>	int	TCP socket 消耗的内存总量.
<code>udp_sockets_inuse</code>	int	使用中的 UDP socket 数量.
<code>udp_sockets_memusage_inbytes</code>	int	UDP socket 消耗的内存总量.
<code>raw_sockets_inuse</code>	int	使用中  RAW socket
<code>frag_sockets_inuse</code>	int	使用中的 FRAG socket 数量.

frag_sockets_memusage_inbytes	int	Frag socket 消耗的内存总量.
-------------------------------	-----	----------------------

Parent topic: [gpperfmon 数据库](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

database_* 表

diskspace_* 表

interface_stats_* 表

log_alert_* 表

queries_* 表

segment_* 表

socket_stats_* 表

system_* 表

dynamic_memory_info
视图

memory_info 视图

□ Greenplum 数据库数据类型

字符集支持

system_* 表存储系统使用指标。这里一共有三张表，每张表都具有相同的结构(列):

- system_now 是一个外部表, 其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 在数据从 gpperfmon 数据采集代理程序获得以后, 自动提交到 system_history 表之前, 当前系统使用指标数据存储在 system_now 表中。
- system_tail 是一个外部表, 其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 它是一个过渡表, 当数据已经从 system_now 中清除, 但还没有提交到 system_history 表中时, 暂存在这里。它通常仅包含数据几分钟时间。
- system_history 是一个常规表, 用于存储历史系统使用指标。它已预先设置为按月分区。分区会根据需要以两个月为增量自动添加。

列	类型	说明
ctime	timestamp	该行的创建时间.
hostname	varchar(64)	与系统指标相关的 Segment 或 master 主机名称.
mem_total	bigint	主机全部系统内存(单位: 字节).
mem_used	bigint	主机已使用的系统内存(单位: 字节).
mem_actual_used	bigint	主机实际已使用的内存(单位: 字节) (不包括缓存和缓冲保留内存).
mem_actual_free	bigint	主机空闲内存(单位: 字节) (不包括缓存和缓冲保留内存).
swap_total	bigint	主机全部交换内存(单位: 字节).
swap_used	bigint	主机已使用的交换内存(单位: 字节).
swap_page_in	bigint	交换进的页数量.
swap_page_out	bigint	交换出的页数量.
cpu_user	float	Greenplum 系统用户的 CPU

		使用量·
cpu_sys	float	主机 CPU 使用量·
cpu_idle	float	指标收集时空闲的 CPU 容量·
load0	float	前一分钟的 CPU 平均负载·
load1	float	前五分钟的 CPU 平均负载·
load2	float	前十五分钟的 CPU 平均负载·
quantum	int	此指标的指标采集间隔·
disk_ro_rate	bigint	每秒磁盘读取操作次数·
disk_wo_rate	bigint	每秒磁盘写入操作次数·
disk_rb_rate	bigint	磁盘每秒读取的字节数·
disk_wb_rate	bigint	磁盘每秒写入的字节数·
net_rp_rate	bigint	系统网络每秒读取的报文数量·
net_wp_rate	bigint	系统网络每秒写入的报文数量·
net_rb_rate	bigint	系统网络每秒读取的字节数·
net_wb_rate	bigint	系统网络每秒写入的字节数·

Parent topic: [gpperfmon 数据库](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

database_* 表

diskspace_* 表

interface_stats_* 表

log_alert_* 表

queries_* 表

segment_* 表

socket_stats_* 表

system_* 表

dynamic_memory_info
视图

memory_info 视图

□ Greenplum 数据库数据类型

字符集支持

dynamic_memory_info 视图展示一个 segment 主机上所有 segment 实例已用内存和可用动态内存的总和。 动态内存指 Greenplum 数据库实例在启动取消进程前，允许一个 segment 实例上查询进程使用的内存(超过该数量则启动取消进程)。 这个限制有当前所使用的资源管理方案决定 (基于资源组或基于资源队列)，按每个 segment 进行评估。

列	类型	说明
ctime	timestamp(0) without time zone	在 segment_history 表中的创建时间.
hostname	varchar(64)	与这些系统内存指标 相关的 Segment 或 master 主机名称.
dynamic_memory_used_mb	numeric	segment 节点上分配 给查询进程的动态内 存大小(单位: MB).
dynamic_memory_available_mb	numeric	segment 节点上可分 配给查询进程的动态 内存大小(单位: MB). 注意这个值是所有 segment 节点上可分 配内存的总和. 虽然 报告了这么多可分配 内存，也可能一个 或多个 segment 在主 机上分配的内存已经 超过了它们的限制。

Parent topic: [gpperfmon 数据库](#)



Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

database_* 表

diskspace_* 表

interface_stats_* 表

log_alert_* 表

queries_* 表

segment_* 表

socket_stats_* 表

system_* 表

dynamic_memory_info
视图

memory_info 视图

□ Greenplum 数据库数据类型

字符集支持

memory_info 视图展示 system_history 表和 segment_history 表中的每个主机上的内存信息。它允许管理员比较 segment 主机上的总可以内存, 总使用内存, 以及查询进程已使用的动态内存.

列	类型	说明
ctime	timestamp(0) without time zone	segment_history 表中该行数据的创建 时间.
hostname	varchar(64)	这些系统内存指标相 关的 Segment or master 主机名称.
mem_total_mb	numeric	segment 主机上的总 系统内存(单位: MB).
mem_used_mb	numeric	segment 主机上的已 使用内存(单位: MB).
mem_actual_used_mb	numeric	segment 主机上的实 际使用内存(单位: MB).
mem_actual_free_mb	numeric	segment 主机上的实 际可用内存(单位: MB).
swap_total_mb	numeric	segment 主机上的总 交换内存(单位: MB).
swap_used_mb	numeric	segment 主机上的已 使用交换内存(单位: MB).
dynamic_memory_used_mb	numeric	segment 主机上分配 给查询进程的动态内 存(单位: MB).
dynamic_memory_available_mb	numeric	segment 主机上查询 进程还可分配的动态 内存(单位: MB). 注意 这个值是一个主机 上所有 segment 的可 用内存, 即使报告为可 用内存, 也可能主机上 的一个或多个 segment 已 经超过了它们自身的内 存限制.

Parent topic: [gpperfmon 数据库](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

database_* 表

diskspace_* 表

interface_stats_* 表

log_alert_* 表

queries_* 表

segment_* 表

socket_stats_* 表

system_* 表

dynamic_memory_info
视图

memory_info 视图

□ Greenplum 数据库数据类型

字符集支持

`interface_stats_*` 表存储一个 Greenplum 数据库实例在每个活动网络接口上的通信统计指标。

这些表为将来使用保留，当前没有填充信息。

一共有三张 `interface_stats` 表，它们具有相同的结构(列):

- `interface_stats_now` 是一个外部表, 其数据文件位于 `$MASTER_DATA_DIRECTORY/gpperfmon/data`.
- `interface_stats_tail` 是一个外部表, 其数据文件位于 `$MASTER_DATA_DIRECTORY/gpperfmon/data`. 它是一个存储接口统计指标的过渡表, 当数据已经从 `interface_stats_now` 中清除, 但还没有提交到 `interface_stats_history`表中时, 暂存在这里。它通常仅包含数据几分钟时间。
- `interface_stats_history` 是一个常规表, 用于存储历史接口统计指标。它已预先设置为按月分区。分区会根据需要以两个月为增量自动添加。

列	类型	说明
<code>interface_name</code>	<code>string</code>	接口名称. 例如: eth0, eth1, lo.
<code>bytes_received</code>	<code>bigint</code>	接收字节数.
<code>packets_received</code>	<code>bigint</code>	接收报文数.
<code>receive_errors</code>	<code>bigint</code>	接收数据时, 发生的错误数.
<code>receive_drops</code>	<code>bigint</code>	接收数据时, 丢弃报文的次数.
<code>receive_fifo_errors</code>	<code>bigint</code>	接收数据时, FIFO (先进先出) 错误发生的次数.
<code>receive_frame_errors</code>	<code>bigint</code>	接收数据时, 帧错误发生的次数.
<code>receive_compressed_packets</code>	<code>int</code>	接受的压缩格式报文数.
<code>receive_multicast_packets</code>	<code>int</code>	接受的多播报文数.
<code>bytes_transmitted</code>	<code>bigint</code>	传输字节数.
<code>packets_transmitted</code>	<code>bigint</code>	传输报文数.

transmit_errors	bigint	数据传输时, 发生的错误数.
transmit_drops	bigint	数据传输时, 丢弃报文的次数.
transmit_fifo_errors	bigint	数据传输时, FIFO (先进先出) 错误发生的次数.
transmit_collision_errors	bigint	数据传输时, 碰撞错误发生的次数.
transmit_carrier_errors	bigint	数据传输时, 载波错误发生的次数.
transmit_compressed_packets	int	传输的压缩格式报文数.

Parent topic: [gpperfmon 数据库](#)

Greenplum数据库® 6.0文档

- [参考指南](#)
- [SQL Command Reference](#)
- [SQL 2008可选特性兼容性](#)
- [Greenplum环境变量](#)
- [保留标识符和SQL关键字](#)
- [系统目录参考](#)
- [gp_toolkit管理模式](#)
- [gpperfmon 数据库](#)
- [Greenplum 数据库数据类型](#)
- [字符集支持](#)
- [服务器配置参数](#)
- [内置函数摘要](#)
- [Greenplum MapReduce规范](#)
- [Greenplum PL/pgSQL过程语言](#)
- [Greenplum PL/R 语言扩展](#)
- [Greenplum PL/Python语言扩展](#)
- [Greenplum PL/Container语言扩展](#)
- [Greenplum的PL/Java语言扩展](#)
- [Greenplum的PL/Perl语言扩展](#)
- [用于分析的Greenplum MADlib扩展](#)
- [附加提供的模块](#)

citext

`hstore`模块实现了一种数据类型，用于在单个Greenplum数据库数据字段中存储（键，值）对的集合。这在各种情况下很有用，例如具有许多很少检查的属性的行或半结构化数据。

在当前的实现中，键和值字符串的长度都不能超过65535个字节。如果超出此限制，将引发错误。这些最大长度可能会在将来的版本中更改。

安装`hstore`

在使用`hstore`数据类型和函数之前，请在要查询其他数据库的每个数据库中运行安装脚本`$GPHOME/share/postgresql/contrib/hstore.sql`：

```
$ psql -d testdb -f $GPHOME/share/postgresql/contrib/hstore.sql
```

`hstore`外部表示

`hstore`值的文本表示形式包括零个或多个`key=>value`项，以逗号分隔。例如：

```
k => v
foo => bar, baz => whatever
"1-a" => "anything at all"
```

项目的顺序不重要（并且可能不会在输出中复制）。项目之间或`=>`符号周围的空格将被忽略。如果键或值包含空格，逗号，`=`或`>`，请使用双引号。要在键或值中包含双引号或反斜杠，请在其前面加上另一个反斜杠。（请记住，根据`standard_conforming_strings`的设置，您可能需要在SQL文字字符串中加双反斜杠。）值（但不是键）可以是SQL NULL。这表示为

```
key => NULL
```

NULL关键字不区分大小写。同样，如果要将字符串null视为普通数据值，请使用双引号引起来。

当前，即使不是绝对必要，双引号始终用于在输出中包含键和值字符串。

hstore运算符和函数

Table 1. hstore运算符

运算符	描述	示例	结果
<code>hstore -> text</code>	获取键的值（如果不存在，则为null）	'a=>x, b=>y'::hstore -> 'a'	x
<code>text => text</code>	制作单项 <code>hstore</code>	'a' => 'b'	"a"=>"b"
<code>hstore hstore</code>	连接	'a=>b, c=>d'::hstore 'c=>x, d=>q'::hstore	"a"=>"b", "c"=>"x", "d"=>"q"
<code>hstore ? text</code>	<code>hstore</code> 是否包含键？	'a=>1'::hstore ? 'a'	t
<code>hstore @> hstore</code>	左操作数是否包含右操作数？	'a=>b, b=>1, c=>NULL'::hstore @> 'b=>1'	t
<code>hstore <@</code>	左操作数是否包含于右操作数？	'a=>c'::hstore <@ 'a=>b, b=>1, c=>NULL'	f

hstore			
Note: 不推荐使用=>运算符，并且可能在以后的版本中将其删除。请改用hstore(text, text)函数。			

Table 2. hstore函数

函数	返回类型	描述	示例	结果
hstore(text, text)	hstore	制作单项hstore	hstore('a', 'b')	"a"=>"b"
akeys(hstore)	text[]	以数组形式获取hstore的键	akeys('a=>1,b=>2')	{a,b}
skeys(hstore)	setof text	以集合形式获取hstore的键	skeys('a=>1,b=>2')	a b
avals(hstore)	text[]	以数组形式获取hstore的值	avals('a=>1,b=>2')	{1,2}
svals(hstore)	setof text	以集合形式获取hstore的值	svals('a=>1,b=>2')	1 2
each(hstore)	setof (key text, value text)	以集合形式获取hstore的键值	select * from each('a=>1,b=>2')	key value -----+----- -- a 1 b 2
exist(hstore, text)	boolean	hstore是否包含键?	exist('a=>1', 'a')	t
defined(hstore, text)	boolean	对于键hstore是否包含非空值?	defined('a=>NULL', 'a')	f
delete(hstore, text)	hstore	删除匹配键的所有项	delete('a=>1,b=>2', 'b')	"a"=>"1"

Indexes

hstore对@>和?运算符有索引支持。您可以使用GiST或GIN索引类型。例如：

```
CREATE INDEX hidx ON testhstore USING GIST(h);
CREATE INDEX hidx ON testhstore USING GIN(h);
```

示例

添加键，或使用新键更新现有键：

```
UPDATE tab SET h = h || ('c' => '3');
```

删除键:

```
UPDATE tab SET h = delete(h, 'k1');
```

统计

hstore类型由于其固有的自由度，可能包含许多不同的键。检查有效键是应用程序的任务。下面的示例演示了几种用于检查键和获取统计信息的技术。

简单示例:

```
SELECT * FROM each('aaa=>bq, b=>NULL, ""=>1');
```

使用表:

```
SELECT (each(h)).key, (each(h)).value INTO stat FROM testhstore;
```

在线统计:

```
SELECT key, count(*) FROM
  (SELECT (each(h)).key FROM testhstore) AS stat
  GROUP BY key
  ORDER BY count DESC, key;
  key      | count
-----+-----
line    |   883
query   |   207
pos     |   203
node    |   202
space   |   197
status  |   195
public  |   194
title   |   190
org     |   189
.....
```

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

citext 模块提供了一个不区分大小写的字符串数据类型, citext。本质上, citext在比较值时在内部调用lower 函数。除此之外, 它的操作几乎与text相同

对文本值执行不区分大小写匹配的方法是曾经是在比较值时使用lower 函数, 例如

```
SELECT * FROM tab WHERE lower(col) = LOWER(?);
```

这种方法可行, 但是有缺点:

- 它会使SQL语句变得冗长, 而且总是在列查询值中同时使用lower函数。
- 除非已经使用lower函数创建一个函数索引, 否则该函数不适用于索引。

citext数据类型允许您在SQL查询中不调用lower函数, 并且可以在citext类型的列上创建不区分大小写的主键。类似于ext类型, citext是本地化识别匹配, 这意味着大写和小写字母的匹配依赖于数据库LC_CTYPE设置规则。这和在查询中使用lower函数功能相同, 但它由数据类型透明地完成, 因此您不必在查询中执行任何特殊操作。

安装 citext

在使用citext类型之前, 必须在要使用该类型的每个数据库中注册扩展名:

```
$ psql -d testdb -c "CREATE EXTENSION citext"
```

使用citext类型

以下是一个定义一个citext类型数据列的简单的例子:

```
CREATE TABLE users (
    id bigint PRIMARY KEY,
    nick CITEXT NOT NULL,
```

Greenplum的PL/Perl语言

```

        pass TEXT    NOT NULL
) DISTRIBUTED BY (id);

INSERT INTO users VALUES (1, 'larry', md5(random()::text)
);
INSERT INTO users VALUES (2, 'Tom',      md5(random()::text)
);
INSERT INTO users VALUES (3, 'Damian',  md5(random()::text)
);
INSERT INTO users VALUES (4, 'NEAL',     md5(random()::text)
);
INSERT INTO users VALUES (5, 'Bjørn',   md5(random()::text)
);

SELECT * FROM users WHERE nick = 'Larry';

```

SELECT语句将返回一个元组，尽管nick列被设置为larry并且查询指定为Larry

字符串比较操作

`citext`通过将每个字符串的每个字符转换为小写（和调用`lower`函数相似）然后正常比较结果。如果调用`lower`函数后两个字符串的值一样，则认为两个字符串相等。

为了尽可能地模拟不区分大小写的排序规则，`citext`提供了许多字符串处理运算符以及运算函数的特殊版本。例如正则表达式`~`和`~*`在应用到`citext`时表现出相同的操作：他们都是大小写不敏感的。相同的特性也表现在`!~`和`!~*`上，以及`LIKE`运算符`~~`，`~~*`，`!~~`和`!~~*`。如果您需要大小写敏感，可以将运算符的参数强制转换为`text`。

类似，如果它们的参数为`citext`，以下函数执行不区分大小写匹配：

- `regexp_match()`
- `regexp_matches()`
- `regexp_replace()`
- `regexp_split_to_array()`
- `regexp_split_to_table()`
- `replace()`
- `split_part()`
- `strpos()`

- `translate()`

对于`regexp`函数，如果想要大小写敏感，可以声明“c”标志以强制大小写匹配。如果您想要大小写敏感操作，则在使用其中一个函数之前必须强制转换为`text`类型。

局限性

- `citext`类型的折叠操作依赖于数据库的`LC_CTYPE`设置。因此，在创建数据库时，应确定如何比较值。通过在Unicode标准定义的术语中并不是真正大小写不敏感的。实际上，只要您对您的排序规则感到满意，您应该对`citext`的比较感到满意。但是，如果您的数据库中存储了不同语言的数据，则即使排序规则适用于其他语言，有一种语言的用户也可能会发现其查询结果与预期不符。
- `citext`不如`text`高效，因为运算符函数和使用B树的比较函数必须拷贝一份数据并将它们转换为小写进行比较。但是，它比使用`lower`执行不区分大小写的匹配效率稍高一些。
- 如果你需要数据比较在某些情况中大小写敏感但其他情况下大小写不敏感`citext`不一定是一个最好的选择，建议的做法是使用`text`类型，然后在需要大小写不敏感的情况下手动使用`lower`函数。这种方法适用于较少需要大小写不敏感比较的情况下。如果您需要的大小写不敏感比较的频率明显大于大小写敏感，您可以将数据存储为`citext`类型，当想要进行大小写敏感比较时将数据类型强制转换为`text`类型。如果您希望在两种情况下都能高效的进行，您需要建立两个索引。
- 包含`citext`运算的模式必须在当前的`search_path`中（通常是在`public`）如果不是这样的话，请调用大小写敏感的`text`运算。

Greenplum数据库® 6.0文档

配置参数会影响服务器行为的类别，例如资源消耗，查询调整和身份验证。以下主题描述了Greenplum配置参数类别。

- 参考指南
- SQL Command Reference
 - SQL 2008可选特性兼容性
 - Greenplum环境变量
 - 保留标识符和SQL关键字
- 系统目录参考
- gp_toolkit管理模式
- gpperfmon 数据库
- Greenplum 数据库数据类型
 - 字符集支持
- 服务器配置参数
 - 参数类型和值
 - 设置参数
- 参数类别
- 配置参数

- [连接和验证参数](#)
- [系统资源消耗参数](#)
- [GPORCA参数](#)
- [查询调优参数](#)
- [错误报告和日志参数](#)
- [系统监控参数](#)
- [运行时统计信息收集参数](#)
- [自动统计收集参数](#)
- [客户端连接默认参数](#)
- [锁管理参数](#)
- [资源管理参数 \(资源队列\)](#)
- [资源管理参数 \(资源组\)](#)
- [外部表参数](#)
- [数据库表参数](#)
- [历史版本兼容性参数](#)
- [Greenplum数据库阵列配置参数](#)
- [master和segment的Greenplum镜像参数](#)
- [Greenplum数据库扩展参数](#)

连接和验证参数

这些参数控制客户端如何连接和验证Greenplum数据库。

Connection Parameters

gp_connection_send_timeout	tcp_keepalives_count
gp_vmem_idle_resource_timeout	tcp_keepalives_idle
listen_addresses	tcp_keepalives_interval
max_connections	unix_socket_directory
max_prepared_transactions	unix_socket_group
superuser_reserved_connections	unix_socket_permissions

安全和身份验证参数

authentication_timeout	password_encryption
db_user_namespace	password_hash_algorithm
krb_caseins_users	ssl

[krb_server_keyfile](#)[ssl_ciphers](#)

系统资源消耗参数

这些参数设置Greenplum数据库消耗的系统资源限制。

内存消耗参数

这些参数控制系统内存使用。

[gp_vmem_idle_resource_timeout](#)[gp_resource_group_memory_limit](#) (基于资源组的
资源管理)[gp_vmem_protect_limit](#) (基于资源队列的资源管
理)[gp_vmem_protect_segworker_cache_limit](#)[gp_workfile_limit_files_per_query](#)[gp_workfile_limit_per_query](#)[gp_workfile_limit_per_segment](#)[maintenance_work_mem](#)[max_stack_depth](#)[shared_buffers](#)[temp_buffers](#)

OS资源参数

[max_files_per_process](#)[shared_preload_libraries](#)

基于成本的清理延迟参数

Warning: 不要使用基于成本的清理延迟，因为它在segment实例之间异步运行。
在segment级别调用清理成本限制和延迟，而不考虑整个Greenplum数据库阵列的状态

您可以配置VACUUM和ANALYZE命令的执行成本，以减少I/O对并发数据库活动的影
响。当I/O操作的累计成本达到限制时，执行操作的进程会暂停一段时间，然后重置计
数器并继续执行

[vacuum_cost_delay](#)[vacuum_cost_page_hit](#)[vacuum_cost_limit](#)[vacuum_cost_page_miss](#)[vacuum_cost_page_dirty](#)

事务ID管理参数

[xid_stop_limit](#)

xid_warn_limit

GPORCA参数

这些参数控制Greenplum数据库对GPORCA的使用。有关GPORCA的信息，请参阅[关于GPORCA](#)。

gp_enable_relsizer_collection	optimizer_join_arity_for_associativity_commutativity
optimizer	optimizer_join_order
optimizer_analyze_root_partition	optimizer_join_order_threshold
optimizer_array_expansion_threshold	optimizer_mdcache_size
optimizer_cteInlining_bound	optimizer_metadata_caching
optimizer_control	optimizer_parallel_union
optimizer_enable_associativity	optimizer_print_missing_stats
optimizer_enable_master_only_queries	optimizer_print_optimization_stats
optimizer_force_agg_skew_avoidance	optimizer_sort_factor
optimizer_force_multistage_agg	
optimizer_force_three_stage_scalar_dqa	

查询调优参数

这些参数控制SQL查询处理的各个方面，例如查询运算符和运算符设置以及统计采样。

Postgres查询优化器运算符控制参数

以下参数控制Postgres查询优化器可以使用的计划操作类型。启用或禁用计划操作以强制Postgres优化程序选择其他计划。这对于使用不同计划类型测试和比较查询性能非常有用。

enable_bitmapscan	gp_enable_agg_distinct_pruning
enable_groupagg	gp_enable_direct_dispatch
enable_hashagg	gp_enable_fast_sri
enable_hashjoin	gp_enable_groupext_distinct_gather
enable_indexscan	gp_enable_groupext_distinct_pruning
enable_mergejoin	gp_enable_multiphase_agg
enable_nestloop	gp_enable_predicate_propagation
enable_seqscan	gp_enable_preunique

enable_sort	gp_enable_relsize_collection
enable_tidscan	gp_enable_sort_distinct
gp_enable_adaptive_nestloop	gp_enable_sort_limit
gp_enable_agg_distinct	

Postgres查询优化器成本计算参数

Warning: 请勿调整这些查询成本计算参数。它们经过调整以反映Greenplum数据库硬件配置和典型工作负载。所有这些参数都是相关的。更改一个而不更改其他一个可能对性能产生负面影响。

cpu_index_tuple_cost	gp_motion_cost_per_row
cpu_operator_cost	gp_segments_for_planner
cpu_tuple_cost	random_page_cost
cursor_tuple_fraction	seq_page_cost
effective_cache_size	

数据库统计采样参数

这些参数调整ANALYZE操作采样的数据量。调整这些参数会影响系统范围内的统计信息收集。您可以使用ALTER TABLE SET STATISTICS子句在特定表和列上配置统计信息收集。

default_statistics_target

排序运算符配置参数

gp_enable_sort_distinct
gp_enable_sort_limit

聚合运算符配置参数

gp_enable_agg_distinct	gp_enable_groupext_distinct_gather
gp_enable_agg_distinct_pruning	gp_enable_groupext_distinct_pruning
gp_enable_multiphase_agg	gp_workfile_compression
gp_enable_preunique	

连接运算符配置参数

join_collapse_limit	gp_statistics_use_fkeys
gp_adjust_selectivity_for_outerjoins	gp_workfile_compression
gp_hashjoin_tuples_per_bucket	

其他Postgres查询优化器配置参数

fromCollapseLimit
gp_enable_predicate_propagation
gp_max_plan_size
gp_statistics_pullup_from_child_partition

查询计划执行

控制查询计划的执行。

gp_max_slices

错误报告和日志参数

这些配置参数控制Greenplum数据库日志记录。

日志轮换

log_rotation_age	log_truncate_on_rotation
log_rotation_size	

何时记录

client_min_messages	log_min_error_statement
gp_interconnect_debug_retry_interval	log_min_messages
log_error_verbosity	optimizer_minidump
log_min_duration_statement	

记录什么

debug.pretty_print	log_executor_stats
debug.print_parse	log_hostname
debug.print_plan	gp_log_interconnect
debug.print_prelim_plan	log_parser_stats
debug.print_rewritten	log_planner_stats
debug.print_slice_table	log_statement
log_autostats	log_statement_stats
log_connections	log_timezone
log_disconnections	gp_debug_linger
log_dispatch_stats	gp_log_format
log_duration	gp_reraise_signal

系统监控参数

这些配置参数控制Greenplum数据库数据收集和与数据库监视相关的通知。

Greenplum性能数据库

以下参数配置填充gpperfmon数据库的数据收集代理。

gp_enable_gpperfmon	gpperfmon_log_alert_level
gp_gpperfmon_send_interval	gpperfmon_port

查询指标收集参数

这些参数启用和配置查询指标收集。启用后，Greenplum数据库会在查询执行期间将指标保存到共享内存。这些指标由Pivotal Greenplum Command Center使用，该系统包含在Pivotal的商业版Greenplum数据库中。

gp_enable_query_metrics	gp_instrument_shmem_size
---	--

运行时统计信息收集参数

这些参数控制服务器统计信息收集功能。启用统计信息收集后，可以使用`pg_stat`系列目录视图访问统计信息数据。

--	--

[stats_queue_level](#)[update_process_title](#)

自动统计收集参数

启用自动统计信息收集时，您可以在与某个阈值的行数被更新 (on_change) 或新生成没有统计信息的表时，在与 INSERT, UPDATE, DELETE, COPY 或 CREATE TABLE ... AS SELECT 语句相同的事务中自动运行 ANALYZE。要启用此功能，请在 Greenplum 数据库 master 的 postgresql.conf 文件中设置以下服务器配置参数，然后重新启动 Greenplum 数据库：

[gp_autostats_mode](#)[gp_autostats_mode_in_functions](#)[gp_autostats_on_change_threshold](#)[log_autostats](#)

Warning: 根据数据库操作的特定性质，自动统计信息收集可能会对性能产生负面影响。仔细评估 on_no_stats 的默认设置是否适合您的系统。

客户端连接默认参数

这些配置参数设置用于客户端连接的默认值。

语句行为参数

[check_function_bodies](#)[default_transaction_read_only](#)[default_tablespace](#)[search_path](#)[default_transaction_deferrable](#)[statement_timeout](#)[default_transaction_isolation](#)[vacuum_freeze_min_age](#)

区域设置和格式化参数

[client_encoding](#)[lc_messages](#)[DateStyle](#)[lc_monetary](#)[extra_float_digits](#)[lc_numeric](#)[IntervalStyle](#)[lc_time](#)[lc_collate](#)[TimeZone](#)[lc_ctype](#)

其他客户端默认参数

<code>dynamic_library_path</code>	<code>local_preload_libraries</code>
<code>explain_pretty_print</code>	

锁管理参数

这些配置参数设置锁和死锁的限制。

<code>deadlock_timeout</code>	<code>lock_timeout</code>
<code>gp_enable_global_deadlock_detector</code>	<code>max_locks_per_transaction</code>
<code>gp_global_deadlock_detector_period</code>	

资源管理参数 (资源队列)

以下配置参数配置Greenplum数据库资源管理功能 (资源队列) , 查询优先级, 内存利用率和并发控制。

<code>gp_resqueue_memory_policy</code>	<code>max_resource_portals_per_transaction</code>
<code>gp_resqueue_priority</code>	<code>max_statement_mem</code>
<code>gp_resqueue_priority_cpucores_per_segment</code>	<code>resource_cleanup_gangs_on_wait</code>
<code>gp_resqueue_priority_sweeper_interval</code>	<code>resource_select_only</code>
<code>gp_vmem_idle_resource_timeout</code>	<code>runaway_detector_activation_percent</code>
<code>gp_vmem_protect_limit</code>	<code>statement_mem</code>
<code>gp_vmem_protect_segworker_cache_limit</code>	<code>stats_queue_level</code>
<code>max_resource_queues</code>	<code>vmem_process_interrupt</code>

资源管理参数 (资源组)

以下参数配置Greenplum数据库资源组工作负载管理功能。

<code>gp_resgroup_memory_policy</code>	<code>gp_vmem_idle_resource_timeout</code>
<code>gp_resource_group_bypass</code>	<code>gp_vmem_protect_segworker_cache_limit</code>
<code>gp_resource_group_cpu_limit</code>	<code>memory_spill_ratio</code>
<code>gp_resource_group_memory_limit</code>	<code>vmem_process_interrupt</code>
<code>gp_resource_manager</code>	

外部表参数

以下参数配置Greenplum数据库的外部表功能。

<code>gp_external_enable_exec</code>	<code>readable_external_table_timeout</code>
<code>gp_external_enable_filter_pushdown</code>	<code>writable_external_table_bufsize</code>
<code>gp_external_max_segs</code>	<code>verify_gpfists_cert</code>
<code>gp_initial_bad_row_limit</code>	
<code>gp_reject_percent_threshold</code>	

数据库表参数

以下参数配置Greenplum数据库表的默认选项设置。

<code>gp_create_table_random_default_distribution</code>
<code>gp_default_storage_options</code>
<code>gp_enable_exchange_default_partition</code>
<code>gp_enable_segment_copy_checking</code>
<code>gp_use_legacy_hashops</code>

追加优化表参数

以下参数配置Greenplum数据库的追加优化表功能。

<code>max_appendonly_tables</code>
<code>gp_appendonly_compaction</code>
<code>gp_appendonly_compaction_threshold</code>
<code>validate_previous_free_tid</code>

历史版本兼容性参数

以下参数提供与较旧的PostgreSQL和Greenplum数据库版本的兼容性。您无需在Greenplum数据库中更改这些参数。

PostgreSQL

<code>array_nulls</code>	<code>regex_flavor</code>
--------------------------	---------------------------

backslash_quote	standard_conforming_strings
escape_string_warning	transform_null_equals

Greenplum数据库

gp_ignore_error_table

Greenplum数据库阵列配置参数

本主题中的参数控制Greenplum数据库阵列及其组件的配置：segment，master，分布式事务管理器，主镜像和互连。

互连配置参数

gp_interconnect_fc_method	gp_interconnect_setup_timeout
gp_interconnect_hash_multiplier	gp_interconnect_snd_queue_depth
gp_interconnect_queue_depth	gp_interconnect_type
	gp_max_packet_size

Note: Greenplum数据库仅支持UDPIFC（默认）和TCP互连类型。

调度配置参数

gp_cached_segworkers_threshold	gp_segment_connect_timeout
gp_enable_direct_dispatch	gp_set_proc_affinity

故障操作参数

gp_set_read_only	gp_fts_probe_timeout
gp_fts_probe_interval	gp_fts_probe_threadcount
gp_fts_probe_retries	gp_log_fts

分布式事务管理参数

gp_max_local_distributed_cache
--

只读参数

gp_command_count	gp_role
gp_content	gp_session_id
gp_dbid	gp_server_version
	gp_server_version_num

master和segment的Greenplum镜像参数

这些参数控制Greenplum数据库master和standby之间的复制配置。

keep_wal_segments
repl_catchup_within_range
replication_timeout
wal_receiver_status_interval

Greenplum数据库扩展参数

本主题中的参数控制Greenplum数据库扩展的配置。

pljava_classpath
pljava_classpath_insecure
pljava_statement_cache_size
pljava_release_lingering_savepoints
pljava_vmoptions

XML数据参数

xmlbinary
xmloption

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

日期/时间类型

伪类型

全文搜索类型

范围类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

范围类型是一种数据类型，它代表一种元素类型(即范围类型的子类型)的一个范围。举例说, `timestamp` 的范围类型可以用来表示一个会议室预定的时间范围。在这个例子中，数据类型是 `tsrange` (“`timestamp range`”的缩写), 这里 `timestamp` 是子数据类型。子数据类型必须有一个整体顺序，因此很好确定一个元素值是否在范围值域内，范围值域前，还是范围值域后面。

范围类型表示单个范围内的许多元素，所以非常有用，因为一些概念，像重叠范围，能够用范围类型清晰表达。为调度目的使用时间和日期是一个更清楚的例子；对于价格范围，仪器的测量范围，等等，也很有用。

父主题：[Greenplum 数据库数据类型](#)

内置范围类型

Greenplum 数据库带有以下内置范围类型：

- `int4range` -- `integer` 范围
- `int8range` -- `bigint` 范围
- `numrange` -- `numeric` 范围
- `tsrange` -- `timestamp without time zone` 范围
- `tstzrange` -- `timestamp with time zone` 范围
- `daterange` -- `date` 范围

除此之外，你也可以定义你自己的范围类型；参见 [CREATE TYPE](#) 了解更多信息。

举例

```
CREATE TABLE reservation (room int, during tsrange);
INSERT INTO reservation VALUES
  (1108, '[2010-01-01 14:30, 2010-01-01 15:30]');
```

-- 包含
`SELECT int4range(10, 20) @> 3;`

-- 重叠
`SELECT numrange(11.1, 22.2) && numrange(20.0, 30.0);`

-- 提取上边界



Greenplum PL/Python语言

```
SELECT upper(int8range(15, 25));  
  
-- 计算交集  
SELECT int4range(10, 20) * int4range(15, 25);  
  
-- 是否为空?  
SELECT isempty(numrange(1, 5));
```

参见 [范围函数和运算符](#) 查看范围类型的操作符和函数完整列表.

包含边界与非包含边界

每一个非空范围有两个边界, 下边界和上边界. 所有在这两个边界之间的点都包含在范围内. 包含边界意味着边界点本身也包括在范围内, 而非包含边界意思是边界点不包括在范围内.

对于一个范围的文本格式, 包含下边界用 [表示, 而非包含下边界用 (表示. 同样, 包含上边界用] 表示, 而非包含上边界用) 表示. (参见 [范围函数和运算符](#) 了解更多信息.)

函数 `upper_inc` 和 `lower_inc` 分别用来测试一个范围值的上下边界包含关系.

无限 (无边界) 范围

范围的下边界可以忽略, 意味着所有小于上边界的点都包含在范围内. 同样地, 如果范围的上边界被忽略, 那么所有大于下边界的点都包含在范围内. 如果上下边界都忽略了, 那么这个元素类型的所有值都被认为包含在范围内.

可以等价地分别认为, 这个时候下边界是“负无穷”, 或者上边界是“正无穷”. 但是请注意, 这些无穷值不是一个范围的元素类型的值, 也不是范围的一部分. (所有无穷边界不存在包含问题 -- 如果你这样写的话, 会自动转换为非包含边界.)

有些元素类型也有“infinity”的表示, 但那仅仅是一个值, 与范围边界无关. 举例说, 在时间戳范围内, `[today,]` 与 `[today,)` 含义相同. 但是 `[today, infinity]` 与 `[today, infinity)` 含义就不同了 -- 后者排除了特殊的时间戳值 `infinity`.

函数 `lower_inf` 和 `upper_inf` 可以分别检测一个范围的无限上边界和下边界.

范围类型输入/输出

范围类型的输入必须遵循下面的模式:

```
(lower-bound, upper-bound)
(lower-bound, upper-bound]
[lower-bound, upper-bound)
[lower-bound, upper-bound]
empty
```

和前面说的一样, 方括号或圆括号表示是否上下边界是包含, 非包含关系. 注意最后一种模式是 `empty`, 表示一个空范围 (一个不包含任何点的范围).

lower-bound 可以是一个子类型的有效字符串, 或者用 `empty` 表示没有下边界. 同样地, *upper-bound* 可以是一个子类型的有效字符串, 或者用 `empty` 表示没有上边界.

每一个边界值都可以是用 " (双引号) 引起来的字符串. 如果边界值包含圆括号, 方括号, 逗号, 双引号, 反斜杠的时候, 则必须引起, 因为这些字符可能是范围语法的一部分. 如果引起的边界值有双引号或者反斜杠, 需要在前面插入额外的反斜杠进行转义. (在双引号引起的边界值中两个双引号也用来表示一个双引号字符, 类似于 SQL 字面值中的单引号规则.) 当然, 你也可以避免引号, 只用反斜杠把所有会被认为是范围语法一部分的字符进行转义. 另外, 边界值也可以写成空字符串 "", 因为啥也不写意味着无限边界.

在范围值的前后, 允许出现空格, 但是任何在圆括号或方括号之间的空格会被认为是上边界或下边界的一部分. (与元素类型有关, 或许有或许没意义.)

例如:

```
-- 包含 3, 不包含 7, 并且包含期间的所有点
SELECT '[3,7)::int4range;

-- 不包含 3 和 7, 单包含之间的所有点
SELECT '(3,7)::int4range;

-- 只包含单个点 4
SELECT '[4,4)::int4range;

-- 不包含任何点 (会被规范化为 'empty')
SELECT '[4,4)::int4range;
```

构造范围

每个范围类型有一个与范围类型同名的构造函数. 由于避免了边界值的额外引号(和转义), 通常使用构造函数比写一个范围字面常量更方便. 构造函数接受二或三个参数. 两个参数形式构造一个标准格式的范围(包含下边界, 不包含上边界), 而三个参数格式构造一个通过第三个参数指定的边界(包含关系). 第三个参数必须是下面的字符串之一: (), (), [], 或者 []. 举例如下:

```
-- 全格式是: 下边界, 上边界, 以及文本参数指示
-- 包含/非包含边界.
SELECT numrange(1.0, 14.0, '[]');

-- 如果第三个参数忽略, 默认是 'D'.
SELECT numrange(1.0, 14.0);

-- 虽然这里指定 '()'，显示值会被转换成经典格式
-- 因为 int8range 是一个离散范围类型 (见下).
SELECT int8range(1, 14, '()'');

-- 对某一边界使用 NULL 导致这一边无边界.
SELECT numrange(NULL, 2.2);
```

离散范围类型

离散范围类型的元素类型有一个明确定义的“步长”, 例如 `integer` 或 `date`. 在这些类型中, 当在两个元素间没有合法的值时, 我们就可以说这两个元素是相邻的. 这个与连续范围相对, 在连续范围内, 两个元素之间总是可以找到其它元素值. 例如, `numeric` 类型的范围就是连续的, `timestamp` 类型的范围也是连续的. (即使 `timestamp` 有有限的精度, 理论上可以认为它是离散的, 但由于步长值通常不重要, 所以最好认为它是连续的.)

另一个理解离散范围类型的方法是, 对于每一个元素值, 它有清晰的“后一个”和“前一个”值. 了解了这些, 就可以通过选择后一个或前一个元素值, 取代原始元素值, 来对范围边界的包含和非包含关系进行转换. 举例说, 一个整数范围类型 [4, 8] 和 (3, 9) 表示的是同样值的集合; 但对于一个 `numeric` 类型的范围就没有这样的关系.

一个离散范围类型应该有一个 `canonicalization` 函数来感知元素类型的步长. `canonicalization` 函数用于把等价范围类型值转换为同一表示, 特别是一致的包含或非包含边界. 如果没有指定 `canonicalization` 函数, 那么不同格式总是当成不相等对待, 即使他们实际上表示同样的值域.

内置范围类型 `int4range`, `int8range`, 和 `daterange` 全部使用一

个包含下边界, 不包含上边界的一个经典格式; 即 `[)`. 然而, 用户定义的范围类型可以使用其他转换形式.

定义新的范围类型

用户可以定义他们自己的范围类型. 这样做的通常原因是使用的子类型的范围没有内置提供. 例如, 定义子类型为 `float8` 的新的范围类型:

```
CREATE TYPE floatrange AS RANGE (
    subtype = float8,
    subtype_diff = float8mi
);

SELECT '[1.234, 5.678]':>floatrange;
```

因为 `float8` 没有“步长”, 这个例子中我们不能定义一个 canonicalization 函数给它.

定义你自己的范围类型也允许你指定一个不同子类型 B-树 操作类或校对方式(collation)来使用, 可以改变排序规则, 以决定哪些值会落在给定范围内.

如果子类型被认为是离散值而不是连续值, `CREATE TYPE` 命令应该指定一个 canonicalization 函数. canonicalization 函数接受一个范围值, 返回一个具有不同边界和格式的等价范围值. 对于表示同一值集的两个范围, canonicalization 函数输出一致., 例如整数范围 `[1, 7]` 和 `[1, 8)`, 一定是相同的. 不论你选择哪一个作为标准输出, 只要不同格式的等价值总是映射到同样格式的相同值(就可以). 除了调整包含/非包含边界格式, canonicalization 函数还可以圆整(四舍五入)边界值, 例如设定的步长比子类型存储分辨能力宽(就需要这个功能). 例如, 一个 `timestamp` 范围类型, 可以定义为步长为 1 小时, 这种情况下, canonicalization 函数需要对不是整小时的边界进行圆整, 当然也可以抛出一个异常.

另外, 与 GiST 或 SP-GiST 索引一起使用的范围类型应该定义一个子类型差异函数, 即 `subtype_diff`, 函数. (没有 `subtype_diff` 函数索引也能工作, 但可能被认为没有提供这个函数时高效.) 子类型差异函数接受两个子类型输入值, 同时返回他们之间的差(即, X 减去 Y)用 `float8` 值表示. 在我们上面的例子中, 函数 `float8mi` 隐含有常规 `float8` 相减操作来用; 但对于任何其他子类型, 某种类型的转换是必要的. 关于如何表示数值之间差异的一些创造性想法也很需要. 为了最大限度扩展这种可能, `subtype_diff` 函数应该与选定操作类和校对函数一致; 也就是说, 根据排序顺序, 当第一个参数比第二个大时, 这个函数应该返回正数.

一个关于 `subtype_diff` 函数不那么简单的例子是:

```
CREATE FUNCTION time_subtype_diff(x time, y time) RETURNS float8 AS
'SELECT EXTRACT(EPOCH FROM (x - y))' LANGUAGE sql STRICT
IMMUTABLE;

CREATE TYPE timerange AS RANGE (
    subtype = time,
    subtype_diff = time_subtype_diff
);

SELECT '[11:10, 23:00]':>timerange;
```

参见 [CREATE TYPE](#) 了解更多关于创建范围类型的详细信息.

索引

GiST 和 SP-GiST 索引可以在范围类型列上创建. 例如, 创建一个 GiST 索引:

```
CREATE INDEX reservation_idx ON reservation USING GIST (during);
```

一个 GiST 或者 SP-GiST 索引能够加速包含以下操作的范围运算: `=`, `&&`, `<@, @>`, `<<, >>`, `- | -`, `&<`, 以及 `&>` (参见 [范围函数和运算符](#) 了解更多信息).

另外, B-树 和 hash 索引也可以创建在范围类型的表列上. 对于这些索引类型, 基本上只对相等范围操作有用. 也有一个范围值的 B-树 排序定义, 具有相应的 `<` 和 `>` 操作, 但是排序方法相当简单粗暴, 现实生活中并不常用. 范围类型的 B-树 和 hash 支持主要允许排序和查询内部建立 hash, 而不是为了创建实际的索引.

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

dblink模块支持从数据库会话中连接到其他Greenplum数据库数据库。这些数据库可以驻留在相同的Greenplum数据库系统中，也可以驻留在远程系统中。

Greenplum数据库支持Greenplum数据库安装中具有相同主要版本号的数据库之间的dblink连接。您还可以使用dblink连接到使用兼容libpq库的其他Greenplum数据库安装。

Note: dblink适用于数据库用户在其他数据库中执行简短的临时查询。dblink不能替代外部表或gpcopy之类的管理工具。

Greenplum数据库dblink模块是PostgreSQL dblink模块的修改版本。在Greenplum数据库中使用该模块时，存在一些限制。

安装和注册模块

当您安装Greenplum数据库时，将安装dblink模块。在使用模块中定义的任何函数之前，必须在要使用这些函数的每个数据库中注册dblink扩展。

Greenplum数据库注意事项

在此版本的Greenplum数据库中，修改表数据的语句不能使用命名或隐式dblink连接。而是必须直接在dblink()函数中提供连接字符串。例如：

```
gpadmin=# CREATE TABLE testdblllocal (a int, b text)
DISTRIBUTED BY (a);
CREATE TABLE
gpadmin=# INSERT INTO testdblllocal select * FROM
dblink('dbname=postgres', 'SELECT * FROM testdblink') AS
dbltab(id int, product text);
INSERT 0 2
```

dblink的Greenplum数据库版本禁用以下异步函数：

- `dblink_send_query()`
- `dblink_is_busy()`
- `dblink_get_result()`



使用dblink

以下过程确定了在Greenplum数据库中配置和使用dblink的基本步骤。这些示例使用dblink_connect()创建与数据库的连接，并使用dblink()执行SQL查询。

- 首先创建一个示例表以使用dblink函数进行查询。这些命令在postgres数据库中创建一个小表，稍后您将使用dblink从testdb数据库中查询该表：

```
$ psql -d postgres
psql (9.4.20)
Type "help" for help.

postgres=# CREATE TABLE testdblink (a int, b text)
DISTRIBUTED BY (a);
CREATE TABLE
postgres=# INSERT INTO testdblink VALUES (1, 'Cheese'),
(2, 'Fish');
INSERT 0 2
postgres=# \q
$
```

- 以超级用户身份登录到另一个数据库。在此示例中，超级用户gpadmin登录到数据库testdb。如果dblink函数尚不可用，请在数据库中注册dblink扩展：

```
$ psql -d testdb
psql (9.4beta1)
Type "help" for help.

testdb=# CREATE EXTENSION dblink;
CREATE EXTENSION
```

- 使用dblink_connect()函数创建与另一个数据库的隐式或命名连接。您提供的连接字符串应该是libpq样式的关键字/值字符串。本示例创建一个名为mylocalconn的连接到本地Greenplum数据库系统上的postgres数据库：

```
testdb=# SELECT dblink_connect('mylocalconn',
'dbname=postgres user=gpadmin');
dblink_connect
-----
OK
(1 row)
```

Note: 如果未指定user，则在启动Greenplum数据库时，`dblink_connect()`将使用PGUSER环境变量的值。如果未设置PGUSER，则默认值为启动Greenplum数据库的系统用户。

4. 使用`dblink()`函数可使用已配置的连接查询数据库。请记住，此函数返回记录类型，因此您必须分配`dblink()`查询中返回的列。例如，以下命令使用命名连接来查询您先前创建的表：

```
testdb=# SELECT * FROM dblink('mylocalconn', 'SELECT *  
FROM testdblink') AS dbltab(id int, product text);  
id | product  
---+-----  
1  | Cheese  
2  | Fish  
(2 rows)
```

要以另一个用户身份连接到本地数据库，请在连接字符串中指定该用户。本示例以用户`test_user`的身份连接到数据库。使用`dblink_connect()`，超级用户无需指定密码即可创建与另一个本地数据库的连接。

```
testdb=# SELECT dblink_connect('localconn2',  
'dbname=postgres user=test_user');
```

要建立与远程数据库系统的连接，请在连接字符串中包含主机和密码信息。例如，要创建到远程系统的隐式`dblink`连接：

```
testdb=# SELECT dblink_connect('host=remotehost port=5432  
dbname=postgres user=gpadmin password=secret');
```

作为非超级用户使用`dblink`

要使用`dblink_connect()`与数据库建立连接，非超级用户必须在连接字符串中包含主机，用户和密码信息。即使连接到本地数据库，也必须包括主机，用户和密码信息。例如，用户`test_user`可以使用以下命令创建到本地系统`mdw`的`dblink`连接：

```
testdb=> SELECT dblink_connect('host=mdw port=5432  
dbname=postgres user=test_user password=secret');
```

如果非超级用户需要创建不需要密码的`dblink`连接，则可以使用`dblink_connect_u()`函数。`dblink_connect_u()`函数与`dblink_connect()`相同，区别在于它允许非超级用户创建不需

要密码的连接。

最初安装dblink_connect_u()时，它具有从PUBLIC撤消的所有特权，因此除超级用户外，它无法调用。在某些情况下，将dblink_connect_u()的EXECUTE权限授予被认为可信任的特定用户可能是适当的，但是应格外小心。

Warning: 如果Greenplum数据库系统为用户配置了不涉及密码的身份验证方法，那么当非超级用户执行dblink_connect_u()时，可能会冒充他人，并随后升级特权。dblink连接似乎源自该函数指定的用户。例如，非超级用户可以执行dblink_connect_u()并指定使用trust认证配置的用户。

同样，即使dblink连接需要密码，也可以从服务器环境中提供该密码，例如属于服务器用户的~/.pgpass文件。建议所有属于服务器用户的~/.pgpass文件都不要包含任何指定通配符主机名的记录。

1. 作为超级用户，对用户数据库中的dblink_connect_u()函数授予EXECUTE特权。本示例向具有创建隐式或命名dblink连接的签名的函数的非超级用户test_user授予特权。

```
testdb=# GRANT EXECUTE ON FUNCTION
dblink_connect_u(text) TO test_user;
testdb=# GRANT EXECUTE ON FUNCTION
dblink_connect_u(text, text) TO test_user;
```

2. 现在，test_user无需密码即可创建到另一个本地数据库的连接。例如，test_user可以登录到testdb数据库并执行此命令以创建一个名为testconn的连接到本地postgres数据库。

```
testdb=> SELECT dblink_connect_u('testconn',
'dbname=postgres user=test_user');
```

Note: 如果未指定user，则在启动Greenplum数据库时，dblink_connect_u()将使用PGUSER环境变量的值。如果未设置PGUSER，则默认值为启动Greenplum数据库的系统用户。

3. test_user可以使用dblink()函数通过dblink连接执行查询。例如，此命令使用在上一步中创建的名为testconn的dblink连接。test_user必须具有对该表的适当访问权限。

```
testdb=> SELECT * FROM dblink('testconn', 'SELECT * FROM
testdblink') AS dbltab(id int, product text);
```

附加模块文档

有关此模块中各个函数的详细信息，请参考PostgreSQL文档中的[dblink](#)。

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

fuzzystrmatch模块提供了根据各种算法确定字符串之间的相似性和距离的函数。

Greenplum数据库fuzzystrmatch模块等同于PostgreSQL fuzzystrmatch模块。该模块没有Greenplum数据库或MPP特定的注意事项。

安装和注册模块

当您安装Greenplum数据库时，将安装fuzzystrmatch模块。在使用模块中定义的任何函数之前，必须在要使用这些函数的每个数据库中注册fuzzystrmatch扩展。

模块文档

有关此模块中各个函数的详细信息，请参见PostgreSQL文档中的[fuzzystrmatch](#)。



Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

描述Greenplum数据库中的Oracle兼容SQL函数。这些函数针对PostgreSQL，但也可以在Greenplum数据库中使用。

关于Greenplum数据库的Oracle兼容性函数

用于Greenplum数据库的Oracle兼容性函数基

于<https://github.com/orafce/orafce> 上的开源Orafce项目。

可以在[Greenplum数据库开源项目](#) 的gpcontrib/orafce目录中找到经过修改的Greenplum数据库的Orafce源文件。源文件包括Orafce 3.6.1版本和对[3af70a28f6](#) 的其他提交。

以下函数默认情况下在Greenplum数据库中可用，并且不需要安装Oracle兼容性函数：

- sinh
- tanh
- cosh
- decode (更多信息请参考[Oracle和Greenplum的实现差异](#)。)

有关使用Oracle兼容性函数的信息，请参阅Orafce项目主页上的文档，网址为<https://github.com/orafce/orafce>。Orafce项目中文档的源文件也包含在Greenplum数据库开源项目中。

安装Oracle兼容性函数

Note: 始终使用Greenplum数据库版本随附的Oracle兼容性函数模块。在升级到新的Greenplum数据库版本之前，请从每个数据库中卸载兼容性函数，然后，在升级完成后，从新的Greenplum数据库版本中重新安装兼容性函数。有关升级先决条件和步骤，请参阅Greenplum数据库发行说明。

使用CREATE EXTENSION SQL命令在每个数据库中安装 Oracle兼容性函数。

Greenplum的PL/Perl语言

```
$ psql -d db_name -c 'CREATE EXTENSION orafce;'
```

Note: 某些Oracle兼容性函数位于oracle模式中。要访问它们，请为数据库设置搜索路径以包括oracle模式名称。例如，此命令设置数据库的默认搜索路径以包括oracle模式：

```
ALTER DATABASE db_name SET search_path = "$user", public, oracle;
```

要卸载Oracle兼容性函数，请使用DROP EXTENSION SQL命令删除orafce扩展。

```
$ psql -d db_name -c 'DROP EXTENSION orafce;'
```

如果要从Greenplum数据库5.x发行版中卸载兼容性函数，请改用以下命令：

```
$ psql -d db_name -f $GPHOME/share/postgresql/contrib/uninstall_orafunc.sql
```

Oracle和Greenplum的实现差异

Greenplum数据库中兼容性函数的实现与Oracle实现之间存在一些差异。如果使用验证脚本，则输出可能与Oracle中的输出不完全相同。一些区别如下：

- Oracle执行十进制舍入，Greenplum数据库则不：
 - 2.00在Oracle中变成2
 - Greenplum数据库中的2.00仍为2.00
- 提供的Oracle兼容性函数以不同的方式处理隐式类型转换。例如，使用decode函数：

```
decode(expression, value, return [,value, return]...
      [, default])
```

Oracle在比较之前自动将表达式和每个值转换为第一个值的数据类型。Oracle自动将return转换为与第一个结果相同的数据类型。Greenplum数据库实现将return和default限制为相同的数据类型。如果可以将值的数据类型转换为表达式的数据类型，则表达式和值可以是不同的类型。这是隐式完成的。否则，decode将失败，并出现invalid input syntax错误。例如：

```

SELECT decode('a','M',true,false);
CASE
-----
f
(1 row)
SELECT decode(1,'M',true,false);
ERROR: Invalid input syntax for integer: "M"
LINE 1: SELECT decode(1,'M',true,false);

```

- bigint格式的数字在Oracle中以科学计数法显示，但在Greenplum数据库中不是这样：
 - 9223372036854775在Oracle中显示为9.2234E + 15
 - 9223372036854775在Greenplum数据库中仍为9223372036854775
- Oracle中的默认日期和时间戳格式与Greenplum数据库中的默认格式不同。如果执行以下代码：

```

CREATE TABLE TEST(date1 date, time1 timestamp, time2
                  timestamp with time zone);
INSERT INTO TEST VALUES ('2001-11-11','2001-12-13
                           01:51:15','2001-12-13 01:51:15
                           -08:00');
SELECT DECODE(date1, '2001-11-11', '2001-01-01') FROM
TEST;

```

Greenplum数据库返回该行，但是Oracle不返回任何行。
Note: Oracle中返回该行的正确语法是：

```

SELECT DECODE(to_char(date1, 'YYYY-MM-DD'), '2001-11-
11',
              '2001-01-01') FROM TEST

```

以下是将Oracle兼容函数与PostgreSQL和Greenplum数据库结合使用的区别。

- 从Greenplum数据库Oracle兼容性函数中删除了decode()函数。Greenplum数据库解析器将decode() - 函数调用转换为case语句。
- Oracle兼容性函数dbms_pipe软件包中的函数仅在Greenplum数据库master主机上执行。
- 没有为Greenplum数据库实现Oracle兼容性函数dbms_alert软件包中的函数。
- Orafce项目中的升级脚本不适用于Greenplum数据库。

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum PL/Python语言扩展

Greenplum PL/Container语言扩展

Greenplum的PL/Java语言扩展

Greenplum数据库安装有一个可选的加密/解密函数模块pgcrypto。pgcrypto函数允许数据库管理员以加密形式存储数据的某些列。这为敏感数据增加了一层额外的保护，因为没有加密密钥的任何人都无法读取以加密形式存储在Greenplum数据库中的数据，也不能直接从磁盘读取该数据。

Note: pgcrypto函数在数据库服务器内部运行，这意味着所有数据和密码都以明文形式在pgcrypto和客户端应用程序之间传输。为了获得最佳安全性，请考虑在客户端和Greenplum主服务器之间使用SSL连接。

有关此模块中各个函数的更多信息，请参见PostgreSQL文档中的[pgcrypto](#) □。



Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ PXF介绍

PXF谓词下推

PXF列投影

□ PXF管理手册

□ 使用PXF访问hadoop

□ 使用PXF访问Azure, Google云端存储, Minio和S3对象存储

使用PXF访问SQL数据库
(JDBC)

PXF故障排除

□ PXF实用程序手册

Back to the Administrator Guide

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

PXF支持谓词下推。启用谓词下推时,可以提取SELECT查询中WHERE子句的约束并将其传递到外部数据源进行过滤。此过程可以提高查询性能,还可以减少传输到Greenplum数据库的数据量。

通过设置 `gp_external_enable_filter_pushdown` 服务器配置参数, 可以为所有外部表协议(包括pxf)启用或禁用谓词下推。此配置参数的默认值为on;将其设置为关闭以禁用谓词下推。例如:

```
SHOW gp_external_enable_filter_pushdown;
SET gp_external_enable_filter_pushdown TO 'on';
```

Note: 某些外部数据源不支持谓词下推。此外,某些数据类型或运算符可能不支持谓词下推。如果查询访问不支持谓词下推的数据源,则执行查询时不进行谓词下推(数据传输到Greenplum数据库后进行过滤)

PXF使用不同的连接器访问数据源, 谓词下推支持由特定的连接器实现方式决定。以下PXF连接器支持谓词下推:

- Hive Connector, all profiles
- HBase Connector
- JDBC Connector
- S3 Connector using the Amazon S3 Select service to access CSV and Parquet data

PXF谓词下推可与这些数据类型一起使用(特定连接器):

- `INT2` , `INT4` , `INT8`
- `CHAR` , `TEXT`
- `FLOAT`
- `NUMERIC` (使用S3 Select时, 不适用于S3连接器)
- `BOOL`
- `DATE` , `TIMESTAMP` (使用S3 Select时仅适用于JDBC连接器和S3连接器)

您可以通过以下运算符使用PXF过滤器下推功能:

- `<` , `<=` , `>=` , `>`
- `<>` , `=`

AND , OR , NOT

- 数组 `INT` 和 `TEXT` 的运算符 `IN` (仅JDBC连接器)
- `LIKE` (`TEXT` 字段, 仅JDBC连接器)

总而言之, 必须满足以下所有条件才能进行谓词下推:

- 将 `gp_external_enable_filter_pushdown` 服务配置参数设置为 '`on`' 来启用外部表谓词下推
- 访问外部数据源的Greenplum数据库协议必须支持谓词下推.PXF外部表协议支持下推
- 访问的外部数据源必须支持下推。例如, HBase和Hive支持下推
- 使用pxf协议创建的外部表查询, 基础PXF连接器还必须支持谓词下推。例如, PXF Hive, HBase和JDBC连接器支持下推。
 - 有关Hive支持此功能的更多信息, 请参考Hive[分区过滤器下推](#)。

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ [PXF介绍](#)

[PXF谓词下推](#)

PXF列投影

□ [PXF管理手册](#)

□ [使用PXF访问hadoop](#)

□ [使用PXF访问Azure, Google云端存储, Minio和S3对象存储](#)

[使用PXF访问SQL数据库\(JDBC\)](#)

[PXF故障排除](#)

□ [PXF实用程序手册](#)

[Back to the Administrator Guide](#)

[Greenplum database 5管理员指南](#)

[Greenplum database 4管理员指南](#)

[FAQ](#)

如果您发现翻译不妥之处，欢迎点击[“反馈”按钮](#)提交详细信息

PXF支持列投影，始终是启用状态。使用列投影时，只从外部数据源返回外部表上的 `SELECT` 查询所需的列。此过程可以提高查询性能，还可以减少传输到Greenplum数据库的数据量。

Note: 某些外部数据源不支持列投影。如果查询访问不支持列投影的数据源，则在没有列投影的情况下执行查询，并且在将数据传输到Greenplum数据库之后对数据进行过滤。

`pxf` 外部表协议自动启用列投影。PXF使用不同的连接器访问外部数据源，列投影支持也由特定的连接器实现决定。以下PXF连接器和配置文件组合支持读取操作的列投影：- PXF Hive Connector, `HiveORC` profile - PXF JDBC Connector, `Jdbc` profile - PXF Hadoop 和 Object Store Connectors, `hdfs:parquet`, `adl:parquet`, `gs:parquet`, `s3:parquet`，和 `wasbs:parquet` profiles - 使用Amazon S3 Select服务, `s3:parquet` 和 `s3:text` 配置文件的PXF S3连接器

Note: 如果无法成功序列化查询过滤条件，PXF可能会禁用列投影；例如，当WHERE子句解析为布尔类型时。

总而言之，必须满足以下所有条件才能进行列投影：

- 外部数据源必须支持列投影。例如，Hive支持ORC格式数据的列投影，某些SQL数据库支持列投影。
- 底层PXF连接器和配置文件实现必须支持列投影。例如，上面标识的PXF Hive和JDBC连接器配置文件支持列投影，支持读取Parquet数据的PXF连接器也支持列投影。
- PXF必须能够序列化查询过滤条件

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ PXF介绍

□ PXF管理手册

□ 配置PXF

升级PXF

PXF的启动、停止和重启

授权用户访问PXF

注册PXF的jar依赖

监控PXF

□ 使用PXF访问hadoop

□ 使用PXF访问Azure, Google云端存储, Minio和S3对象存储

使用PXF访问SQL数据库
(JDBC)

PXF故障排除

□ PXF实用程序手册

Back to the Administrator Guide

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

GPDB集群包括一个master节点和多个segment节点。初始化和配置PXF时，可以在每个GPDB的segment上启动单个PXF JVM进程。

PXF提供到Hadoop, Hive, HBase, 对象存储和外部SQL数据存储的连接器。您必须配置PXF以支持您计划使用的连接器。

配置PXF,你必须:

1. 在每个Greenplum数据库段主机上安装Java包,详情参见[Installing Java for PXF](#)。
2. [Initialize the PXF Service](#)。
3. 如果您计划使用Hadoop, Hive或HBase PXF连接器，则必须执行配置PXF Hadoop连接器中所述的配置过程。[Configuring PXF Hadoop Connectors](#)。
4. 如果您计划使用PXF连接器访问Azure, Google云端存储, Minio或S3对象存储，则必须执行配置Azure连接, Google云端存储, Minio和S3对象存储中所述的配置过程。详见[Configuring Connectors to Azure, Google Cloud Storage, Minio, and S3 Object Stores](#)。
5. 如果计划使用PXF JDBC Connector访问外部SQL数据库，请执行配置JDBC连接器中所述的配置过程。详见[Configuring the JDBC Connector](#)。
6. [Start PXF](#)

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ PXF介绍

□ PXF管理手册

□ 配置PXF

PXF安装和配置目录

为PXF安装JAVA环境

初始化PXF

配置PXF服务器

□ 配置PXF HADOOP连接器 (可选)

配置Minio和S3对象存储的连接器 (可选)

配置Azure和Google云端存储的连接器 (可选)

配置JDBC连接器 (可选)

配置PXF客户端主机和端口(可选)

升级PXF

PXF的启动、停止和重启

授权用户访问PXF

注册PXF的jar依赖

监控PXF

In this topic:

- 准备
- 过程
- 更新hadoop配置

PXF与Cloudera, Hortonworks Data Platform, MapR和通用Apache Hadoop发行版兼容。本主题描述如何配置PXF Hadoop, Hive和HBase连接器。

如果您不想使用与*Hadoop*相关的*PXF*连接器，则无需执行此过程。

准备

配置PXF Hadoop连接器需要将配置文件从Hadoop群集复制到Greenplum master主机。如果你使用的是MapR Hadoop发行版，则还必须将某些JAR文件复制到master主机。在配置PXF Hadoop连接器之前，请确保可以将文件从Hadoop群集中的主机复制到Greenplum数据库master服务器。

过程

在GPDB master主机上执行以下操作去配置PXF映射hadoop的连接器。在你配置连接器完成后，需要运行 `pxf cluster sync` 命令拷贝PXF的配置到Greenplum数据库集群。在此过程中，您将使用 `default`，或创建新的PXF服务器配置。您将Hadoop配置文件复制到Greenplum数据库master主机上的服务器配置目录。您也可以将库复制到 `$PXF_CONF/lib` 以获取MapR支持。然后，您可以将master主机上的PXF配置同步到standby和segment主机。(当您运行 `pxf cluster init` 时，PXF将创建 `$PXF_CONF/*` 目录。)

1. 登录到GPDB master节点:

```
$ ssh gpadmin@<gpmaster>
```

2. 确定您的PXF Hadoop服务器配置的名称。如果您的Hadoop集群是Kerberized，则必须使用 `default` PXF服务器。

3. 如果您没有使用默认的PXF服务器, 请创建

`$PXF_HOME/servers/<server_name>` 目录。例如, 使用以下命令创建名为 `hdp3` 的Hadoop服务器配置:

```
gpadmin@gpmaster$ mkdir $PXF_CONF/servers/hdp3
```

4. 转到服务器目录。例如:

```
gpadmin@gpmaster$ cd $PXF_CONF/servers/default
```

或,

```
gpadmin@gpmaster$ cd $PXF_CONF/servers/hdp3
```

5. PXF服务需要从 `core-site.xml` 和其他Hadoop配置文件中获取需要的信息。使用你选择的工具从你的Hadoop集群namenode节点主机上拷贝 `core-site.xml`、`hdfs-site.xml`、`mapred-site.xml` 和 `yarn-site.xml` 文件到当前主机上(您的文件路径可能会根据使用的Hadoop发行版而有所不同)。例如, 这些命令使用 `scp` 去拷贝文件:

```
gpadmin@gpmaster$ cd $PXF_CONF/servers/default
gpadmin@gpmaster$ scp
hdfsuser@namenode:/etc/hadoop/conf/core-site.xml .
gpadmin@gpmaster$ scp
hdfsuser@namenode:/etc/hadoop/conf/hdfs-site.xml .
gpadmin@gpmaster$ scp
hdfsuser@namenode:/etc/hadoop/conf/mapred-site.xml .
gpadmin@gpmaster$ scp
hdfsuser@namenode:/etc/hadoop/conf/yarn-site.xml .
```

6. 如果你想要使用PXF的HIVE连接器访问hive表的数据, 同样拷贝hive的配置到GPDB master上。例如:

```
gpadmin@gpmaster$ scp hiveuser@hivehost:/etc/hive/conf/hive-
site.xml .
```

7. 如果你想要使用PXF的HBASE连接器访问hbase表数据, 同样需要拷贝hbase的配置到GPDB master上。例如:

```
gpadmin@gpmaster$ scp
hbaseuser@hbasehost:/etc/hbase/conf/hbase-site.xml .
```

- 如果你要使用PXF访问MapR Hadoop发行版，你必须从你的MapR拷贝匹配的JAR文件到GPDB master上(您的文件路径可能会根据使用的MapR版本而有所不同)。例如：这些是使用 `scp` 命令拷贝文件

```
gpadmin@gpmaster$ cd $PXF_CONF/lib
gpadmin@gpmaster$ scp
mapruser@maprhost:/opt/mapr/hadoop/hadoop-
2.7.0/share/hadoop/common/lib/maprfs-5.2.2-mapr.jar .
gpadmin@gpmaster$ scp
mapruser@maprhost:/opt/mapr/hadoop/hadoop-
2.7.0/share/hadoop/common/lib/hadoop-auth-2.7.0-mapr-1707.jar
.
gpadmin@gpmaster$ scp
mapruser@maprhost:/opt/mapr/hadoop/hadoop-
2.7.0/share/hadoop/common/hadoop-common-2.7.0-mapr-1707.jar .
```

- 同步PXF的配置到Greenplum数据库集群。例如：

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster sync
```

- Greenplum数据库最终用户访问Hadoop服务。默认情况下，PXF服务尝试使用GPDB的用户去验证访问HDFS, Hive, and HBase。为了支持此功能，如果要使用这些PXF连接器，则必须为Hadoop以及Hive和HBase配置代理设置。参照[Configuring User Impersonation and Proxying](#) 中的过程为Hadoop服务配置用户模拟和代理，或关闭PXF用户模拟。
- 授予HDFS文件和目录的读取权限，这些文件和目录将作为Greenplum数据库中的外部表进行访问。如果启用了用户模拟(默认设置)，则必须向每个Greenplum数据库用户/角色名称授予此权限，这些用户/角色名称将使用引用HDFS文件的外部表。如果未启用用户模拟，则必须将此权限授予gpadmin用户。
- 如果您的Hadoop集群使用Kerberos保护，则为安全HDFS配置PXF必须为每个segment主机生成Kerberos主体和密钥表。

更新hadoop配置

在PXF服务运行时，如果你想要更新Hadoop、Hive或者HBase的配置，你必须在你的GPDB集群上重新同步PXF的配置并且在每个segment节点上重启pxf服务。例如：

```
gpadmin@gpmaster$ cd $PXF_CONF/servers/<server_name>
```

```
gpadmin@gpmaster$ scp hiveuser@hivehost:/etc/hive/conf/hive-site.xml .
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster sync
```

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

 PXF介绍 PXF管理手册 配置PXF

PXF安装和配置目录

为PXF安装JAVA环境

初始化PXF

配置PXF服务器

 配置PXF HADOOP连接器 (可选)

配置Minio和S3对象存储的连接器 (可选)

配置Azure和Google云端存储的连接器 (可选)

配置JDBC连接器 (可选)

配置PXF客户端主机和端口(可选)

升级PXF

PXF的启动、停止和重启

授权用户访问PXF

注册PXF的jar依赖

监控PXF

In this topic:

- 准备
- 过程

PXF是Java服务。它要求在每个Greenplum数据库主机上安装Java 8或Java 11。

如果已在每个Greenplum数据库主机上安装了适当版本的Java，则无需执行本主题中的过程。

准备

确保在每个Greenplum数据库主机上具有访问Java 8或Java 11的权限，或具有超级用户权限来安装Java 8或Java 11。

过程

执行以下过程，在master,standby和各个segment上安装Java，可以使用 `gpssh` 命令在多个主机上执行。

1. 登录greenplum的master节点

```
$ ssh gpadmin@<gpmaster>
```

2. 创建一个text文件列出你的gp集群的standby节点和segment节点，每行一个主机名。例如，一个叫做 `seglistfile` 的文件

```
mstandby
seglist1
seglist2
seglist3
```

3. 在master,standby节点和各个segment节点上安装Java，然后在每个主机上配置环境变量

1. 安装java包。比如安装java 8:

```
gpadmin@gpmaster$ sudo yum -y install java-1.8.0-openjdk-1.8.0*
gpadmin@gpmaster$ gpssh -e -v -f gghostfile sudo yum -y install java-1.8.0-openjdk-1.8.0*
```

2. 标识Java基本安装目录。如果每台主机上的 `gpadmin` 用户的 `.bashrc` 文件不存在，请更新它以包含 `$JAVA_HOME` 设置。例如，如果您安装了Java 8：

```
gpadmin@gpmaster$ echo 'export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.x86_64/jre' >> /home/gpadmin/.bashrc
gpadmin@gpmaster$ gpssh -e -v -f gghostfile "echo 'export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.x86_64/jre' >> /home/gpadmin/.bashrc"
```

如果安装了Java 11，`JAVA_HOME` 可能是

```
/usr/lib/jvm/java-11-openjdk-11.0.4.11-0.el7_6.x86_64。
```

注意：如果超级用户选择了新安装的Java替代品作为系统缺省值，则为
`JAVA_HOME=/usr/lib/jvm/jre`。

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ PXF介绍

□ PXF管理手册

□ 配置PXF

PXF安装和配置目录

为PXF安装JAVA环境

初始化PXF

配置PXF服务器

□ 配置PXF HADOOP连接器 (可选)

配置Minio和S3对象存储的连接器 (可选)

配置Azure和Google云端存储的连接器 (可选)

配置JDBC连接器 (可选)

配置PXF客户端主机和端口(可选)

升级PXF

PXF的启动、停止和重启

授权用户访问PXF

注册PXF的jar依赖

监控PXF

In this topic:

- [PXF配置属性](#)
 - [过程](#)
- [重置PXF](#)
 - [步骤](#)

你必须显式初始化PXF服务实例。此一次性初始化将创建PXF服务Web应用程序并生成PXF配置文件和模板。

PXF提供了两个你可以在初始化时候使用的管理命令

- `pxf cluster init` - 初始化所有在GP集群中的pxf服务实例
- `pxf init` - 初始化当前节点上的pxf服务实例

PXF还提供了类似的 `reset` 命令，可用于重置PXF配置。

PXF配置属性

PXF同时支持内部的和用户自定义的配置属性。初始化PXF会生成PXF内部配置文件，并设置特定于您的配置的默认属性。初始化PXF还会为用户自定义设置生成配置文件模板，例如自定义配置文件和PXF运行时间和日志记录设置。

PXF内部配置文件位于 `$GPHOME/pxf/conf` 目录，在初始化PXF时，你可以通过指定环境变量 `$PXF_CONF` 来配置用户目录。如果在初始化PXF之前未设置 `$PXF_CONF`，PXF可能会在初始化过程中提示您接受或拒绝默认用户配置目录 `$HOME/pxf`。

Note: 选择可以备份的 `$PXF_CONF` 位置，并确保他在GPDB安装目录之外。

在初始化期间，PXF会根据需要创建 `$PXF_CONF` 目录然后使用子目录和模板文件填充他。有关这些目录和内容的列表，可以参照[PXF User Configuration Directories](#)

准备

GPDB实例初始化PXF之前，请确保：

- GPDB集群是启动并且运行的
- 可以通过 `$PXF_CONF` 指定PXF用户配置目录的文件位置，并且 `gpadmin` 用户也需要对这个目录有写权限

过程

执行一下流程确保在你的GPDB集群的每个节点初始化PXF。

1. 登录GPDB的master节点

```
$ ssh gpadmin@<gpmaster>
```

2. 运行 `pxf cluster init` 命令会在master,standby节点和所有的segment节点主机上初始化PXF服务。如下命令指定 `/usr/local/greenplum-pxf` 作为pxf初始化的用户配置目录

```
gpadmin@gpmaster$ PXF_CONF=/usr/local/greenplum-pxf
$GPHOME/pxf/bin/pxf cluster init
```

`init` 命令创建了pxf网页应用和pxf内部配置。`init`命令也创建了 `$PXF_CONF` 用户配置目录(如果目录不存在)并使用用户可自定义的配置模板填充 `conf` 和 `templates` 目录。如果 `$PXF_CONF` 存在，则PXF仅更新 `templates` 目录。

Note:PXF服务只运行在segment主机上，然而 `pxf cluster init` 命令也会安装在GPDB的master和standby主机上创建PXF用户配置目录。

重置PXF

如果需要，可以将PXF重置为其未初始化状态。如果您指定了错误的 `PXF_CONF` 目录，或者您想从头开始初始化过程，则可以选择重置PXF。

重置PXF时，PXF会提示您确认操作。如果确认，PXF将删除以下运行时文件和目录（其中 `PXF_HOME=$GPHOME/pxf`）：

- `$PXF_HOME/conf/pxf-private.classpath`

- \$PXF_HOME/pxf-service
- \$PXF_HOME/run

重置操作期间PXF不会删除 \$PXF_CONF 目录。

必须先在segment主机上停止PXF服务实例，然后才能在主机上重置PXF。

步骤

执行以下过程在Greenplum数据库集群中的每个segment主机上重置PXF。

1. 登录到Greenplum数据库master节点：

```
$ ssh gpadmin@<gpmaster>
```

2. 在每个segment主机上停止PXF服务实例。例如：

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster stop
```

3. 在所有Greenplum主机上重置PXF服务实例。例如：

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster reset
```

注意：重置PXF之后，必须初始化并启动PXF才能再次使用该服务。

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

[PXF介绍](#)

[PXF管理手册](#)

[配置PXF](#)

PXF安装和配置目录

为PXF安装JAVA环境

初始化PXF

配置PXF服务器

[配置PXF HADOOP连接器 \(可选\)](#)

配置Minio和S3对象存储的连接器 (可选)

配置Azure和Google云端存储的连接器 (可选)

配置JDBC连接器 (可选)

配置PXF客户端主机和端口(可选)

升级PXF

PXF的启动、停止和重启

授权用户访问PXF

注册PXF的jar依赖

监控PXF

In this topic:

- [关于服务器模板文件](#)
- [关于默认服务器](#)
- [配置服务器](#)
- [配置PXF用户](#)
 - [过程](#)
- [关于配置属性优先级](#)
- [使用服务器配置](#)

本主题概述了PXF服务器配置。要配置服务器，请参阅特定于您要配置的连接器的主题。

您可以通过PXF连接器读取数据或将数据写入外部数据存储。要访问外部数据存储，您必须提供服务器位置。您可能还需要提供客户端访问凭据和其他特定于外部数据存储的属性。PXF通过以下方式简化了对外部数据存储的配置访问：

- 支持基于文件的连接器和用户配置
- 提供特定于连接器的模板配置文件

PXF * Server * 定义只是一个命名配置，提供对特定外部数据存储的访问。PXF服务器名称是位于 `$PXF_CONF/servers/` 中的目录的名称。

您在服务器配置中提供的信息是特定于连接器的。例如，PXF JDBC连接器服务器定义可以包括JDBC驱动程序类名称，URL，用户名和密码的设置。您还可以在JDBC服务器定义中配置特定于连接的属性和特定于会话的属性。

PXF为每个连接器提供一个服务器模板文件。此模板标识必须配置才能使用连接器的典型属性集。

您将为Greenplum数据库用户需要访问的每个外部数据存储配置服务器定义。例如，如果您需要访问两个Hadoop集群，则将为每个集群创建PXF Hadoop服务器配置。如果需要访问Oracle和MySQL数据库，则将为每个数据库创建一个或多个PXF JDBC服务器配置。

服务器配置可以包括用户访问凭据的默认设置以及外部数据存储的其他属性。您可以允许Greenplum数据库用户使用默认设置访问外部数据存储，也可以基于每个用户配置访问权限和其他属性。这使您可以在单个PXF服务器定义中使用不同的外部数据存储访问凭据来配置不同的Greenplum数据库用户。您可以允许Greenplum数据库用户使用默认设置访问外部数据存储，也可以基于每个用户配置访问权限和其他属

性。这使您可以在单个PXF服务器定义中使用不同的外部数据存储访问凭据来配置不同的Greenplum数据库用户。

关于服务器模板文件

PXF服务器的配置信息位于 `$PXF_CONF/servers/<server_name>/` 中的一个或多个 `<connector> -site.xml` 文件中。

PXF为每个连接器提供一个模板配置文件。初始化PXF后，这些服务器模板配置文件位于 `$PXF_CONF/templates/` 目录中：

```
gpadmin@gpmaster$ ls $PXF_CONF/templates
adl-site.xml  hbase-site.xml  jdbc-site.xml  s3-site.xml
core-site.xml  hdfs-site.xml  mapred-site.xml  wasbs-site.xml
gs-site.xml    hive-site.xml   minio-site.xml  yarn-site.xml
```

例如，`s3-site.xml` 模板文件的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <property>
    <name>fs.s3a.access.key</name>
    <value>YOUR_AWS_ACCESS_KEY_ID</value>
  </property>
  <property>
    <name>fs.s3a.secret.key</name>
    <value>YOUR_AWS_SECRET_ACCESS_KEY</value>
  </property>
  <property>
    <name>fs.s3a.fast.upload</name>
    <value>true</value>
  </property>
</configuration>
```

您可以在配置文件中以明文形式为PXF指定凭据。

Note: Hadoop连接器的模板文件不打算被修改并用于配置，因为它们仅提供所需信息的示例。您无需修改Hadoop模板，而是将几个Hadoop `* -site.xml` 文件从Hadoop集群复制到PXF Hadoop服务器配置。

关于默认服务器

PXF定义了一个名为 `default` 的特殊服务器。初始化PXF时，它会自

动创建一个 `$PXF_CONF/servers/default/` 目录。此目录最初为空，标识默认的PXF服务器配置。您可以配置默认PXF服务器并将其分配给任何外部数据源。例如，您可以将PXF默认服务器分配给Hadoop集群，或者分配给用户经常访问的MySQL数据库。

如果您在 `CREATE EXTERNAL TABLE` 命令 `LOCATION` 子句中省略了 `SERVER=<server_name>` 设置，则PXF将自动使用 `default` 服务器配置。

Note: 当您的Hadoop集群使用Kerberos身份验证时，您必须将Hadoop服务器配置为PXF `default` 服务器。

配置服务器

将PXF连接器配置为外部数据存储时，将为该连接器添加命名的PXF服务器配置。在执行的任务中，您可以：

1. 确定是要配置 `default` PXF服务器，还是为服务器配置选择新名称。
2. 创建目录 `$PXF_CONF/servers/<server_name>`。
3. 将模板或其他配置文件复制到新的服务器目录。
4. 为模板文件中的属性填写适当的默认值。
5. 添加环境所需的所有其他配置属性和值。
6. 如[关于配置PXF用户](#)中所述为服务器配置配置一个或多个用户。
7. 将服务器和用户配置同步到Greenplum数据库集群。

Note: 添加或更新PXF服务器配置后，必须将PXF配置重新同步到Greenplum数据库集群。

配置PXF服务器后，将服务器名称发布给需要访问数据存储的Greenplum数据库用户。用户只需要在创建访问外部数据存储的外部表时提供服务器名称即可。PXF从驻留在由服务器名称标识的服务器配置目录中的服务器和用户配置文件中获取外部数据源位置和访问凭据。

要配置PXF服务器，请参考连接器配置主题：

- 要为Hadoop配置PXF服务器，请参考[配置PXF Hadoop连接器](#)。

- 要为对象存储配置PXF服务器, 请参阅[为Azure, Google Cloud Storage, Minio和S3对象存储配置连接器](#).
- 要配置PXF JDBC服务器, 请参阅[配置JDBC连接器](#).

配置PXF用户

您可以基于每个服务器, 每个Greenplum用户配置对外部数据存储的访问。

您可以通过在PXF服务器配置目录

`$PXF_CONF/servers/<server_name>/` 中提供

`<greenplum_user_name>-user.xml` 用户配置文件来为特定

的Greenplum数据库用户配置外部数据存储用户访问凭据和属性。例如, 您在 `$PXF_CONF/servers/<server_name>/bill-user.xml` 文件中为名为 `bill` 的Greenplum数据库用户指定属性。您可以在PXF服务器配置中配置零个, 一个或多个用户。

您在用户配置文件中指定的属性是特定于连接器的。您可以在

`<greenplum_user_name>-user.xml` 配置文件中指定PXF连接器服务器支持的任何配置属性。

例如, 假设您已经在名为 `pgsrv1` 的PXF JDBC服务器配置中配置了对PostgreSQL数据库的访问。要允许名为 `bill` 的Greenplum数据库用户以名为 `pguser1` 的PostgreSQL用户(密码为 `changeme`)访问该数据库, 请使用以下命令创建用户包含如下属性的配置文件

`$PXF_CONF/servers/pgsrv1/bill-user.xml` :

```

<configuration>
  <property>
    <name>jdbc.user</name>
    <value>pguser1</value>
  </property>
  <property>
    <name>jdbc.password</name>
    <value>changeme</value>
  </property>
</configuration>

```

如果要为 `bill` 配置特定的搜索路径和较大的读取大小, 则还应将以下属性添加到 `bill-user.xml` 用户配置文件中:

```

<property>
  <name>jdbc.session.property.search_path</name>

```

```

<value>bill_schema</value>
</property>
<property>
  <name>jdbc.statement.fetchSize</name>
  <value>2000</value>
</property>

```

过程

对于要配置的每个PXF用户，您将：

1. 标识Greenplum数据库用户的名称。
2. 标识要为其配置用户访问权限的PXF服务器定义。
3. 标识要为用户配置的每个属性的名称和值。
4. 创建/编辑文件

```
$PXF_CONF/servers/<server_name>/<greenplum_user_name>-user.xml
```

， 并添加外部配置块：

```

<configuration>
</configuration>

```

5. 将在步骤3中标识的每个属性/值对添加到 `<greenplum_user_name>-user.xml` 文件中的配置块中。
6. 如果要将PXF用户配置添加到以前配置的PXF服务器定义中，请将该用户配置同步到Greenplum数据库集群。

关于配置属性优先级

PXF服务器配置可能包括用于用户访问凭据的默认设置以及用于访问外部数据存储的其他属性。一些PXF连接器(例如S3和JDBC连接器)允许您通过 `CREATE EXTERNAL TABLE` 命令 `LOCATION` 子句中的自定义选项直接指定某些服务器属性。`<greenplum_user_name>-user.xml` 文件为Greenplum数据库用户指定了外部数据存储的属性设置。

对于给定的Greenplum数据库用户，PXF使用以下优先级规则(从高到低)获取该用户的配置属性设置：

1. 您在 `<server_name>/<greenplum_user_name>-user.xml` 中配置的属性将覆盖其他位置的属性设置。
2. 通过 `CREATE EXTERNAL TABLE` 命令的 `LOCATION` 子句中的自定义选项指定的属性将覆盖PXF服务器配置中该属性的任何设置。
3. 您在PXF服务器定义 `<server_name>` 中配置的属性标识默认属性值。

这些优先级规则使您可以创建一个可由多个Greenplum数据库用户访问的外部表，每个用户都有各自独特的外部数据存储用户凭据。

使用服务器配置

为了访问外部数据存储，Greenplum数据库用户在 `CREATE EXTERNAL TABLE` 命令中的 `LOCATION` 子句 `SERVER=<server_name>` 选项中指定服务器名称。用户提供的 `<server_name>` 标识服务器配置目录，PXF从该目录获取配置和凭据以访问外部数据存储。

例如，以下命令使用 `$PXF_CONF/servers/s3srvcfg/s3-site.xml` 文件中定义的服务器配置访问S3对象存储：

```
CREATE EXTERNAL TABLE pxf_ext_tbl(name text, orders int)
  LOCATION ('pxf://BUCKET/dir/file.txt?
PROFILE=s3:text&SERVER=s3srvcfg')
  FORMAT 'TEXT' (delimiter=E',');
```

如果未提供 `SERVER=<server_name>` 设置，则PXF自动使用 `default` 服务器配置。

例如，如果 `default` 服务器配置标识了Hadoop集群，则以下示例命令将引用位于 `/path/to/file.txt` 的HDFS文件：

```
CREATE EXTERNAL TABLE pxf_ext_hdfs(location text, miles
int)
  LOCATION ('pxf://path/to/file.txt?PROFILE=hdfs:text')
  FORMAT 'TEXT' (delimiter=E',');
```

查询或写入外部表的Greenplum数据库用户使用为 `<server_name>` 用户配置的凭据访问外部数据存储。如果没有为 `<server_name>` 配置任何用户专用的凭据，则Greenplum用户将使用为 `<server_name>` 配置

的默认凭据访问外部数据存储。

(可选)

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

[PXF介绍](#)

[PXF管理手册](#)

[配置PXF](#)

PXF安装和配置目录

为PXF安装JAVA环境

初始化PXF

配置PXF服务器

[配置PXF HADOOP连接器（可选）](#)

配置Minio和S3对象存储的连接器（可选）

配置Azure和Google云端存储的连接器（可选）

配置JDBC连接器（可选）

配置PXF客户端主机和端口（可选）

升级PXF

PXF的启动、停止和重启

授权用户访问PXF

注册PXF的jar依赖

监控PXF

In this topic:

- [关于对象存储配置](#)
- [Minio服务器配置](#)
- [S3服务器配置](#)
 - [配置S3服务器端加密](#)
- [示例服务器配置过程](#)

您可以使用PXF访问S3兼容的对象存储。本主题描述如何将PXF连接器配置为这些外部数据源。

如果您不打算使用这些PXF对象存储连接器，则不需要执行此过程。

关于对象存储配置

要访问对象存储中的数据，必须提供服务器位置和客户端凭据。配置PXF对象库连接器时，请为连接器添加至少一个名为PXF服务器的配置，如[配置PXF服务器](#)中所述。

PXF为每个对象存储连接器提供了一个模板配置文件。这些模板文件位于`$PXF_CONF/templates/`目录中。

Minio服务器配置

Minio的模板配置文件为`$PXF_CONF/templates/minio-site.xml`。配置Minio服务器时，必须提供以下服务器配置属性，并用凭据替换模板值：

属性	描述	值
<code>fs.s3a.endpoint</code>	要连接的Minio S3端点。	您的端点。
<code>fs.s3a.access.key</code>	Minio帐户访问密钥ID。	您的访问密钥。
<code>fs.s3a.secret.key</code>	与Minio访问密钥ID相关联的密钥。	您的密钥。

S3服务器配置

S3的模板配置文件为 `$PXF_CONF/templates/s3-site.xml`。配置S3服务器时，必须提供以下服务器配置属性，并用凭据替换模板值：

属性	描述	值
<code>fs.s3a.access.key</code>	AWS账户访问密钥ID。	您的访问密钥。
<code>fs.s3a.secret.key</code>	与AWS访问密钥ID关联的密钥。	您的密钥。

如果需要，可以通过指定 `s3-site.xml` 服务器配置文件中的Hadoop-AWS模块文档的[S3A](#) 部分来调优PXF S3连接。

您可以通过CREATE EXTERNAL TABLE命令LOCATION子句中的自定义选项直接指定S3访问ID和密钥，从而覆盖S3服务器配置的凭据。有关其他信息，请参考[使用DDL覆盖S3服务器配置](#)。

配置S3服务器端加密

PXF支持使用可读写的Greenplum数据库外部表（指定 `pxf` 协议和 `s3:*` 配置文件）访问的S3文件的Amazon Web Service S3服务器端加密（SSE）。AWS S3服务器端加密可安静地保护您的数据；它会在将对象数据写入磁盘时对其进行加密，并在您访问数据时为您透明地解密数据。

PXF支持以下AWS SSE加密密钥管理方案：

- 具有S3受管密钥的SSE(SSE-S3) - Amazon管理数据和master加密密钥。
- 具有密钥管理服务托管密钥的SSE(SSE-KMS) - Amazon管理数据密钥，而您在AWS KMS中管理加密密钥。
- SSE与客户提供的密钥(SSE-C) - 您设置和管理加密密钥。

无论数据是否经过加密，您的S3访问密钥和加密密钥都将控制您对所有S3存储桶对象的访问。

在您通过指定 `pxf` 协议和 `s3:*` 配置文件创建的可读外部表访问的加密文件的读取操作期间，S3透明地解密数据。无需其他配置。

要加密通过这种类型的外部表写入S3的数据，您有两个选择：

- 通过AWS控制台或命令行工具（建议），在每个S3存储桶的基础上配置默认的SSE加密密钥管理方案。
- 在PXF S3服务器的 `s3-site.xml` 配置文件中配置SSE加密选项。

通过S3存储桶策略配置SSE（推荐）

您可以创建S3桶策略`</i>`，以标识要加密的对象，加密密钥管理方案以及在这些对象上允许的写入操作。请参阅AWS S3文档中的[使用服务器端加密保护数据](#)，以获取有关SSE加密密钥管理方案的更多信息。[如何为S3存储桶启用默认加密？](#)介绍了如何设置默认加密存储桶策略。

在PXF S3服务器配置中指定SSE选项

您必须在 `s3-site.xml` 中包含某些属性，才能在PXF S3服务器配置中配置服务器端加密。您添加到文件中的属性和值取决于SSE加密密钥管理方案。

SSE-S3

要在写入任何S3存储桶的任何文件上启用SSE-S3，请在 `s3-site.xml` 文件中设置以下加密算法属性和值：

```
<property>
  <name>fs.s3a.server-side-encryption-algorithm</name>
  <value>AES256</value>
</property>
```

要为特定的S3存储桶启用SSE-S3，请使用包含存储桶名称的属性名称变体。例如：

```
<property>
  <name>fs.s3a.bucket.YOUR_BUCKET1_NAME.server-side-encryption-
algorithm</name>
  <value>AES256</value>
</property>
```

将 `YOUR_BUCKET1_NAME` 替换为S3存储桶的名称。

SSE-KMS

要在您写入任何S3存储桶的任何文件上启用SSE-KMS，请同时设置加密算法和加密密钥ID。要在 `s3-site.xml` 文件中设置这些属性：

```
<property>
  <name>fs.s3a.server-side-encryption-algorithm</name>
  <value>SSE-KMS</value>
</property>
<property>
  <name>fs.s3a.server-side-encryption.key</name>
  <value>YOUR_AWS_SSE_KMS_KEY_ARN</value>
</property>
```

用您的密钥资源名称替换 `YOUR_AWS_SSE_KMS_KEY_ARN`。如果您未指定加密密钥，则使用Amazon KMS中定义的默认密钥。`KMS`密钥示例：

```
arn:aws:kms:us-west-2:123456789012:key/1a23b456-7890-12cc-
d345-6ef7890g12f3
```

◦

注意：确保在同一Amazon可用区中创建存储桶和密钥。

要为特定的S3存储桶启用SSE-KMS，请使用包含存储桶名称的属性名称变体。例如：

```
<property>
  <name>fs.s3a.bucket.YOUR_BUCKET2_NAME.server-side-encryption-
algorithm</name>
  <value>SSE-KMS</value>
</property>
<property>
  <name>fs.s3a.bucket.YOUR_BUCKET2_NAME.server-side-
encryption.key</name>
  <value>YOUR_AWS_SSE_KMS_KEY_ARN</value>
</property>
```

将 `YOUR_BUCKET2_NAME` 替换为S3存储桶的名称。

SSE-C

要在写入任何S3存储桶的任何文件上启用SSE-C，请同时设置加密算法和加密密钥（base-64编码）。所有客户端必须共享相同的密钥。

要在 `s3-site.xml` 文件中设置这些属性：

```
<property>
  <name>fs.s3a.server-side-encryption-algorithm</name>
  <value>SSE-C</value>
</property>
```

```

<property>
  <name>fs.s3a.server-side-encryption.key</name>
  <value>YOUR_BASE64-ENCODED_ENCRYPTION_KEY</value>
</property>

```

要为特定的S3存储桶启用SSE-C，请使用包含存储桶名称的属性名称变体，如SSE-KMS示例中所述。

示例服务器配置过程

在配置对象存储连接器服务器之前，请确保已初始化PXF。

在此过程中，您将在S3云存储连接器的Greenplum数据库master主机上的`$PXF_CONF/servers`目录中命名并添加PXF服务器配置。然后，您可以使用`pxf cluster sync`命令将服务器配置同步到Greenplum数据库集群。

1. 登录到您的Greenplum数据库主节点：

```
$ ssh gpadmin@<gpmaster>
```

2. 选择服务器的名称。您将为需要引用对象存储中文件的最终用户提供名称。

3. 创建`$PXF_HOME/servers/<server_name>`目录。例如，使用以下命令为名为`s3_user1cfg`的S3服务器创建服务器配置：

```
gpadmin@gpmaster$ mkdir $PXF_CONF/servers/s3_user1cfg
```

4. 将S3的PXF模板文件复制到服务器配置目录。例如：

```
gpadmin@gpmaster$ cp $PXF_CONF/templates/s3-site.xml
$PXF_CONF/servers/s3_user1cfg/
```

5. 在您选择的编辑器中打开模板服务器配置文件，并为您的环境提供适当的属性值。例如：

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <property>
    <name>fs.s3a.access.key</name>
    <value>access_key_for_user1</value>
  </property>

```

```
<property>
  <name>fs.s3a.secret.key</name>
  <value>secret_key_for_user1</value>
</property>
<property>
  <name>fs.s3a.fast.upload</name>
  <value>true</value>
</property>
</configuration>
```

6. 保存更改并退出编辑器。
7. 使用 `pxf cluster sync` 命令将新的服务器配置复制到 *Greenplum* 数据库集群。例如：

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster sync
```

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ PXF介绍

□ PXF管理手册

□ 配置PXF

PXF安装和配置目录

为PXF安装JAVA环境

初始化PXF

配置PXF服务器

□ 配置PXF HADOOP连接器（可选）

配置Minio和S3对象存储的连接器
(可选)配置Azure和Google云端存储的连接器
(可选)

配置JDBC连接器（可选）

配置PXF客户端主机和端口(可选)

In this topic:

- 对象存储配置
 - [Azure Blob存储服务配置](#)
 - [Azure数据湖服务配置](#)
 - [Google Cloud Storage服务配置](#)
- [示例服务器配置过程](#)

您可以使用PXF访问Azure Data Lake， Azure Blob存储和Google Cloud Storage对象存储。这个章节主要描述如何配置PXF连接器访问这个外部数据源。

如果您不打算使用PXF对象存储连接器，则不需要执行此过程。

对象存储配置

要访问对象存储中的数据，必须提供服务器位置和客户端凭据。配置PXF对象库连接器时，请为连接器添加至少一个名为PXF服务器的配置，如[配置PXF服务器](#)中所述。

PXF为每个对象存储连接器提供了一个模板配置文件。这些模板文件位于`$PXF_CONF/templates/`目录中。

Azure Blob存储服务配置

Azure Blob存储服务的默认配置文件是`$PXF_CONF/templates/wasbs-site.xml`。如果你想要配置Azure Blob存储服务，你需要提供如下的配置项并且替换模板中的值。

Property	Description	Value
<code>fs.adl.oauth2.access.token.provider.type</code>	令牌类型	必须指定 <code>clientCredential</code> 。
<code>fs.azure.account.key.<YOUR_AZURE_BLOB_STORAGE_ACCOUNT_NAME>.blob.core.windows.net</code>	Azure帐户密钥	用你的账户秘钥替换
<code>fs.AbstractFileSystem.wasbs.impl</code>	文件系统类名称	必须指定 <code>org.apache.hadoop.fs.azure.Wasbs</code>

Azure数据湖服务配置

Azure数据湖服务默认模板文件是 `$PXF_CONF/templates/adl-site.xml`，当你配置Azure数据湖服务时需要配置如下的配置项并替换模板中的值：

Property	Description	Value
<code>fs.adl.oauth2.access.token.provider.type</code>	令牌类型	必须指定 <code>clientCredential</code> 。
<code>fs.adl.oauth2.refresh.url</code>	要连接的Azure结束位置	Your refresh URL.
<code>fs.adl.oauth2.client.id</code>	Azure账户的客户端ID	Your client ID (UUID).
<code>fs.adl.oauth2.credential</code>	Azure账户的客户端ID的密码	Your password.

Google Cloud Storage服务配置

Google Cloud Storage服务默认配置模板文件是 `$PXF_CONF/templates/gs-site.xml`，当你配置Google Cloud Storage服务的时候需要修改如下的配置项和值：

Property	Description	Value
<code>google.cloud.auth.service.account.enable</code>	启用服务帐户授权	Must specify <code>true</code> .
<code>google.cloud.auth.service.account.json.keyfile</code>	Google Storage的秘钥文件	Path to your key file.
<code>fs.AbstractFileSystem.gs.impl</code>	文件系统类名称	Must specify <code>com.google.cloud.hadoop.fs.gcs.GoogleHadoopFS</code>

示例服务器配置过程

在配置对象存储连接器服务器之前，请确保已初始化PXF。

在此过程中，您需要在Greenplum数据库master主机上的 `$PXF_CONF/servers` 目录中为Google Cloud Storage (GCS) 连接器命名并添加PXF服务器配置。然后，您可以使用 `pxf cluster sync` 命令将服务器配置同步到Greenplum数据库集群。

1. 登录到GPDB master主机

```
$ ssh gpadmin@<gpmaster>
```

2. 选择服务器的名称。您将为需要引用对象存储中文件的最终用户提供名称。
3. 创建 `$PXF_HOME/servers/<server_name>` 目录。例如，使用以下命令为名为 `gs_public` 的Google Cloud Storage服务器创建服务器配置：

```
gpadmin@gpmaster$ mkdir $PXF_CONF/servers/gs_public
```

4. 将GCS的PXF模板文件复制到服务器配置目录。例如：

```
gpadmin@gpmaster$ cp $PXF_CONF/templates/gs-site.xml $PXF_CONF/servers/gs_public/
```

5. 在您选择的编辑器中打开模板服务器配置文件，并为您的环境提供适当的属性值。例如，如果您的Google Cloud Storage密钥文件位于 `/home/gpadmin/keys/gcs-account.key.json` 中：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <property>
    <name>google.cloud.auth.service.account.enable</name>
    <value>true</value>
  </property>
  <property>
    <name>google.cloud.auth.service.account.json.keyfile</name>
    <value>/home/gpadmin/keys/gcs-account.key.json</value>
  </property>
  <property>
    <name>fs.AbstractFileSystem.gs.impl</name>
    <value>com.google.cloud.hadoop.fs.gcs.GoogleHadoopFS</value>
  </property>
</configuration>
```

6. 保存更改并退出编辑器。
7. 使用 `pxf cluster sync` 命令将新的服务器配置复制到Greenplum数据库集群。例如：

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster sync
```

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ PXF介绍

□ PXF管理手册

□ 配置PXF

PXF安装和配置目录

为PXF安装JAVA环境

初始化PXF

配置PXF服务器

□ 配置PXF HADOOP连接器（可选）

配置Minio和S3对象存储的连接器（可选）

配置Azure和Google云端存储的连接器（可选）

配置JDBC连接器（可选）

配置PXF客户端主机和端口（可选）

升级PXF

PXF的启动、停止和重启

授权用户访问PXF

注册PXF的jar依赖

监控PXF

In this topic:

- [关于JDBC配置](#)
- [JDBC驱动程序JAR注册](#)
- [JDBC服务器配置](#)
 - [连接级属性](#)
 - [连接事务隔离属性](#)
 - [语句级属性](#)
 - [会话级属性](#)
 - [关于JDBC用户模拟](#)
 - [关于JDBC连接池](#)
- [JDBC命名查询配置](#)
 - [定义命名查询](#)
 - [查询命名](#)
- [覆盖JDBC服务器配置](#)
- [示例配置过程](#)
- [配置Hive访问](#)
 - [JDBC服务器配置](#)
 - [示例配置步骤](#)

您可以使用PXF访问外部SQL数据库，包括MySQL, ORACLE, PostgreSQL, Hive和Apache Ignite。本主题描述如何配置PXF JDBC连接器以访问这些外部数据源。

如果您不打算使用PXF JDBC连接器，则不需要执行此过程。

关于JDBC配置

要使用PXF JDBC连接器访问外部SQL数据库中的数据，您必须：

- 注册兼容的JDBC驱动程序JAR文件
- 指定JDBC驱动程序类名称，数据库URL和客户端凭据

在以前的Greenplum数据库版本中，您可能已通过 `CREATE EXTERNAL TABLE` 命令中的选项指定了JDBC驱动程序类名称，数据库URL和客户端凭据。PXF现在支持JDBC连接器的基于文件的服务器配置。如下所述，此配置使您可以在文件中指定这些选项和凭据。

注意：您先前创建的直接指定JDBC连接选项的PXF外部表将继续起作用。如果要移动这些表以使用基于JDBC文件的服务器配置，则必须创建服务器配置，删除外部表，然后重新创建表，并指定适当的 `SERVER=<server_name>` 子句。

JDBC驱动程序JAR注册

PXF JDBC连接器随 `postgresql-42.2.5.jar` JAR文件一起安装。如果您需要其他JDBC驱动程序，请确保在每个segment主机的 `$PXF_CONF/lib` 目录中为外部SQL数据库安装JDBC驱动程序JAR文件。确保安装与您的JRE版本兼容的JDBC驱动程序JAR文件。有关更多信息，请参见[注册PXF JAR依赖项](#)。

JDBC服务器配置

配置PXF JDBC连接器时，请为连接器添加至少一个名为PXF服务器的配置，如[配置PXF服务器](#)中所述。您还可以配置一个或多个静态定义的查询以对远程SQL数据库运行。

PXF提供了JDBC连接器的模板配置文件。该服务器模板配置文件位于 `$PXF_CONF/templates/jdbc-site.xml` 中，标识可以配置以建立与外部SQL数据库的连接的属性。该模板还包括可选属性，您可以在执行查询或在外部数据库会话中插入命令之前设置这些可选属性。

服务器模板文件 `jdbc-site.xml` 中的必需属性如下：

属性	描述	值
<code>jdbc.driver</code>	JDBC驱动类名称	JDBC驱动程序Java类名；例如 <code>org.postgresql.Driver</code> 。
<code>jdbc.url</code>	JDBC驱动程序用于连接数据库的URL	数据库连接URL(特定于数据库)，例如： <code>jdbc:postgresql://phost:pport/pdatabase</code> 。
<code>jdbc.user</code>	数据库用户名	连接数据库的用户名。
<code>jdbc.password</code>	<code>jdbc.user</code> 的密码	连接数据库的密码。

配置PXF JDBC服务器时，可以在配置文件中以明文形式为PXF指定外部数据库用户凭据。

连接级属性

要设置其他JDBC连接级别的属性，请将 `jdbc.connection.property.<CPROP_NAME>` 属性添加到 `jdbc-site.xml` 中。当PXF建立到外部SQL数据库（`DriverManager.getConnection()`）的连接时，会将这些属性传递给JDBC驱动程序。

将 `<CPROP_NAME>` 替换为连接属性名称，并指定其值：

属性	描述	值
<code>jdbc.connection.property.<CPROP_NAME></code>	当PXF建立与外部SQL数据库的连接时，传递给JDBC驱动程序的属性名称(<CPROP_NAME>)。	<CPROP_NAME>属性的值。

示例：要在与PostgreSQL数据库的JDBC连接上设置 `createDatabaseIfNotExist` 连接属性，请在 `jdbc-site.xml` 中包含以下属性块：

```
<property>
  <name>jdbc.connection.property.createDatabaseIfNotExist</name>
  <value>true</value>
</property>
```

确保外部SQL数据库的JDBC驱动程序支持您指定的任何连接级属性。

连接事务隔离属性

SQL标准定义了四个事务隔离级别。您为与外部SQL数据库的给定连接指定的级别决定了对另一连接如何和何时可见在该连接上执行的一个事务所做的更改。

PXF JDBC连接器公开了一个名为 `jdbc.connection.transactionIsolation` 的可选服务器配置属性，该属性使您可以指定事务隔离级别。建立与外部SQL数据库的连接后，PXF会设置级别 (`setTransactionIsolation()`)。

JDBC连接器支持以下 `jdbc.connection.transactionIsolation` 属性值：

SQL 级别	PXF属性值
未提交读	READ_UNCOMMITTED
已提交读	READ_COMMITTED
可重复读	REPEATABLE_READ
可串行化	SERIALIZABLE

例如，要将事务隔离级别设置为未提交读，请将以下属性块添加到 `jdbc-site.xml` 文件中：

```
<property>
  <name>jdbc.connection.transactionIsolation</name>
  <value>READ_UNCOMMITTED</value>
</property>
```

不同的SQL数据库支持不同的事务隔离级别。确保外部数据库支持您指定的级别。

语句级属性

PXF JDBC连接器通过语句在外部SQL数据库表上执行查询或插入命令。连接器公开的属性使您可以在外部数据库中执行命令之前配置语句的某些方面。连接器支持以下语句级属性：

属性	描述	值
<code>jdbc.statement.batchSize</code>	批量写入外部数据库表的行数。默认的写入批处理大小为100。	行数。默认的写入批处理大小为100。
<code>jdbc.statement.fetchSize</code>	从外部数据库表读取时要提取/缓冲的行数。默认读取大小为1000。	行数。默认读取大小为1000。

<pre>jdbc.statement.queryTimeout</pre>	JDBC驱动程序等待语句执行的时间（以秒为单位）。此超时适用于为读取和写入操作创建的语句。	超时时间（以秒为单位）。默认等待时间是无限的。
--	---	-------------------------

对于您未明确配置的任何语句级属性，PXF使用默认值。

示例：要将读取获取大小设置为5000，请将以下属性块添加到 `jdbc-site.xml` 中：

```
<property>
  <name>jdbc.statement.fetchSize</name>
  <value>5000</value>
</property>
```

确保外部SQL数据库的JDBC驱动程序支持您指定的任何语句级属性。

会话级属性

要设置会话级别的属性，请在 `jdbc-site.xml` 中添加 `jdbc.session.property.<SPROP_NAME>` 属性。在执行查询之前，PXF将在外部数据库中 `SET` 这些属性。

将 `<SPROP_NAME>` 替换为会话属性名称，并指定其值：

属性	描述	值
<code>jdbc.session.property.<SPROP_NAME></code>	在执行查询之前要设置的会话属性的名称(<code><SPROP_NAME></code>)。	<code><SPROP_NAME></code> 属性的值。

注意：PXF JDBC连接器完全按照 `jdbc-site.xml` 服务器配置文件中的指定，将会话属性名和属性值都传递给外部SQL数据库。为了限制SQL注入的潜在威胁，连接器拒绝包含 `;`，`\n`，`\b`，或 `\0` 字符的任何属性名称或值。

PXF JDBC连接器为所有支持的外部SQL数据库处理会话属性 `SET` 语法。

示例：要在PostgreSQL数据库中运行查询之前设置 `search_path` 参数，请将以下属性块添加到 `jdbc-site.xml` 中：

```
<property>
  <name>jdbc.session.property.search_path</name>
  <value>public</value>
</property>
```

确保外部SQL数据库的JDBC驱动程序支持您指定的任何属性。

关于JDBC用户模拟

PXF JDBC连接器使用 `jdbc.user` 设置或 `jdbc.url` 中的信息来确定连接到外部数据存

储的用户身份。禁用PXF JDBC用户模拟时（默认设置），JDBC连接器的行为进一步取决于外部数据存储。例如，如果您使用JDBC连接器访问Hive，则连接器将使用某些Hive身份验证和模拟属性的设置来确定用户。您可能需要提供 `jdbc.user` 设置，或在服务器 `jdbc-site.xml` 文件中的 `jdbc.url` 设置中添加属性。

启用PXF JDBC用户模拟时，PXF JDBC连接器代表Greenplum数据库最终用户访问外部数据存储。连接器使用访问PXF外部表的Greenplum数据库用户的名称来尝试连接到外部数据存储。

`pxf.impersonation.jdbc` 属性控制JDBC用户模拟。默认情况下，禁用JDBC用户模拟。要为服务器配置启用JDBC用户模拟，请将属性设置为true：

```
<property>
  <name>pxf.impersonation.jdbc</name>
  <value>true</value>
</property>
```

为PXF服务器启用JDBC用户模拟时，PXF会覆盖在 `jdbc-site.xml` 或 `<greenplum_user_name>-user.xml` 中定义或在外部表DDL中指定的 `jdbc.user` 属性设置的值，以及Greenplum数据库用户名。为了在外部数据存储区需要密码来验证连接用户的身份时有效地模拟用户，必须为可以模拟在该用户的 `<greenplum_user_name>-user.xml` 属性覆盖文件中的每个用户指定 `jdbc.password` 设置。有关每个服务器，每个Greenplum用户配置的更多信息，请参考[配置PXF用户](#)。

关于JDBC连接池

PXF JDBC连接器使用由[HikariCP] (<https://github.com/brettwooldridge/HikariCP>) 实现的JDBC连接池。当用户查询或写入外部表时，连接器在首次遇到 `jdbc.url`，`jdbc.user`，`jdbc.password`，连接属性和池属性设置的唯一组合时，将为关联的服务器配置建立连接池。连接器会根据某些连接和超时设置重用池中的连接。

对于给定的服务器配置，可能存在一个或多个连接池，并且用户访问指定同一服务器的不同外部表可能会共享一个连接池。

注意：如果在服务器配置中启用了JDBC用户模拟，则JDBC连接器将为每个Greenplum数据库用户创建一个单独的连接池，该用户访问指定该服务器配置的任何外部表。

`jdbc.pool.enabled` 属性控制着服务器配置的JDBC连接池。默认情况下启用连接池。要为服务器配置禁用JDBC连接池，请将属性设置为false：

```
<property>
  <name>jdbc.pool.enabled</name>
  <value>false</value>
</property>
```

如果为服务器配置禁用JDBC连接池，则PXF不会为该服务器重用JDBC连接。PXF为每个分区的查询创建到远程数据库的连接，并在该分区的查询完成时关闭该连接。

PXF公开了可以在JDBC服务器定义中配置的连接池属性。这些属性以

`jdbc.pool.property.` 前缀命名，并且应用于每个PXF JVM。 JDBC连接器自动设置以下连接池属性和默认值：

属性	描述	默认值
<code>jdbc.pool.property.maximumPoolSize</code>	与数据库后端的最大连接数。	5
<code>jdbc.pool.property.connectionTimeout</code>	等待来自池的连接的最长时间（以毫秒为单位）。	30000
<code>jdbc.pool.property.idleTimeout</code>	最长时间（以毫秒为单位），在此时间之后，不活动的连接被视为空闲。	30000
<code>jdbc.pool.property.minimumIdle</code>	连接池中维护的最小空闲连接数。	0

您可以通过指定 `jdbc.pool.property.<HIKARICP_PROP_NAME>` 以及服务器的 `jdbc-site.xml` 配置文件中的所需值来为服务器配置设置其他HikariCP特定的连接池属性。还要注意，当JDBC连接器请求来自JDBC `DriverManager` 的连接时，它会传递您用 `jdbc.connection.property.` 前缀指定的任何属性。请参考上面的[连接级属性](#)。

调整最大连接池大小

为了不超过目标数据库所允许的最大连接数，并同时确保每个PXF JVM服务公平共享的JDBC连接，请根据Greenplum数据库集群的大小确定 `maxPoolSize` 的最大值。如下：

```
max_conns_allowed_by_remote_db / #_greenplum_segment_hosts
```

例如，如果您的Greenplum数据库集群具有16个segment主机，并且目标数据库允许160个并发连接，则按以下方式计算 `maxPoolSize`：

```
160 / 16 = 10
```

实际上，您可以选择将 `maxPoolSize` 设置为较低的值，因为每个JDBC查询的并发连接数取决于查询中使用的分区数。当查询不使用分区时，单个PXF JVM将为查询提供服务。如果查询使用12个分区，则PXF将建立与远程数据库的12个并发JDBC连接。理想情况下，这些连接在PXF JVM之间平均分配，但这不能保证。

JDBC命名查询配置

PXF*命名查询*是您配置的静态查询，并且PXF在远程SQL数据库中运行。

要配置和使用PXF JDBC命名查询：

1. 您[定义查询](#)在文本文件中。
2. 您向Greenplum数据库用户提供[查询名称](#)。

3. Greenplum数据库用户在Greenplum数据库外部表定义中引用查询。

每当用户在Greenplum数据库外部表上调用 `SELECT` 命令时，PXF都会运行查询。

定义命名查询

通过将查询语句添加到具有以下命名格式的文本文件中来创建命名查询：

`<query_name>.sql`。您可以为JDBC服务器配置定义一个或多个命名查询。每个查询必须驻留在单独的文本文件中。

您必须将查询文本文件放置在PXF JDBC服务器配置目录中，从该目录可以访问该查询文本文件。如果要使查询可用于多个JDBC服务器配置，则必须将查询文本文件复制到每个JDBC服务器的配置目录中。

查询文本文件必须包含要在远程SQL数据库中运行的单个查询。您必须根据数据库支持的语法来构造查询。

例如，如果一个MySQL数据库有一个 `customers` 表和一个 `orders` 表，则可以在查询文本文件中包含以下SQL语句：

```
SELECT c.name, c.city, sum(o.amount) AS total, o.month
  FROM customers c JOIN orders o ON c.id = o.customer_id
 WHERE c.state = 'CO'
 GROUP BY c.name, c.city, o.month
```

您可以选择为SQL语句提供结尾分号(`;`)。

查询命名

Greenplum数据库用户通过指定不带扩展名的查询文件名来引用命名查询。例如，如果您在名为 `report.sql` 的文件中定义查询，则该查询的名称为 `report`。

命名查询与特定的JDBC服务器配置相关联。您将向Greenplum数据库用户提供可用的查询名称，允许您使用服务器配置创建外部表。

引用命名查询

当创建外部表时，Greenplum数据库用户指定 `query:<query_name>` 而不是远程SQL数据库表的名称。例如，如果查询在文件 `$PXF_CONF/servers/mydb/report.sql` 中定义，则 `CREATE EXTERNAL TABLE` `LOCATION` 子句将包含以下组件：

```
LOCATION ('pxf://query:report?PROFILE=JDBC&SERVER=mydb ...')
```

有关使用PXF JDBC命名查询的信息，请参考[关于使用命名查询](#)。

覆盖JDBC服务器配置

您可以通过在 `CREATE EXTERNAL TABLE` 命令 `LOCATION` 子句中通过自定义选项直接指定某些 JDBC 属性来覆盖 JDBC 服务器配置。有关其他信息，请参考[通过 DDL 覆盖 JDBC 服务器配置](#)。

示例配置过程

在配置 JDBC 连接器服务器之前，请确保已初始化 PXF。

在此过程中，您将命名并添加 PostgreSQL 数据库的 PXF JDBC 服务器配置，并将服务器配置同步到 Greenplum 数据库集群。

1. 登录到您的 Greenplum 数据库主节点：

```
$ ssh gpadmin@<gpmaster>
```

2. 选择 JDBC 服务器的名称。您将名称提供给 Greenplum 用户，您可以选择这些用户允许其以配置用户身份引用外部 SQL 数据库中的表。

注意：服务器名称 `default` 已保留。

3. 创建 `$PXF_HOME/servers/<server_name>` 目录。例如，使用以下命令来创建名为 `pg_user1_testdb` 的 JDBC 服务器配置：

```
gpadmin@gpmaster$ mkdir $PXF_CONF/servers/pg_user1_testdb
```

4. 将 PXF JDBC 服务器模板文件复制到服务器配置目录。例如：

```
gpadmin@gpmaster$ cp $PXF_CONF/templates/jdbc-site.xml  
$PXF_CONF/servers/pg_user1_testdb/
```

5. 在您选择的编辑器中打开模板服务器配置文件，并为您的环境提供适当的属性值。例如，如果要在名为 `pgserverhost` 的主机上运行的 PostgreSQL 实例上为名为 `user1` 的用户配置对名为 `testdb` 的 PostgreSQL 数据库的访问：

```
<?xml version="1.0" encoding="UTF-8"?>  
<configuration>  
  <property>  
    <name>jdbc.driver</name>  
    <value>org.postgresql.Driver</value>  
  </property>  
  <property>  
    <name>jdbc.url</name>  
    <value>jdbc:postgresql://pgserverhost:5432/testdb</value>  
  </property>  
  <property>  
    <name>jdbc.user</name>  
    <value>user1</value>  
  </property>
```

```

<property>
  <name>jdbc.password</name>
  <value>changeme</value>
</property>
</configuration>

```

6. 保存更改并退出编辑器。

7. 使用 `pxf cluster sync` 命令将新的服务器配置复制到Greenplum数据库集群。例如：

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster sync
```

配置Hive访问

您可以使用PXF JDBC连接器从Hive检索数据。您还可以使用JDBC命名查询向Hive提交自定义SQL查询，并使用JDBC连接器检索结果。

本主题描述如何配置PXF JDBC连接器以访问Hive。使用JDBC配置Hive访问时，必须考虑Hive用户模拟设置，以及是否使用Kerberos保护Hadoop群集。

JDBC服务器配置

PXF JDBC连接器安装了通过JDBC访问Hive所需的JAR文件，

`hive-jdbc-<version>.jar` 和 `hive-service-<version>.jar`，并自动注册了这些JAR。

为Hive访问配置PXF JDBC服务器时，必须像配置与SQL数据库的客户端连接时一样指定JDBC驱动程序类名称，数据库URL和客户端凭据。

要通过JDBC访问Hive，您必须在 `jdbc-site.xml` 服务器配置文件中指定以下属性和值：

属性	值
<code>jdbc.driver</code>	<code>org.apache.hive.jdbc.HiveDriver</code>
<code>jdbc.url</code>	<code>jdbc:hive2://<hiveserver2_host>:<hiveserver2_port>/<database></code>

HiveServer2身份验证(`hive.server2.authentication`)和模拟(`hive.server2.enable.doAs`)属性的值以及Hive服务是否正在使用Kerberos身份验证将通知其他JDBC服务器的设置配置属性。这些属性在Hadoop集群的 `hive-site.xml` 配置文件中定义。您将需要获取这些属性的值。

下表枚举了PXF JDBC连接器支持的Hive2身份验证和模拟组合。它标识可能的Hive用户身份以及每个身份所需的JDBC服务器配置。

表标题键：

- *authentication* -> Hive `hive.server2.authentication`设置

- `enable.doAs` -> Hive `hive.server2.enable.doAs`设置
- *User Identity* -> HiveServer2将用于访问数据的身份
- *Configuration Required* -> *User Identity*需要的PXF JDBC连接器或Hive配置

authentication	enable.doAs	User Identity	Configuration Required
NOSASL	n/a	无认证	必须设置成 <code>jdbc.connection.property.auth = noSasl</code>
NONE, 或未指定	TRUE	您提供的用户名	设置 <code>jdbc.user</code>
NONE, 或未指定	TRUE	Greenplum用户名	设置 <code>pxf.impersonation.jdbc = true</code>
NONE, 或未指定	FALSE	启动Hive的用户名, 通常是 <code>hive</code>	None
KERBEROS	TRUE	PXF Kerberos主体中提供的身份, 通常是 <code>gpadmin</code>	None
KERBEROS	TRUE	您提供的用户名	设置 <code>hive.server2.proxy.user</code> 中的 <code>jdbc.url</code>
KERBEROS	TRUE	Greenplum用户名	设置 <code>pxf.impersonation.jdbc = true</code>
KERBEROS	FALSE	PXF Kerberos主体中提供的身份, 通常是 <code>gpadmin</code>	None

注意: Hive利用Kerberos身份验证时, 还需要其他配置步骤。

示例配置步骤

执行以下过程为Hive配置PXF JDBC服务器:

1. 登录到您的Greenplum数据库主节点:

```
$ ssh gpadmin@<gpmaster>
```

2. 选择JDBC服务器的名称。

3. 创建 `$PXF_HOME/servers/<server_name>` 目录。例如, 使用以下命令创建名为 `hivejdbc1` 的JDBC服务器配置:

```
gpadmin@gpmaster$ mkdir $PXF_CONF/servers/hivejdbc1
```

4. 将PXF JDBC服务器模板文件复制到服务器配置目录。例如：

```
gpadmin@gpmaster$ cp $PXF_CONF/templates/jdbc-site.xml  
$PXF_CONF/servers/hivejdbc1/
```

5. 在您选择的编辑器中打开 `jdbc-site.xml` 文件，并设置 `jdbc.driver` 和 `jdbc.url` 属性。确保指定您的Hive主机，端口和数据库名称：

```
<property>  
  <name>jdbc.driver</name>  
  <value>org.apache.hive.jdbc.HiveDriver</value>  
</property>  
<property>  
  <name>jdbc.url</name>  
  <value>jdbc:hive2://<hiveserver2_host>:<hiveserver2_port>/<database>  
</value>  
</property>
```

6. 从Hadoop群集中获取 `hive-site.xml` 文件并检查该文件。

7. 如果 `hive-site.xml` 中的 `hive.server2.authentication` 属性设置为 `NOSASL`，则HiveServer2不执行身份验证。将以下连接级别属性添加到 `jdbc-site.xml` 中：

```
<property>  
  <name>jdbc.connection.property.auth</name>  
  <value>noSasl</value>  
</property>
```

或者，您可以选择将 `;auth=noSasl` 添加到 `jdbc.url` 中。

8. 如果 `hive-site.xml` 中的 `hive.server2.authentication` 属性设置为 `NONE`，或者未指定该属性，则必须设置 `jdbc.user` 属性。设置 `jdbc.user` 属性的值取决于 `hive-site.xml` 中的 `hive.server2.enable.doAs` 模拟设置：

- 如果将 `hive.server2.enable.doAs` 设置为 `TRUE`（默认值），则Hive代表连接到Hive的用户运行Hadoop操作。选择/执行以下选项之一：设置 `jdbc.user` 以指定对Greenplum数据库访问的所有Hive数据具有读取权限的用户。例如，要连接到Hive并以 `gpadmin` 用户身份运行所有请求：

```
<property>  
  <name>jdbc.user</name>  
  <value>gpadmin</value>  
</property>
```

或，打开JDBC级用户模拟，以便PXF自动使用Greenplum数据库用户名连接到Hive：

```
<property>  
  <name>pxf.impersonation.jdbc</name>  
  <value>true</value>
```

```
</property>
```

如果以这种方式启用JDBC模拟，则既不能指定 `jdbc.user` 也不能在 `jdbc.url` 中包含设置。

2. 如果需要，创建一个PXF用户配置文件来管理密码设置。
3. 如果将 `hive.server2.enable.doAs` 设置为 `FALSE`，则Hive将以启动HiveServer2进程的用户（通常是用户 `hive`）运行Hadoop操作。在这种情况下，PXF会忽略 `jdbc.user` 设置。

9. 如果 `hive-site.xml` 中的 `hive.server2.authentication` 属性设置为 `KERBEROS`：
 1. 确保按照[为安全HDFS配置PXF](#)中的说明为PXF启用了Kerberos身份验证。
 2. 确保已将Hadoop集群配置为 `default` PXF服务器。
 3. 确保 `$PXF_CONF/servers/default/core-site.xml` 文件包含以下设置：

```
<property>
  <name>hadoop.security.authentication</name>
  <value>kerberos</value>
</property>
```

4. 在 `jdbc.url` 中添加 `saslQop` 属性，并将其设置为与 `hive-site.xml` 中的 `hive.server2.thrift.sasl.qop` 属性设置匹配。例如，如果 `hive-site.xml` 文件包含以下属性设置：

```
<property>
  <name>hive.server2.thrift.sasl.qop</name>
  <value>auth-conf</value>
</property>
```

你可以将 `;saslQop=auth-conf` 添加到 `jdbc.url`。

5. 将HiveServer2 `principal` 名称添加到 `jdbc.url` 中。例如：

```
jdbc:hive2://hs2server:10000/default;principal=hive/hs2server@REALM;sa
conf
```

6. 如果将 `hive.server2.enable.doAs` 设置为 `TRUE`（默认值），则Hive代表连接到Hive的用户运行Hadoop操作。选择/执行以下选项之一：不要指定任何其他属性。在这种情况下，PXF使用PXF Kerberos主体（通常是 `gpadmin`）中提供的身份来启动所有Hadoop访问。或，在 `jdbc.url` 中设置 `hive.server2.proxy.user` 属性，以指定对所有Hive数据具有读取权限的用户。例如，要连接到Hive并以名为 `integration` 的用户身份运行所有请求，请使用以下 `jdbc.url`：

```
jdbc:hive2://hs2server:10000/default;principal=hive/hs2server@REALM;sa
conf;hive.server2.proxy.user=integration
```

或，在 `jdbc-site.xml` 文件中启用PXF JDBC模拟，以便PXF自动使用Greenplum数据库用户名连接到Hive。例如：

```
<property>
  <name>pxf.impersonation.jdbc</name>
```

```
<value>true</value>
</property>
```

如果启用JDBC模拟，则不得在 `jdbc.url` 中显式指定
`hive.server2.proxy.user`。

7. 如果需要，创建一个PXF用户配置文件来管理密码设置。
8. 如果将 `hive.server2.enable.doAs` 设置为 `FALSE`，则Hive将使用PXF Kerberos主体提供的身份（通常为 `gpadmin`）运行Hadoop操作。
10. 保存更改并退出编辑器。
11. 使用 `pxf cluster sync` 命令将新的服务器配置复制到Greenplum数据库集群。例如：

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster sync
```

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ PXF介绍

□ PXF管理手册

□ 配置PXF

PXF安装和配置目录

为PXF安装JAVA环境

初始化PXF

配置PXF服务器

□ 配置PXF HADOOP连接器 (可选)

配置Minio和S3对象存储的连接器 (可选)

配置Azure和Google云端存储的连接器 (可选)

配置JDBC连接器 (可选)

配置PXF客户端主机和端口(可选)

升级PXF

PXF的启动、停止和重启

授权用户访问PXF

注册PXF的jar依赖

监控PXF

In this topic:

- [过程](#)

默认情况下，在segment主机上启动的PXF代理侦听localhost上的5888端口。您可以将PXF配置为从其他端口号启动，或使用其他主机名或IP地址。要更改默认配置，您将设置以下标识的一个或两个环境变量：

Environment Variable	Description
PXF_HOST	主机名或IP地址。默认主机名是 <code>localhost</code>
PXF_PORT	PXF代理用来侦听主机上的请求的端口号。默认端口号是 <code>5888</code>

在每个segment主机上的gpadmin用户的.bashrc shell登录文件中设置环境变量。

以这种方式配置代理程序主机(和/或)端口号时，必须重新启动Greenplum数据库和PXF。考虑在计划的停机时间内执行此配置。

过程

执行以下过程以在一台或多台Greenplum数据库主机上配置PXF代理主机(和/或)端口号：

1. 登录到您的Greenplum数据库主节点

```
$ ssh gpadmin@<gpmaster>
```

2. 对于每个Greenplum数据库segment主机

1. 指定PXF代理的主机名或IP地址。
2. 指定要在其上运行PXF代理的端口号。
3. 登录到Greenplum segment主机：

```
$ ssh gpadmin@<segghost>
```

4. 使用编辑器中打开 `~/.bashrc` 文件

5. 设置 `PXF_HOST` (和/或) `PXF_PORT` 环境变量。例如，要将PXF代理端口号设置为5998，请将以下内容添加到 `.bashrc` 文件中

```
export PXF_PORT=5998
```

6. 保存修改并退出。
3. 重启数据库。参阅[Restarting Greenplum Database](#)。
4. 重启每个segment上部署的PXF,参阅[Restarting PXF](#)。

[Greenplum数据库® 6.0文档](#)

[Greenplum平台扩展框架\(PXF\)](#)

[PXF介绍](#)

[PXF管理手册](#)

[配置PXF](#)

升级PXF

[PXF的启动、停止和重启](#)

[授权用户访问PXF](#)

[注册PXF的jar依赖](#)

[监控PXF](#)

[使用PXF访问hadoop](#)

[使用PXF访问Azure, Google云端存储, Minio和S3对象存储](#)

[使用PXF访问SQL数据库\(JDBC\)](#)

[PXF故障排除](#)

[PXF实用程序手册](#)

[Back to the Administrator Guide](#)

[Greenplum database 5管理员指南](#)

[Greenplum database 4管理员指南](#)

[FAQ](#)

[如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息](#)

In this topic:

- [Step 1: PXF Pre-Upgrade Actions](#)
- [Step 2: Upgrading PXF](#)

如果在当前的Greenplum数据库中安装使用PXF，则必须在升级到新版本Greenplum数据库时升级PXF服务。

PXF升级过程描述了如何在Greenplum数据库安装中升级PXF。此过程使用*PXF.from*来表示当前安装的PXF版本, *PXF.to*表示升级新版本的Greenplum数据库时安装的PXF版本。

大多数PXF安装不需要修改PXF配置文件，并且可以进行无缝升级

Note: 从Greenplum Database版本5.12开始，PXF不再需要安装Hadoop客户端。PXF现在捆绑了它所依赖的所有JAR文件，并在运行时加载这些JAR。

PXF升级过程分为两部分。您在升级到Greenplum数据库的新版本之前执行一个过程，之后执行一个过程：

- [Step 1: PXF Pre-Upgrade Actions](#)
- GPDB升级到一个新的版本
- [Step 2: Upgrading PXF](#) GPDB升级到一个新的版本

Step 1: PXF Pre-Upgrade Actions

在升级到Greenplum数据库的新版本之前执行此过程：

1. 登录到GPDB master节点

```
$ ssh gpadmin@<gpmaster>
```

2. 安装[Stopping PXF](#) 章节描述停止每个segment节点上的PXF

3. 如果你想从GPDB版本**5.14**或更早的版本开始升级：

1. 备份 *PXF.from* 在 `$GPHOME/pxf/conf/` 目录的配置文件。所有segment主机上的这些文件应该相同，因此您只需要从其中一个主机进行复制。例如：

```
gpadmin@gpmaster$ mkdir -p /save/pxf-from-conf
gpadmin@gpmaster$ scp
gpadmin@segghost1:/usr/local/greenplum-db/pxf/conf/*
/save/pxf-from-conf/
```

2. 请注意您可能已添加到 *PXF.from* 安装的任何自定义 JAR 文件的位置。保存这些 JAR 文件的副本
4. 升级到新版本的 Greenplum 数据库，然后参照 [Step 2: Upgrading PXF](#)。

Step 2: Upgrading PXF

升级到新版本的 Greenplum 数据库后，请执行以下步骤以升级和配置 *PXF.to* 软件：

1. 登录到 GPDB master 节点

```
$ ssh gpadmin@<gpmaster>
```

2. 按照 [Initializing PXF](#) 的描述在每个 segment 主机上初始化 PXF。
3. 默认情况下，在 Greenplum 数据库版本 5.5.0 及更高版本中启用 PXF 用户模拟。如果要从较旧的 *PXF.from* 版本升级，则必须为基础 Hadoop 服务配置用户模拟。有关说明，请参阅 [Configuring User Impersonation and Proxying](#)，包括关闭 PXF 用户模拟的配置过程。
4. 如果你想从 GPDB 版本 **5.14** 或更早的版本开始升级：

1. 如果更新了 *PXF.from* 安装中的 `pxf-env.sh` 配置文件，请将这些更改重新应用于 `$PXF_CONF/conf/pxf-env.sh`。例如：

```
gpadmin@gpmaster$ vi $PXF_CONF/conf/pxf-env.sh
<update the file>
```

2. 同样，如果您在 *PXF.from* 安装中更新了 `pxf-profiles.xml` 配置文件，请将这些更改重新应用到主机上的 `$PXF_CONF/conf/pxf-profiles.xml`。
- Note:** 从 Greenplum Database 版本 5.12 开始，PXF 类的包名称已更改为使用前缀 `org.greenplum.*`。如果要从较旧的 *PXF.from* 版本升级并自定义 `pxf-profiles.xml` 文件，则必须在重新应用时更改对 `org.greenplum.pxf.*` 的任何

`org.apache.hawq.pxf.*` 的引用。

3. 如果更新了 `PXF.from` 安装中的 `pxf-log4j.properties` 配置文件，请将这些更改重新应用到主机上的 `$PXF_CONF/conf/pxf-log4j.properties`
4. 如果在 `PXF.from` 安装中更新了 `pxf-public.classpath` 配置文件，请将文件中引用的每个JAR复制到naster主机的 `$PXF_CONF/lib`
5. 如果你将其他JAR文件添加到 `PXF.from` 中，请将它们复制到master主机上的 `$PXF_CONF/lib`
6. 从Greenplum Database版本 5.15 开始，PXF需要Hadoop配置文件放置在 `$PXF_CONF/servers/default` 目录中。如果在 `PXF.from` 安装中配置了PXF Hadoop连接器，请将 `/etc/<hadoop_service>/conf` 中的Hadoop配置文件复制到Greenplum Database主机上的 `$PXF_CONF/servers/default`
7. 从Greenplum Database版本 5.15 开始，PXF的默认Kerberos keytab文件位置是 `$PXF_CONF/keytabs`。如果先前已将PXF配置为安全HDFS 且PXF密钥表文件位于 `PXF.from` 安装目录中(例如, `$GPHOME/pxf/conf`)，请考虑将keytab文件重定位到 `$PXF_CONF/keytabs` ,或者, 更新 `$PXF_CONF/conf/pxf-env.sh` 文件中的 `pxf_keytab` 属性设置以引用您的keytab文件。

5. 如果要从Greenplum数据库5.18或更早版本升级：

1. 现在，PXF捆绑了Hadoop 2.9.2版相关的JAR文件。如果您在 `$PXF_CONF/lib` 中注册了其他与Hadoop相关的JAR文件，请确保这些库与Hadoop 2.9.2版兼容。
2. 现在，PXF JDBC连接器支持基于文件的服务器配置。如果选择将此新功能与引用外部SQL数据库的现有外部表一起使用，请参阅[配置JDBC连接器](#)中的配置说明，以获取更多信息。
3. 现在，PXF JDBC连接器支持语句查询超时。此特性需要JDBC驱动程序的显式支持。默认查询超时为 `0`，请耐心等待。一些JDBC驱动程序在不完全支持语句查询超时功能的情况下支持 `0` 超时值。确保已注册的所有JDBC驱动程序都支持默认超时，或者更好的是，它完全支持此特性。您可能需要更新JDBC驱动程序版本以获得此支持。有关使用PXF注册JAR文件的信息，请参考[JDBC驱动程序JAR注册](#)中的配置说明。
4. 如果您打算对整数类型使用Hive分区过滤，则必须在Hadoop群集的 `hive-site.xml` 和PXF用户配置 `$PXF_CONF/servers/default/hive-site.xml` 中设置 `hive.metastore.integral.jdo.pushdown` Hive属性。请参阅[更新Hadoop配置](#)。

6. 如果您要从Greenplum数据库5.21.1或更早版本升级：当您创建指

定 `HiveText` 或 `HiveRC` 配置文件的外部表时，PXF Hive连接器不再支持在 `LOCATION` URI中提供 `DELIMITER=<delim>` 选项。如果您以前创建了一个在 `LOCATION` URI中指定 `DELIMITER` 的外部表，则必须删除该表，然后从 `LOCATION` 省略 `DELIMITER` 来重新创建它。您仍然需要在外部表格式设置选项中提供非默认定界符。

7. 将PXF配置从master主机同步到standby和每个Greenplum数据库segment主机。例如：

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster sync
```

8. 按照[Starting PXF](#) 章节描述在每个segment主机上启动PXF。

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ PXF介绍

□ PXF管理手册

□ 配置PXF

升级PXF

PXF的启动、停止和重启

授权用户访问PXF

注册PXF的jar依赖

监控PXF

□ 使用PXF访问hadoop

□ 使用PXF访问Azure, Google云端存储, Minio和S3对象存储

使用PXF访问SQL数据库
(JDBC)

PXF故障排除

□ PXF实用程序手册

Back to the Administrator Guide

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

In this topic:

- [启动PXF](#)
- [停止PXF](#)
- [重启PXF](#)

PXF提供了两个管理命令：

- `pxf cluster` - 管理Greenplum数据库集群中的所有PXF服务实例
- `pxf` - 在特定的Greenplum数据库主机上管理PXF服务实例

`pxf cluster` 支持 `init`, `start`, `status`, `stop`, and `sync` 子命令。在Greenplum数据库master主机上运行 `pxf cluster` 子命令时, 将在Greenplum数据库集群中的所有segment主机上执行该操作。PXF还在standby主机上运行 `init` 和 `sync` 命令。

`pxf` 支持 `init`, `start`, `stop`, `restart`, `status` 操作。这些操作在本地执行, 也就是说, 如果要在特定的Greenplum数据库segment主机上启动或停止PXF代理, 需要登录到该主机并运行命令。

启动PXF

初始化PXF之后, 必须在Greenplum数据库集群中的每个segment主机上启动PXF。PXF服务启动后, 将以 `gpadmin` 用户身份在默认端口5888上运行。只有 `gpadmin` 用户可以启动和停止pxf服务。

如果要更改默认的PXF配置, 则必须在启动PXF之前更新配置。

`$PXF_CONF/conf` 包含用户自定义的配置文件:

- `pxf-env.sh` - 运行时配置参数
- `pxf-log4j.properties` - 日志记录配置参数
- `pxf-profiles.xml` - 自定义配置文件定义

`pxf-env.sh` 包含以下用户可自定义的配置文件:

Parameter	Description	Default Value
JAVA_HOME	Java JRE家目录	/usr/java/default
PXF_LOGDIR	PXF日志目录	\$PXF_CONF/logs

PXF_JVM_OPTS	PXF Java虚拟机的默认选项	-Xmx2g -Xms1g
PXF_KEYTAB	PXF服务Kerberos主体密钥表文件的绝对路径	\$PXF_CONF/keytabs/pxf.service.keytab
PXF_PRINCIPAL	PXF服务Kerberos主体	gpadmin/_ HOST@EXAMPLE.COM

您必须将对 `pxf-env.sh` , `pxf-log4j.properties` 或 `pxf-profiles.xml` 所做的所有变更同步到Greenplum数据库集群，并在每个segment节点上(重新)启动PXF。

准备

在Greenplum数据库集群中启动PXF之前，请确保：

- Greenplum数据库集群已启动并正在运行
- PXF已经被初始化

过程

执行以下过程以在Greenplum数据库集群中的每个segment主机上启动PXF。

1. 登录greenplum master节点

```
$ ssh gpadmin@<gpmaster>
```

2. 在每个segment主机上运行 `pxf cluster start` 命令启动pxf服务

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster start
```

停止PXF

如果必须停止PXF,例如如果要升级PXF，则必须在Greenplum数据库集群中的每个segment主机上停止PXF。只有 `gpadmin` 用户可以停止PXF服务

准备

在Greenplum数据库集群中停止PXF之前，请确保Greenplum数据库集群已启动并正在运行。

过程

执行以下过程在Greenplum数据库集群中的每个segment主机上停止PXF。

1. 登录greenplum master节点

```
$ ssh gpadmin@<gpmaster>
```

2. 在每个segment主机上运行 `pxf cluster stop` 命令停止pxf服务。例如:

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster stop
```

重启PXF

如果必须重新启动PXF，例如在 `$PXF_CONF/conf` 中更新了PXF用户配置文件，则可以在Greenplum数据库集群中先停止服务然后再启动PXF服务。

只有 `gpadmin` 用户可以重启PXF服务。

准备

在Greenplum数据库集群中重新启动PXF之前，请确保Greenplum数据库集群已启动并正在运行。

过程

执行以下过程在Greenplum数据库集群中的每个segment上重启PXF。

1. 登录greenplum master节点:

```
$ ssh gpadmin@<gpmaster>
```

2. 重启PXF:

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster stop
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster start
```


[Greenplum数据库® 6.0文档](#)[Greenplum平台扩展框架\(PXF\)](#)[PXF介绍](#)[PXF管理手册](#)[配置PXF](#)[升级PXF](#)[PXF的启动、停止和重启](#)

授权用户访问PXF

[注册PXF的jar依赖](#)[监控PXF](#)[使用PXF访问hadoop](#)[使用PXF访问Azure, Google云端存储, Minio和S3对象存储](#)[使用PXF访问SQL数据库\(JDBC\)](#)[PXF故障排除](#)[PXF实用程序手册](#)[Back to the Administrator Guide](#)[Greenplum database 5管理员指南](#)[Greenplum database 4管理员指南](#)[FAQ](#)

[如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息](#)

In this topic:

- [在数据库中启用PXF](#)
- [在数据库中禁用PXF](#)
- [授权一个用户访问PXF](#)

Greenplum平台扩展框架(PXF)实现了一个名为 `pxf` 的协议，您可以使
用该协议创建引用外部数据存储中数据的外部表。PXF协议和Java服
务打包作为Greenplum数据库扩展。

您必须在将要使用框架访问外部数据的每个数据库实例中启用PXF扩
展。您还必须向需要访问权限的用户/角色明确授予 `pxf` 协议的
`GRANT` 权限。

在数据库中启用PXF

您必须在计划使用该扩展的每个Greenplum数据库中显式注册PXF扩
展。您必须具有Greenplum数据库管理员权限才能注册扩展。

对要在其中使用PXF的每个数据库执行以下过程：

1. 使用 `gpadmin` 用户连接数据库

```
gpadmin@gpmaster$ psql -d <database-name> -U gpadmin
```

2. 创建PXF扩展名。您必须具有Greenplum数据库管理员权限才能创
建扩展。例如：

```
database-name=# CREATE EXTENSION pxf;
```

创建 `pxf` 扩展来注册 `pxf` 协议和PXF访问外部数据所需的调用处
理程序。

在数据库中禁用PXF

当您不再希望在特定数据库上使用PXF时，必须显式删除该数据库
的PXF扩展名。您必须具有Greenplum数据库管理员权限才能删除扩

展。

1. 使用 `gpadmin` 用户连接数据库

```
gpadmin@gpmaster$ psql -d <database-name> -U gpadmin
```

2. 删除pxf扩展

```
database-name=# DROP EXTENSION pxf;
```

如果当前使用 `pxf` 协议定义了任何外部表，则 `DROP` 命令将失败。如果选择强制删除这些外部表，请添加 `CASCADE` 选项。

授权一个用户访问PXF

要使用PXF读取外部数据，请使用 `CREATE EXTERNAL TABLE` 命令创建一个外部表，该命令指定 `pxf` 协议。您必须向所有需要此类访问权限的非 `SUPERUSER` 的Greenplum数据库角色明确授予对pxf协议的 `SELECT` 权限。

要授予特定角色对 `pxf` 协议的访问权限，请使用 `GRANT` 命令。例如，要授予名为 `bill` 的角色对使用 `pxf` 协议创建的外部表引用的数据的读取访问权限，请执行以下操作：

```
GRANT SELECT ON PROTOCOL pxf TO bill;
```

要使用PXF将数据写入外部数据存储，请使用 `CREATE WRITABLE EXTERNAL TABLE` 命令创建一个外部表，该命令指定 `pxf` 协议。您必须向需要此类访问的所有非 `SUPERUSER` 的Greenplum数据库角色明确授予对 `pxf` 协议的 `INSERT` 权限。例如：

```
GRANT INSERT ON PROTOCOL pxf TO bill;
```

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

- PXF介绍

- PXF管理手册

- 配置PXF

- 升级PXF

- PXF的启动、停止和重启

- 授权用户访问PXF

- 注册PXF的jar依赖**

- 监控PXF

- 使用PXF访问hadoop

- 使用PXF访问Azure, Google云端存储, Minio和S3对象存储

使用PXF访问SQL数据库
(JDBC)

PXF故障排除

- PXF实用程序手册

Back to the Administrator Guide

Greenplum database 5管理员指南

Greenplum database 4管理员指南

FAQ

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

您使用PXF访问外部系统上存储的数据。根据外部数据的存储，此访问可能需要您安装和/或配置外部数据存储的其他组件或服务。

PXF取决于这些附加组件提供的JAR文件和其他配置信息。

`$GPHOME/pxf/conf/pxf-private.classpath` 文件标识PXF内部JAR依赖性。在大多数情况下，PXF将管理 `pxf-private.classpath` 文件，并根据您使用的连接器根据需要添加条目。

如果您需要为PXF添加其他JAR依赖关系，例如JDBC驱动程序JAR文件，则必须登录到Greenplum数据库master主机，将JAR文件复制到PXF用户配置运行时库目录(`$PXF_CONF/lib`)，将PXF配置同步到Greenplum数据库集群，然后在每个主机上重新启动PXF。例如：

```
$ ssh gpadmin@<gpmaster>
gpadmin@gpmaster$ cp new_dependent_jar.jar $PXF_CONF/lib/
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster sync
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster stop
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster start
```

[Greenplum数据库® 6.0文档](#)[Greenplum平台扩展框架\(PXF\)](#)[PXF介绍](#)[PXF管理手册](#)[配置PXF](#)[升级PXF](#)[PXF的启动、停止和重启](#)[授权用户访问PXF](#)[注册PXF的jar依赖](#)[监控PXF](#)[使用PXF访问hadoop](#)[使用PXF访问Azure, Google云端存储, Minio和S3对象存储](#)[使用PXF访问SQL数据库\(JDBC\)](#)[PXF故障排除](#)[PXF实用程序手册](#)[Back to the Administrator Guide](#)[Greenplum database 5管理员指南](#)[Greenplum database 4管理员指南](#)[FAQ](#)

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

要显示PXF状态，必须在Greenplum数据库集群中的每个segment主机上显式请求PXF服务实例的状态。`pxf cluster status` 命令可显示Greenplum数据库集群中所有segment主机上PXF服务实例的状态。`pxf status` 显示本地(segment)主机上PXF服务实例的状态。

只有 `gpadmin` 用户可以请求PXF服务的状态。

执行以下步骤哦，以请求Greenplum数据库集群的PXF状态。

1. 登录gpdb集群的master主机

```
$ ssh gpadmin@<gpmaster>
```

2. 运行 `pxf cluster status` 命令：

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster status
```

Greenplum数据库® 6.0文档

Greenplum平台扩展框
架(PXF)

□ PXF介绍

□ PXF管理手册

□ 使用PXF访问hadoop

读写文本数据

读取Avro数据

读取JSON数据

读写Parquet数据

读写SequenceFile数据

将多行文本文件读入单个
表行

读取Hive表数据

读取HBase表数据

□ 使用PXF访
问Azure, Google云端存
储, Minio和S3对象存储使用PXF访问SQL数据库
(JDBC)

PXF故障排除

□ PXF实用程序手册

Back to the Administrator
GuideGreenplum database 5管理员
指南

In this topic:

- [先决条件](#)
- [读取文本数据](#)
 - [示例：读取HDFS中的文本数据](#)
- [读取含有双引号引起来的换行符的文本数据](#)
 - [示例：在HDFS上读取多行文本数据](#)
- [将文本文件写入HDFS](#)
 - [示例：将文本数据写入HDFS](#)

PXF HDFS连接器支持纯分隔和逗号分隔的值表单文本数据。本节介绍如何使用PXF访问HDFS文本数据，包括如何创建引用了HDFS文件的外部表，查询外部表和向外部表写入数据。

先决条件

在尝试从HDFS读取或向HDFS写入数据之前，请确保已满足PXF Hadoop[先决条件](#)。

读取文本数据

当您在读取每行都是单条记录的纯文本分隔或csv数据时，请使用 `hdfs:text` 配置文件。以下语法创建了一个Greenplum数据库可读外部表，该表引用了HDFS上的此类文本文件：

```
CREATE EXTERNAL TABLE <table_name>
  ( <column_name> <data_type> [, ...] | LIKE <other_table> )
LOCATION ('pxf://<path-to-hdfs-file>?PROFILE=hdfs:text[&SERVER=<server_name>]')
FORMAT '[TEXT|CSV]' (delimiter[=|<space>][E]<delim_value>');
```

`CREATE EXTERNAL TABLE` 命令中使用的特定关键字和值见下表中描述。

关键字	值
<path-to-hdfs-file>	HDFS数据存储中目录或文件的绝对路径
PROFILE	<code>PROFILE</code> 关键字必须指定为 <code>hdfs:text</code>
SERVER=<server_name>	PXF用于访问数据的命名服务器配置。可选的；如果未指定，PXF将使用 <code>default</code> 服务器。
FORMAT	当<path-to-hdfs-file>引用纯文本分隔数据时，请使用 <code>FORMAT 'TEXT'</code> 。 当<path-to-hdfs-file>引用逗号分隔值数据时，请使用

	<code>FORMAT 'CSV'。</code>
delimiter	数据中的分隔符。对于 <code>FORMAT 'CSV'</code> ，默认的 <code><delim_value></code> 是逗号 <code>,</code> 。当分隔符为转义序列时，在 <code><delim_value></code> 前面加上 <code>E</code> 。示例： <code>(delimiter=E'\t')</code> , <code>(delimiter ':')</code> 。

示例：读取HDFS中的文本数据

执行以下过程来创建示例文本文件，将文件复制到HDFS，然后使用 `hdfs:text` 配置文件和默认的PXF服务器创建两个PXF外部表来查询数据：

1. 为PXF示例数据文件创建HDFS目录。例如：

```
$ hdfs dfs -mkdir -p /data/pxf_examples
```

2. 创建名为 `pxf_hdfs_simple.txt` 的纯文本分割数据文件：

```
$ echo 'Prague,Jan,101,4875.33
Rome,Mar,87,1557.39
Bangalore,May,317,8936.99
Beijing,Jul,411,11600.67' > /tmp/pxf_hdfs_simple.txt
```

注意使用逗号`,`来分隔四个数据字段。.

3. 将数据文件添加到HDFS：

```
$ hdfs dfs -put /tmp/pxf_hdfs_simple.txt /data/pxf_examples/
```

4. 显示存储在HDFS中的 `pxf_hdfs_simple.txt` 文件的内容：

```
$ hdfs dfs -cat /data/pxf_examples/pxf_hdfs_simple.txt
```

5. 启动 `psql` 子系统：

```
$ psql -d postgres
```

6. 使用PXF `hdfs:text` 配置文件创建一个引用刚刚创建并添加到HDFS的 `pxf_hdfs_simple.txt` 文件的Greenplum数据库外部表：

```
postgres=# CREATE EXTERNAL TABLE pxf_hdfs_textsimple(location text,
month text, num_orders int, total_sales float8)
LOCATION ('pxf://data/pxf_examples/pxf_hdfs_simple.txt?
PROFILE=hdfs:text')
FORMAT 'TEXT' (delimiter=','');
```

7. 查询外部表:

```
postgres=# SELECT * FROM pxf_hdfs_textsimple;
```

location	month	num_orders	total_sales
Prague	Jan	101	4875.33
Rome	Mar	87	1557.39
Bangalore	May	317	8936.99
Beijing	Jul	411	11600.67
(4 rows)			

8. 创建第二个引用 `pxf_hdfs_simple.txt` 的外部表，这一次指定 `FORMAT` 为 `CSV`：

```
postgres=# CREATE EXTERNAL TABLE pxf_hdfs_textsimple_csv(location text,
month text, num_orders int, total_sales float8)
LOCATION ('pxf://data/pxf_examples/pxf_hdfs_simple.txt?
PROFILE=hdfs:text')
FORMAT 'CSV';
postgres=# SELECT * FROM pxf_hdfs_textsimple_csv;
```

当您为逗号分隔值数据指定 `FORMAT 'CSV'` 时，不需要提供 `delimiter` 分隔符选项，因为默认的分隔符是逗号。

读取含有双引号引起的换行符的文本数据

使用 `hdfs:text:multi` 配置文件读取数据中含有引号引起的换行符，单行或多行的分隔文本数据。以下语句创建一个引用此类文件的Greenplum外部表：

```
CREATE EXTERNAL TABLE <table_name>
( <column_name> <data_type> [, ...] | LIKE <other_table> )
LOCATION ('pxf://<path-to-hdfs-file>?PROFILE=hdfs:text:multi[&SERVER=
<server_name>]')
FORMAT '[TEXT|CSV]' (delimiter[=|<space>][E]<delim_value>');
```

`CREATE EXTERNAL TABLE` 命令中使用的特定关键字和值见下表中描述。

关键字	值
<path-to-hdfs-file>	HDFS数据存储中目录或文件的绝对路径
PROFILE	<code>PROFILE</code> 关键字必须指定为 <code>hdfs:text:multi</code>
SERVER=<server_name>	PXF用于访问数据的命名服务器配置。可选的；如果未指定，PXF将使用 <code>default</code> 服务器。
	当<path-to-hdfs-file>引用纯文本分隔数据时，请使用 <code>FORMAT 'TEXT'</code> 。

FORMAT	当<path-to-hdfs-file>引用逗号分隔值数据时, 请使用 <code>FORMAT 'CSV'</code> 。
delimiter	数据中的分隔符。对于 <code>FORMAT 'CSV'</code> , 默认的<delim_value>是逗号 <code>,</code> 。当分隔符为转义序列时, 在<delim_value>前面加上 <code>E</code> 。示例: <code>(delimiter=E'\t')</code> , <code>(delimiter ':')</code> 。

示例：在HDFS上读取多行文本数据

执行以下步骤来创建示例文本文件, 将文件复制到HDFS, 然后使用PXF `hdfs: text: multi` 配置文件和默认的PXF服务器创建Greenplum数据库可读的外部表来查询数据:

1. 创建第二个分隔的纯文本文件:

```
$ vi /tmp/pxf_hdfs_multi.txt
```

2. 将以下数据复制/黏贴到`pxf_hdfs_multi.txt`中:

```
"4627 Star Rd.  
San Francisco, CA 94107":Sept:2017  
"113 Moon St.  
San Diego, CA 92093":Jan:2018  
"51 Belt Ct.  
Denver, CO 90123":Dec:2016  
"93114 Radial Rd.  
Chicago, IL 60605":Jul:2017  
"7301 Brookview Ave.  
Columbus, OH 43213":Dec:2018
```

注意使用冒号`:`分隔三个字段。另外请注意第一个(地址)字段周围的引号。这个字段包含了一个嵌入的换行符, 用于将街道地址与城市和州分开。

3. 将文本文件复制到HDFS:

```
$ hdfs dfs -put /tmp/pxf_hdfs_multi.txt /data/pxf_examples/
```

4. 使用`hdfs:text:multi`配置文件创建一个引用HDFS文件`pxf_hdfs_multi.txt`的外部表, 确保将`:`(冒号)标识为字段分隔符:

```
postgres=# CREATE EXTERNAL TABLE pxf_hdfs_textmulti(address text, month
text, year int)
  LOCATION ('pxf://data/pxf_examples/pxf_hdfs_multi.txt?
PROFILE=hdfs:text:multi')
  FORMAT 'CSV' (delimiter ':');
```

Notice the alternate syntax for specifying the `delimiter`.

5. 查询 `pxf_hdfs_textmulti` 表:

```
postgres=# SELECT * FROM pxf_hdfs_textmulti;
```

address	month	year
4627 Star Rd. San Francisco, CA 94107	Sept	2017
113 Moon St. San Diego, CA 92093	Jan	2018
51 Belt Ct. Denver, CO 90123	Dec	2016
93114 Radial Rd. Chicago, IL 60605	Jul	2017
7301 Brookview Ave. Columbus, OH 43213	Dec	2018

(5 rows)

将文本文件写入HDFS

PXF HDFS连接器“`hdfs:text`”配置文件支持将单行纯文本数据写入HDFS。当您使用PXF HDFS连接器创建可写外部表时，可以指定在HDFS上的目录名称。当您向可写外部表写入数据时，您写入的数据块将写入到指定目录中的一个或多个文件。

注意: 使用可写配置文件创建的外部表只能用于 `INSERT` 操作。如果要查询写入的数据，则必须另外创建一个引用HDFS目录的可读外部表。

使用以下语法创建一个引用HDFS目录的Greenplum可写外部表：

```
CREATE WRITABLE EXTERNAL TABLE <table_name>
  ( <column_name> <data_type> [, ...] | LIKE <other_table> )
LOCATION ('pxf://<path-to-hdfs-dir>
?PROFILE=hdfs:text[&SERVER=<server_name>][&<custom-option>=<value>
[...]]')
FORMAT '[TEXT|CSV]' (<delimiter>[=|<space>][E]<delim_value>');
[DISTRIBUTED BY (<column_name> [, ... ] ) | DISTRIBUTED RANDOMLY];
```

`CREATE EXTERNAL TABLE`命令中使用的特定关键字和值见下表中描述。

关键字	值
<path-to-hdfs-dir>	HDFS数据存储中目录的绝对路径
PROFILE	<code>PROFILE</code> 关键字必须指定为 <code>hdfs:text</code>
SERVER=<server_name>	PXF用于访问数据的命名服务器配置。可选的；如果未指定，PXF将使用 <code>default</code> 服务器。
<custom-option>	<custom-option>描述见下方
	当<path-to-hdfs-file>引用纯文本分隔数据时，请使用 <code>TEXT</code> 或 <code>CSV</code>

FORMAT	<code>FORMAT 'TEXT'</code> 。 当<path-to-hdfs-file>引用逗号分隔值数据时,请使用 <code>FORMAT 'CSV'</code> 。
delimiter	数据中的分隔符。对于 <code>FORMAT 'CSV'</code> ,默认的<delim_value>是逗号 <code>,</code> 。当分隔符为转义序列时,在<delim_value>前面加上 <code>E</code> 。示例: <code>(delimiter=E'\t')</code> , <code>(delimiter ':')</code> 。
DISTRIBUTED BY	如果您计划将现有Greenplum数据库表中的数据加载到可写外部表,考虑在可写外部表上使用Greenplum表相同的分布策略或<字段名>。这样做可以避免数据加载操作中segment节点间额外的数据移动。

使用`hdfs:text`配置文件创建的可写外部表可以选择使用记录或块压缩。PXF`hdfs:text`配置文件支持以下压缩编解码器:

- org.apache.hadoop.io.compress.DefaultCodec
- org.apache.hadoop.io.compress.GzipCodec
- org.apache.hadoop.io.compress.BZip2Codec

您可以通过`CREATE EXTERNAL TABLE`子句中自定义选择指定压缩编解码器。`hdfs:text`配置文件支持以下自定义写入选项:

选项	值描述
COMPRESSION_CODEC	压缩编解码器Java类名。如果未提供此选项,Greenplum数据库不会执行压缩编码。支持的压缩编解码器包括: <code>org.apache.hadoop.io.compress.DefaultCodec</code> <code>org.apache.hadoop.io.compress.BZip2Codec</code> <code>org.apache.hadoop.io.compress.GzipCodec</code>
COMPRESSION_TYPE	采用的压缩类型;支持的值为 <code>RECORD</code> (默认)或 <code>BLOCK</code>
THREAD-SAFE	确定表查询是否可以在多线程模式下运行的布尔值。默认为 <code>TRUE</code> 。将此选项设置为 <code>FALSE</code> 以处理单个线程中所有非线程安全操作(例如,压缩)的请求。

示例：将文本数据写入HDFS

此示例使用了[示例：读取HDFS中的文本数据](#)中的数据格式。

列名	数据类型
location	text

month	text
number_of_orders	int
total_sales	float8

在此示例中您还可以选择在该练习创建的 `pxf_hdfs_textsimple` Greenplum数据库外部表。

步骤

执行以下步骤，使用与上述相同的数据模式创建Greenplum数据库可写外部表，其中一个表将使用压缩。您将使用PXF `hdfs: text` 配置文件和默认的PXF服务器将数据写入基础HDFS目录。您还将创建一个单独的可读外部表，以读取您写入HDFS目录的数据。

1. 使用上述数据格式创建Greenplum数据库可写外部表。写入HDFS目录

`/data/pxf_examples/pxfwritable_hdfs_textsimple1`。使用逗号 `,` 作为分隔符创建表：

```
postgres=# CREATE WRITABLE EXTERNAL TABLE
pxf_hdfs_writetabletbl_1(location text, month text, num_orders int,
total_sales float8)
LOCATION
('pxf://data/pxf_examples/pxfwritable_hdfs_textsimple1?
PROFILE=hdfs:text')
FORMAT 'TEXT' (delimiter=',');
```

您将 `FORMAT` 子句 `delimiter` 的值指定为单个ascii逗号字符 `,`。

2. 通过在 `pxf_hdfs_writetabletbl_1` 上调用SQL `INSERT` 命令，将一些单独的记录写入 `pxfwritable_hdfs_textsimple1` HDFS目录：

```
postgres=# INSERT INTO pxf_hdfs_writetabletbl_1 VALUES ( 'Frankfurt',
'Mar', 777, 3956.98 );
postgres=# INSERT INTO pxf_hdfs_writetabletbl_1 VALUES ( 'Cleveland',
'Oct', 3812, 96645.37 );
```

3. (可选) 写入您在[示例：读取HDFS中的文本数据](#)创建的

`pxf_hdfs_textsimple` 表中的数据到 `pxf_hdfs_writetabletbl_1`：

```
postgres=# INSERT INTO pxf_hdfs_writetabletbl_1 SELECT * FROM
pxf_hdfs_textsimple;
```

4. 在另一个终端窗口，显示刚添加到HDFS的数据：

```
$ hdfs dfs -cat /data/pxf_examples/pxfwritable_hdfs_textsimple1/*
Frankfurt,Mar,777,3956.98
Cleveland,Oct,3812,96645.37
Prague,Jan,101,4875.33
Rome,Mar,87,1557.39
```

```
Bangalore,May,317,8936.99
Beijing,Jul,411,11600.67
```

因为您在创建可写外部表时使用了逗号 `,` 作为分隔符, 这个字符是HDFS数据中每条记录的字段分隔符.

5. Greenplum数据库不支持直接查询可写外部表。要查询刚刚添加到HDFS的数据, 你必须创建一个引用这个HDFS目录的Greenplum可读外部表:

```
postgres=# CREATE EXTERNAL TABLE pxf_hdfs_textsimple_r1(location text,
month text, num_orders int, total_sales float8)
LOCATION
('pxf://data/pxf_examples/pxfwritable_hdfs_textsimple1?
PROFILE=hdfs:text')
FORMAT 'CSV';
```

在创建可读外部表时使用 `'CSV'` `FORMAT`, 因为您在创建可写外部表时使用逗号 `,` 作为分隔符, 即 `'CSV'` `FORMAT` 的默认分隔符。

6. 查询可读外部表:

```
postgres=# SELECT * FROM pxf_hdfs_textsimple_r1 ORDER BY total_sales;
```

location	month	num_orders	total_sales
Rome	Mar	87	1557.39
Frankfurt	Mar	777	3956.98
Prague	Jan	101	4875.33
Bangalore	May	317	8936.99
Beijing	Jul	411	11600.67
Cleveland	Oct	3812	96645.37
(6 rows)			

`pxf_hdfs_textsimple_r1` 表包含您单独插入的记录, 以及执行可选步骤时 `pxf_hdfs_textsimple` 表的完整内容。

7. 创建第二个Greenplum数据库可写外部表, 这次使用Gzip压缩并使用冒号 `:` 作为分隔符:

```
postgres=# CREATE WRITABLE EXTERNAL TABLE pxf_hdfs_writetabletbl_2 (location
text, month text, num_orders int, total_sales float8)
LOCATION ('pxf://data/pxf_examples/pxfwritable_hdfs_textsimple'
PROFILE=hdfs:text&COMPRESSION_CODEC=org.apache.hadoop.io.compress.GzipCode
FORMAT 'TEXT' (delimiter=':');
```

8. 通过直接写入 `pxf_hdfs_writetabletbl_2` 表, 将一些记录写入 `pxfwritable_hdfs_textsimple2` HDFS目录:

```
gpadmin=# INSERT INTO pxf_hdfs_writetabletbl_2 VALUES ( 'Frankfurt',
'Mar', 777, 3956.98 );
gpadmin=# INSERT INTO pxf_hdfs_writetabletbl_2 VALUES ( 'Cleveland',
```

```
'Oct', 3812, 96645.37 );
```

9. 在另一个终端窗口中，显示您刚添加到HDFS的数据；使用 `hdfs dfs -text` 选项以文本的形式查看压缩数据：

```
$ hdfs dfs -text /data/pxf_examples/pxfwritable_hdfs_textsimple2/*
Frankfurt:Mar:777:3956.98
Cleveland:Oct:3812:96645.3
```

请注意冒号 `:` 是此HDFS数据的字段分隔符。

要从名为 `pxfwritable_hdfs_textsimple2` 的新创建的HDFS目录中查询数据，您可以创建一个Greenplum可读外部表并指定

```
FORMAT 'CSV' (delimiter=':') .
```

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ PXF介绍

□ PXF管理手册

□ 使用PXF访问hadoop

读写文本数据

读取Avro数据

读取JSON数据

读写Parquet数据

读写SequenceFile数据

将多行文本文件读入单个表行

读取Hive表数据

读取HBase表数据

□ 使用PXF访问Azure, Google云端存储, Minio和S3对象存储

使用PXF访问SQL数据库(JDBC)

PXF故障排除

□ PXF实用程序手册

Back to the Administrator Guide

Greenplum database 5管理员指南

In this topic:

- [先决条件](#)
- [使用Avro数据](#)
 - [数据类型映射](#)
 - [Avro 模式和数据](#)
- [创建外部表](#)
- [示例：读取Avro数据](#)
 - [创建模式](#)
 - [创建Avro数据文件\(JSON\)](#)
 - [使用hdfs:avro配置文件查询](#)

使用PXF HDFS连接器读取Avro格式数据。本节描述如何使用PXF访问HDFS中的Avro数据，包括如何创建和查询引用HDFS中Avro文件的外部表。

先决条件

在尝试从HDFS读取或向HDFS写入数据之前，请确保已满足PXF Hadoop[先决条件](#)。

使用Avro数据

Apache Avro是一个数据序列化框架，其中的数据都以压缩二进制格式序列化。Avro在Json中指定数据类型。Avro格式数据具有独立的模式，这也在Json中定义。Avro模式及其数据完全是自描述的。

数据类型映射

Avro支持原生数据类型和复杂数据类型。

要在Greenplum数据库中表示Avro原生数据类型，请将数据值映射到相同类型的Greenplum列。

Avro支持复杂数据类型数组(array)，映射(map)，记录(record)，枚举(enumeration)以及固定长度类型(fixed type)。将这些复杂数据类型的顶层字段映射为Greenplum数据库的 `TEXT` 类型。虽然Greenplum本

身并不支持这些类型，您可以创建Greenplum数据库函数或应用程序代码来提取或进一步处理这些复杂数据类型的子部件。

下表总结了Avro数据的外部映射规则。

Avro数据类型	PXF/Greenplum 数据类型
boolean	boolean
bytes	bytea
double	double
float	real
int	int or smallint
long	bigint
string	text
复杂类型: Array, Map, Record, or Enum	text, 并在集合项、映射的键值对和记录数据之间插入分隔符。
复杂类型: Fixed	bytea
并集	对于原始数据类型或复杂数据类型，遵循上述约定，具体取决于并集；支持Null值。

Avro 模式和数据

Avro 模式使用json定义，由上面数据类型映射部分中的原生类型和复杂类型组成。Avro 模式文件通常具有 `.avsc` 后缀。

Avro模式文件中的字段是通过对象数组定义的，每个对象都由名称和类型指定。

创建外部表

使用 `hdfs:avro` 配置文件读取HDFS中的Avro格式数据，以下语法创建了引用该配置文件的可读外部表：

```
CREATE EXTERNAL TABLE <table_name>
( <column_name> <data_type> [, ...] | LIKE <other_table> )
LOCATION ('pxf://<path-to-hdfs-file>?PROFILE=hdfs:avro[&<custom-option>=<value>[...]]')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

`CREATE EXTERNAL TABLE` 命令中使用的特定关键字和值见下表中描述。

关键字	值
<code><path-to-hdfs-file></code>	HDFS数据存储中目录或文件的绝对路径
<code>PROFILE</code>	<code>PROFILE</code> 关键字必须指定为 <code>hdfs:avro</code> 。
<code>SERVER= <server_name></code>	PXF用于访问数据的命名服务器配置。可选的；如果未指定，PXF将使用 <code>default</code> 服务器。
<code><custom-option></code>	<code><custom-option></code> 描述见下方
<code>FORMAT 'CUSTOM'</code>	在 <code>hdfs:avro</code> 配置文件中使用 <code>FORMAT 'CUSTOM'</code> 。 <code>CUSTOM FORMAT</code> 要求您指定为 <code>(FORMATTER='pxfwrifiable_import')</code> 。

对于复杂数据类型，PXF `hdfs:avro` 配置文件在集合项和值之间插入默认分隔符。您可以通过指定在 `CREATE EXTERNAL TABLE` 命令中指定 `hdfs:avro` 自定义选项来使用非默认分隔符。

`hdfs:avro` 配置文件支持以下`<custom-option>`：

选项关键字	描述
<code>COLLECTION_DELIM</code>	当PXF将Avro复杂数据类型映射到文本列时，放置在顶层数组、映射或记录字段中的条目之间的分隔符。默认为逗号 <code>,</code> 字符。
<code>MAPKEY_DELIM</code>	当PXF将Avro复杂数据类型映射到文本列时，放置在映射项的键和值之间的分隔符。默认为冒号 <code>:</code> 字符。
<code>RECORDKEY_DELIM</code>	当PXF将Avro复杂数据类型映射到文本列时，放置在字段名称和记录条目之间的分隔符。默认为冒号 <code>:</code> 字符。

示例：读取Avro数据

本节中的示例将对具有以下字段名称和数据类型模式的Avro数据进行操作：

- id - 长整型

- username - 字符串
- followers - 字符串数组
- fmap - 长整型映射
- relationship - 枚举类型
- address - 由街道号码(整型)、街道名称(字符串)、以及城市(字符串)组成的记录类型

创建模式

执行以下操作创建一个Avro模式，以表示上述模式。

1. 创建一个名为 `avro_schema.avsc` 的文件:

```
$ vi /tmp/avro_schema.avsc
```

2. 将以下文本复制并粘贴到 `avro_schema.avsc` 中:

```
{
  "type" : "record",
    "name" : "example_schema",
    "namespace" : "com.example",
    "fields" : [ {
      "name" : "id",
      "type" : "long",
      "doc" : "Id of the user account"
    }, {
      "name" : "username",
      "type" : "string",
      "doc" : "Name of the user account"
    }, {
      "name" : "followers",
      "type" : {"type": "array", "items": "string"},
      "doc" : "Users followers"
    }, {
      "name": "fmap",
      "type": {"type": "map", "values": "long"}
    }, {
      "name": "relationship",
      "type": {
        "type": "enum",
        "name": "relationshipEnum",
        "symbols": [
          "MARRIED", "LOVE", "FRIEND", "COLLEAGUE", "STRANGER", "ENEMY"
        ]
      }
    }, {
      "name": "address",
      "type": {
        "type": "record",
        "name": "address"
      }
    }
  ]
}
```

```

        "name": "addressRecord",
        "fields": [
            {"name": "number", "type": "int"},
            {"name": "street", "type": "string"},
            {"name": "city", "type": "string"}]
    }
},
"doc": "A basic schema for storing messages"
}

```

创建Avro数据文件(JSON)

执行以下步骤来创建符合上述模式的示例Avro文件。

1. 创建一个名为 `pxf_avro.txt` 的文本文件:

```
$ vi /tmp/pxf_avro.txt
```

2. 在 `pxf_avro.txt` 中输入以下数据:

```

{"id":1, "username":"john", "followers":["kate", "santosh"],
"relationship": "FRIEND", "fmap": {"kate":10, "santosh":4},
"address": {"number":1, "street": "renaissance drive",
"city": "san jose"}}

{"id":2, "username": "jim", "followers": ["john", "pam"],
"relationship": "COLLEAGUE", "fmap": {"john":3, "pam":3},
"address": {"number":9, "street": "deer creek", "city": "palo
alto"}}

```

示例数据使用逗号 `,` 分隔顶层字段，并使用冒号 `:` 分隔键-值对以及记录字段和值。

3. 将文本文件转换为Avro格式文件。有多种方法可以通过编程方式和通过命令行转换。在这个例子中，我们使用[Java Avro tools](#)。

`avro-tools-1.8.1.jar` jar文件放置在当前目录中：

```
$ java -jar ./avro-tools-1.8.1.jar fromjson --schema-file
/tmp/avro_schema.avsc /tmp/pxf_avro.txt > /tmp/pxf_avro.avro
```

生成的Avro二进制文件被写入 `/tmp/pxf_avro.avro`。

4. 将生成的Avro文件复制到HDFS：

```
$ hdfs dfs -put /tmp/pxf_avro.avro /data/pxf_examples/
```

使用hdfs:avro配置文件查询

执行以下来创建和查询引用您在上一节添加到HDFS的 `pxf_avro.avro` 文件。创建表时：

- 使用PXF默认服务器。
 - 将顶层原生字段 `id` (长整型)和 `username` (字符串类型)映射到其等效的Greenplum的数据库类型(bigint和text)。
 - 将剩下的复杂类型映射为text类型
 - 使用 `hdfs:avro` 配置文件的自定义选项设置记录(record)、映射(map)和集合(collection)的分隔符
1. 使用 `hdfs:avro` 配置文件从 `pxf_avro.avro` 文件创建可查询外部表：

```
postgres=# CREATE EXTERNAL TABLE pxf_hdfs_avro(id bigint, username text, followers text, fmap text, relationship text, address text
LOCATION ('pxf://data/pxf_examples/pxf_avro.avro'
PROFILE=hdfs:avro&COLLECTION_DELIM=,&MAPKEY_DELIM=:&RECORDKEY_DELIM=
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

2. 对 `pxf_hdfs_avro` 表执行简单查询：

```
postgres=# SELECT * FROM pxf_hdfs_avro;
```

	<code>id</code>	<code>username</code>	<code>followers</code>	<code>fmap</code>	<code>relationship</code>	<code>address</code>
	1	john	[kate,santosh]	{kate:10,santosh:4}	FRIEND	{number:1,street:renaissance drive,city:san jose}
	2	jim	[john,pam]	{pam:3,john:3}	COLLEAGUE	{number:9,street:deer creek,city:palo alto}
(2 rows)						

外部表的简单查询显示了复杂数据类型的组成部分，这些组成部分由 `CREATE EXTERNAL TABLE` 中指定的分隔符分隔。

3. 根据您的应用程序处理文本列中被分隔的部分。例如，以下命令使用Greenplum数据库内部的 `string_to_array` 函数将 `followers` 字

段中的条目转换为新视图中的文本数组列。

```
postgres=# CREATE VIEW followers_view AS
  SELECT username, address, string_to_array(substr(followers
  FROM 2 FOR (char_length(followers) - 2)), ',')::text[]
    AS followers
  FROM pxf_hdfs_avro;
```

4. 根据特定的关注者是否出现在视图中来执行查询并过滤行：

```
postgres=# SELECT username, address FROM followers_view WHERE
  followers @> '{john}';
```

username	address
jim	{number:9,street:deer creek,city:palo alto}

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ PXF介绍

□ PXF管理手册

□ 使用PXF访问hadoop

读写文本数据

读取Avro数据

读取JSON数据

读写Parquet数据

读写SequenceFile数据

将多行文本文件读入单个表行

读取Hive表数据

读取HBase表数据

□ 使用PXF访问Azure, Google云端存储, Minio和S3对象存储

使用PXF访问SQL数据库(JDBC)

PXF故障排除

□ PXF实用程序手册

Back to the Administrator Guide

Greenplum database 5管理员指南

In this topic:

- [先决条件](#)
- [使用JSON数据](#)
 - [JSON到Greenplum数据库数据类型映射](#)
 - [JSON数据读取模式](#)
- [将样本JSON数据加载到HDFS](#)
- [创建外部表](#)
- [示例: 读取单行记录的JSON文件](#)

使用PXF HDFS连接器读取JSON格式数据的数据。本节描述了如何使用PXF访问HDFS中的JSON数据，包括如何创建和查询引用HDFS中JSON文件的外部表。

先决条件

在尝试从HDFS读取或向HDFS写入数据之前，请确保已满足PXF Hadoop[先决条件](#)。

使用JSON数据

JSON是基于文本的数据交换格式。JSON数据通常存储在带有 `.json` 后缀的文件中。

`.json` 文件将包含对象的集合。JSON对象是无序键/值对的集合。值可以是字符串(string)、数字(number)、true、false、null或对象(object)或数组(array)。您可以定义嵌套的JSON对象和数组。

样本JSON数据文件内容：

```
{
  "created_at": "MonSep3004:04:53+00002013",
  "id_str": "384529256681725952",
  "user": {
    "id": 31424214,
    "location": "COLUMBUS"
  },
  "coordinates": {
    "type": "Point",
    "values": [
      ...
    ]
  }
}
```

```

    13,
    99
]
}
}

```

在上述的示例中，`user` 是一个由名为 `id` 和 `location` 的字段组成的对象。要将 `user` 对象中的嵌套字段指定为Greenplum数据库外部表列，请使用 `.` 映射：

```

user.id
user.location

```

`coordinates` 是一个由名为 `type` 字段和名为 `values` 的整数数组组成的对象。使用 `[]` 来指定特定 `values` 数组中特定的元素作为Greenplum数据库外部表的列：

```

coordinates.values[0]
coordinates.values[1]

```

有关JSON语法的详细信息，请参阅[JSON 简介](#)。

JSON到Greenplum数据库数据类型映射

要在Greenplum数据库中表示JSON数据，请将使用基本数据类型的值映射到相同类型的Greenplum数据库列。JSON支持复杂的数据类型包括投影和数组。使用N级投影将嵌套对象和数组的成员映射到基本数据类型。

下表总结了JSON数据的外部映射规则。

表1. JSON映射

JSON 数据类型	PXF/Greenplum Data Type
基本类型(integer, float, string, boolean, null)	使用相应的Greenplum数据库内置数据类型；请参阅 Greenplum数据库数据类型 。
数组(Array)	使用 <code>[]</code> 括号标识基本数据类型数组特定成员的索引。
对象(Object)	使用 <code>.</code> 点号指定基本类型的每一层投影(嵌套)。

JSON数据读取模式

PXF支持两种数据读取模式。默认模式是每行一条完成的JSON记录。PXF还支持对跨多行的JSON记录进行操作的读取模式。

在接下来的示例中，您将使用两种模式对样本数据集进行操作。样本数据集的模式定义具有以下成员名称和数据类型的对象：

- “created_at” - text
- “id_str” - text
- “user” - object
 - “id” - integer
 - “location” - text
- “coordinates” - object (可选)
 - “type” - text
 - “values” - array
 - [0] - integer
 - [1] - integer

每行单条JSON记录数据如下：

```
{"created_at": "FriJun0722:45:03+00002013", "id_str": "343136551322136",
{
  "id": 395504494, "location": "NearCornwall", "coordinates": {
    "type": "Point", "values": [
      [ 6, 50 ]
    ]},
  {"created_at": "FriJun0722:45:02+00002013", "id_str": "343136547115253",
  {
    "id": 26643566, "location": "Austin, Texas", "coordinates": null},
  {"created_at": "FriJun0722:45:02+00002013", "id_str": "343136547136235",
  {
    "id": 287819058, "location": ""}, "coordinates": null}
```

这是用于多行JSON数据的数据集：

```
{
  "root": [
    {
      "record_obj": {
        "created_at": "MonSep3004:04:53+00002013",
        "id_str": "384529256681725952",
        "user": {
          "id": 31424214,
```

```

        "location": "COLUMBUS"
    },
    "coordinates": null
},
"record_obj": {
    "created_at": "MonSep3004:04:54+00002013",
    "id_str": "384529260872228864",
    "user": {
        "id": 67600981,
        "location": "KryberWorld"
    },
    "coordinates": {
        "type": "Point",
        "values": [
            8,
            52
        ]
    }
}
]
}

```

在下一节中您将为示例数据集创建JSON文件，并将其添加到HDFS中。

将样本JSON数据加载到HDFS

PXF HDFS连接器读取存储在HDFS本地的JSON文件。在使用Greenplum数据库查询JSON格式数据之前，该数据必须存在于HDFS中。

将上面的单行JSON记录样本数据集复制并黏贴到名为 `singleline.json` 的文件中。同样的，将多行JSON记录数据集复制并黏贴到名为 `multiline.json` 的文件中。

注意：确保JSON文件中没有空白行。

将您刚创建的文件复制到HDFS中。如果您在上一个练习中没有创建 `/data/pxf_examples` 目录，请创建它。例如：

```
$ hdfs dfs -mkdir /data/pxf_examples
$ hdfs dfs -put singleline.json /data/pxf_examples
$ hdfs dfs -put multiline.json /data/pxf_examples
```

数据加载到HDFS后，您可以使用Greenplum数据库和PXF查询和分析JSON数据。

创建外部表

使用 `hdfs:json` 配置文件从HDFS中读取JSON格式的文件。以下语法创建一个引用此类文件的Greenplum数据库可读外部表：

```
CREATE EXTERNAL TABLE <table_name>
  ( <column_name> <data_type> [, ...] | LIKE <other_table> )
LOCATION ('pxf://<path-to-hdfs-file>?PROFILE=hdfs:json[&SERVER=
<server_name>][&<custom-option>=<value>[...]]')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

`CREATE EXTERNAL TABLE` 命令中使用的特定关键字和值见下表中描述。

关键字	值
<path-to-hdfs-file>	HDFS数据存储中目录或文件的绝对路径
PROFILE	<code>PROFILE</code> 关键字必须指定为 <code>hdfs:json</code>
SERVER= <server_name>	PXF用于访问数据的命名服务器配置。可选的；如果未指定，PXF将使用 <code>default</code> 服务器。
<custom-option>	<custom-option>描述见下方
FORMAT 'CUSTOM'	在 <code>hdfs:json</code> 配置文件中使用 <code>FORMAT 'CUSTOM'</code> 。 <code>CUSTOM FORMAT</code> 要求您指定为 <code>(FORMATTER='pxfwritable_import')</code> 。

PXF支持单行和多行JSON记录。当您想读取多行JSON记录时，必须提供 `IDENTIFIER` <custom-option> 和值。使用这个<custom-option>标识JSON记录对象中第一个字段的成员名称：

选项关键字	语法、示例	描述
IDENTIFIER	<code>&IDENTIFIER=<value></code> <code>&IDENTIFIER=created_at</code>	仅在访问多行JSON记录时，才必须在 <code>LOCATION</code> 字符串中指定 <code>IDENTIFIER</code> 关键字和值。使用这个值标识JSON记录对象中第一个字段的成员名称。

示例：读取单行记录的JSON文件

使用以下CREATE EXTERNAL TABLE SQL命令来创建可读的外部表，该表引用每条记录单行JSON数据文件并使用PXF默认服务器。

```
CREATE EXTERNAL TABLE singleline_json_tbl(
    created_at TEXT,
    id_str TEXT,
    "user.id" INTEGER,
    "user.location" TEXT,
    "coordinates.values[0]" INTEGER,
    "coordinates.values[1]" INTEGER
)
LOCATION('pxf://data/pxf_examples/singleline.json?
PROFILE=hdfs:json')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

注意使用`.`投影访问`user`和`coordinates`对象中的嵌套字段。还要注意使用`[]`来访问`coordinates.values[]`数组中的特定元素。

查询外部表中的JSON数据：

```
SELECT * FROM singleline_json_tbl;
```

示例：读取多行记录的JSON文件

从多行JSON记录文件中创建可读外部表的SQL命令和上面单行JSON记录的非常类似。当您要读取多行JSON记录时，您必须额外指定`LOCATION`子句的`IDENTIFIER`关键字和值。例如：

```
CREATE EXTERNAL TABLE multiline_json_tbl(
    created_at TEXT,
    id_str TEXT,
    "user.id" INTEGER,
    "user.location" TEXT,
    "coordinates.values[0]" INTEGER,
    "coordinates.values[1]" INTEGER
)
LOCATION('pxf://data/pxf_examples/multiline.json?
PROFILE=hdfs:json&IDENTIFIER=created_at')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

`created_at`标识示例数据模式中JSON记录`record_obj`中第一个字段的成员名称。

查询外部表中的JSON数据:

```
SELECT * FROM multiline_json_tbl;
```

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ PXF介绍

□ PXF管理手册

□ 使用PXF访问hadoop

读写文本数据

读取Avro数据

读取JSON数据

读写Parquet数据

读写SequenceFile数据

将多行文本文件读入单个表行

读取Hive表数据

读取HBase表数据

□ 使用PXF访问Azure, Google云端存储, Minio和S3对象存储

使用PXF访问SQL数据库(JDBC)

PXF故障排除

□ PXF实用程序手册

Back to the Administrator Guide

Greenplum database 5管理员指南

In this topic:

- [先决条件](#)
- [数据类型映射](#)
- [创建外部表](#)
- [示例](#)

使用PXF HDFS连接器读写Parquet格式数据。本节介绍如何读写以Parquet格式存储的HDFS文件，包括如何创建、查询和写入引用HDFS文件的外部表。

PXF当前仅支持读写基本Parquet数据类型。

PXF Parquet写入支持是一个Beta功能。

先决条件

在尝试从HDFS读取或向HDFS写入数据之前，请确保已满足PXF Hadoop[先决条件](#)。

数据类型映射

要在Greenplum数据库中读写Parquet基本数据类型，请将Parquet数据值映射到相同类型的Greenplum数据库列。下表总结了外部映射规则：

Parquet数据类型	PXF/Greenplum数据类型
boolean	Boolean
byte_array	Bytea, Text
double	Float8
fixed_len_byte_array	Numeric
float	Real
int_8, int_16	Smallint, Integer
int64	Bigint
int96	Timestamp, Timestamptz

写入Parquet时：

- PXF将 `timestamp` 本地化为当前系统时区，并将其转换为通用时间(UTC)，然后最终转换为 `int96`。
- PXF将 `timestamp` 转换为UTC `timestamp`，然后转换为 `int96`。在此转换过程中，PXF会丢失时区信息。

创建外部表

PXF HDFS连接器 `hdfs:parquet` 配置文件支持读写Parquet格式的HDFS数据。当您将记录插入可写外部表中时，您插入的数据块将写入指定目录中一个或多个文件。

使用以下语法创建引用HDFS目录的Greenplum数据库外部表：

```
CREATE [WRITABLE] EXTERNAL TABLE <table_name>
  ( <column_name> <data_type> [, ...] | LIKE <other_table> )
LOCATION ('pxf://<path-to-hdfs-dir>
?PROFILE=hdfs:parquet[&SERVER=<server_name>][&<custom-
option>=<value>[...]]')
FORMAT 'CUSTOM'
(FORMATTER='pxfwritable_import' | 'pxfwritable_export');
[DISTRIBUTED BY (<column_name> [, ... ] ) | DISTRIBUTED
RANDOMLY];
```

`CREATE EXTERNAL TABLE` 命令中使用的特定关键字和值见下表中描述。

关键字	值
<code><path-to-hdfs-file></code>	HDFS数据存储中目录或文件的绝对路径
<code>PROFILE</code>	<code>PROFILE</code> 关键字必须指定为 <code>hdfs:parquet</code>
<code>SERVER=</code> <code><server_name></code>	PXF用于访问数据的命名服务器配置。可选的；如果未指定，PXF将使用 <code>default</code> 服务器。
<code><custom-option>=</code> <code><value></code>	<code><custom-option></code> 描述见下方
<code>FORMAT</code> <code>'CUSTOM'</code>	使用 <code>FORMAT</code> ' <code>CUSTOM</code> ' 时指定 (<code>FORMATTER='pxfwritable_export'</code>) (写入) 或 (<code>FORMATTER='pxfwritable_import'</code>) (读取)。
<code>DISTRIBUTED</code> <code>BY</code>	如果您计划将现有Greenplum数据库表中的数据加载到可写外部表，考虑在可写外部表上使用与Greenplum表相同的分布策略或<字段名>。这样做可以避免数据加载操作中segment节点间

额外的数据移动。

PXF `hdfs:parquet` 配置文件支持与编码和压缩有关的写入选项。您可以在 `CREATE WRITABLE EXTERNAL TABLE` `LOCATION` 子句中指定这些写入选项。`hdfs:parquet` 配置文件支持以下自定义选项：

写入选项	值描述
<code>COMPRESSION_CODEC</code>	压缩编码器别名。用于写入Parquet数据受支持的压缩编码器包括： <code>snappy</code> , <code>gzip</code> , <code>lzo</code> , 和 <code>uncompressed</code> 。如果未提供此选项, PXF将使用 <code>snappy</code> 压缩编码器来压缩数据。
<code>ROWGROUP_SIZE</code>	Parquet文件由一个或多个行组组成, 将数据逻辑划分为行。 <code>ROWGROUP_SIZE</code> 标识行组的大小(以字节为单位)。默认的行组大小为 <code>8 * 1024 * 1024</code> 字节。
<code>PAGE_SIZE</code>	行组由划分为页面的列块组成。 <code>PAGE_SIZE</code> 是此类页面的大小(以字节为单位)。默认的页面大小为 <code>1024 * 1024</code> 字节。
<code>DICTIONARY_PAGE_SIZE</code>	当PXF写入Parquet文件时, 默认情况下启用字典编码。每列、每行组只有一个字典页面。 <code>DICTIONARY_PAGE_SIZE</code> 与 <code>PAGE_SIZE</code> 类似, 但是对于字典而言: 默认字典页面大小为 <code>512 * 1024</code> 字节。
<code>PARQUET_VERSION</code>	Parquet版本; 支持 <code>v1</code> 和 <code>v2</code> 。默认的Parquet版本为 <code>v1</code> 。

注意: 如果您不希望PXF压缩数据, 则必须明确指定 `uncompressed`。

使用PXF写入HDFS的Parquet文件具有以下命名格式:

`<file>. <compress_extension>.parquet`, 例如
`1547061635-0000004417_0.gz.parquet`.

示例

本示例采用在[示例：读取HDFS中的文本数据](#)中介绍的数据模式。

列名	数据类型
location	text
month	text
number_of_orders	int
total_sales	float8

在此示例中，您创建一个Parquet格式的可写外部表，该表使用默认的PXF服务器引用HDFS中的Parquet格式的数据，将一些数据插入表中，然后创建一个可读外部表以读取数据。

1. 使用 `hdfs:parquet` 配置文件创建一个可写外部表。例如：

```
postgres=# CREATE WRITABLE EXTERNAL TABLE pxf_tbl_parquet
(location text, month text, number_of_orders int, total_sales
double precision)
LOCATION ('pxf://data/pxf_examples/pxf_parquet?
PROFILE=hdfs:parquet')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_export');
```

2. 通过直接写入 `pxf_tbl_parquet` 表将一些记录写入HDFS的 `pxf_parquet` 目录中。例如：

```
postgres=# INSERT INTO pxf_tbl_parquet VALUES ( 'Frankfurt',
'Mar', 777, 3956.98 );
postgres=# INSERT INTO pxf_tbl_parquet VALUES ( 'Cleveland',
'Oct', 3812, 96645.37 );
```

3. 回想一下，Greenplum 数据库不支持直接查询一个可写外部表。要读取 `pxf_parquet` 中的数据，请创建一个引用此HDFS目录的可读外部表：

```
postgres=# CREATE EXTERNAL TABLE read_pxf_parquet(location
text, month text, number_of_orders int, total_sales double
precision)
LOCATION ('pxf://data/pxf_examples/pxf_parquet?
PROFILE=hdfs:parquet')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

4. 查询可读外部表 `read_pxf_parquet`：

```
postgres=# SELECT * FROM read_pxf_parquet ORDER BY
total_sales;
```

location	month	number_of_orders	total_sales
Frankfurt	Mar	777	3956.98
Cleveland	Oct	3812	96645.4
(2 rows)			

Greenplum数据库® 6.0文档

Greenplum平台扩展框
架(PXF)

PXF介绍

PXF管理手册

使用PXF访问hadoop

读写文本数据

读取Avro数据

读取JSON数据

读写Parquet数据

读写SequenceFile数据

将多行文本文件读入单个表行

读取Hive表数据

读取HBase表数据

使用PXF访问Azure, Google云端存储, Minio和S3对象存储

使用PXF访问SQL数据库(JDBC)

PXF故障排除

PXF实用程序手册

Back to the Administrator Guide

Greenplum database 5管理员指南

In this topic:

- [先决条件](#)
- [创建外部表](#)
- [读写二进制数据](#)
- [读取记录键](#)
 - [示例: 使用记录键](#)

PXF HDFS连接器支持SequenceFile格式的二进制数据。本节描述如何使用PXF读写HDFS SequenceFile数据，包括如何创建引用HDFS文件的外部表，查询和向外部表写入数据。

先决条件

在尝试从HDFS读取或向HDFS写入数据之前，请确保已满足PXF Hadoop先决条件。

创建外部表

PXF HDFS连接器 `hdfs:SequenceFile` 配置文件支持读写HDFS中SequenceFile格式的二进制数据。当您将记录插入可写外部表中时，您插入的数据块将写入指定目录中的一个或多个文件。

注意：使用可写配置创建的外部表仅INSERT操作。如果要查询插入的数据，则必须单独创建一个引用相关HDFS目录的可读外部表。

使用以下语法创建一个引用HDFS目录的Greenplum数据库外部表：

```
CREATE [WRITABLE] EXTERNAL TABLE <table_name>
  ( <column_name> <data_type> [, ...] | LIKE <other_table> )
LOCATION ('pxf://<path-to-hdfs-dir>
?PROFILE=hdfs:SequenceFile[&SERVER=<server_name>][&<custom-option>=<value>[...]]')
FORMAT 'CUSTOM' (<formatting-properties>)
[DISTRIBUTED BY (<column_name> [, ... ] ) | DISTRIBUTED RANDOMLY];
```

`CREATE EXTERNAL TABLE`命令中使用的特定关键字和值见下表中描述。

关键字	值
<path-to-hdfs-dir>	HDFS数据存储中目录或文件的绝对路径
PROFILE	<code>PROFILE</code> 关键字必须指定为 <code>hdfs:SequenceFile</code> 。
SERVER=	PXF用于访问数据的命名服务器配置。可选的；如果未指

<server_name>	定，PXF将使用 default 服务器。
<custom-option>	<custom-option> 描述见下方。
FORMAT	使用 FORMAT , CUSTOM 时指定 (FORMATTER= 'pxfwritable_export') (写入) 或 (FORMATTER= 'pxfwritable_import') (读取)。
DISTRIBUTED BY	如果您计划将现有Greenplum数据库表中的数据加载到可写外部表，考虑在可写外部表上使用与Greenplum表相同的分布策略或<字段名>。这样做可以避免数据加载操作中segment节点间额外的数据移动。

SequenceFile 格式数据可以选择采用record或block压缩。PXF

`hdfs:SequenceFile` 配置支持以下压缩编解码器：

- org.apache.hadoop.io.compress.DefaultCodec
- org.apache.hadoop.io.compress.BZip2Codec

使用 `hdfs:SequenceFile` 配置文件写入SequenceFile格式数据时，则必须提供Java类的名称，以用于序列化/反序列化二进制数据。这个类必须为数据模式中引用的每种数据类型提供读写方法。

您可以通过 `CREATE EXTERNAL TABLE` `LOCATION` 子句的自定义选项指定压缩编解码器和Java 序列化类。`hdfs:SequenceFile` 配置文件支持以下自定义选项：

选项	值描述
COMPRESSION_CODEC	压缩编码器类名称。如果此选项未提供，Greenplum 数据库不执行数据压缩。支持的压缩编解码器包括： <code>org.apache.hadoop.io.compress.DefaultCodec</code> <code>org.apache.hadoop.io.compress.BZip2Codec</code> <code>org.apache.hadoop.io.compress.GzipCodec</code>
COMPRESSION_TYPE	采用的压缩类型；支持的值有 <code>RECORD</code> (默认) 或 <code>BLOCK</code> 。
DATA-SCHEMA	编写器序列化/反序列化类的名称。此类所在的jar文件必须位于PXF类路径中。 <code>hdfs:SequenceFile</code> 配置文件需要此选项，并且没有默认值。
THREAD-SAFE	确定表查询是否可以在多线程模式下运行的布尔值。默认为 <code>TRUE</code> 。将此选项设置为 <code>FALSE</code> 以处理单个线程中所有非线程安全操作 (例如，压缩) 的请求。

读写二进制数据

当您要读写HDFS中 SequenceFile 格式的数据, 请使用HDFS 连接器 `hdfs:SequenceFile` 配置文件。这种类型的文件由二进制键/值对组成。SequenceFile 格式是MapReduce作业之间常见的数据传输格式。

示例： 将二进制数据写入HDFS

在此示例中, 您将创建一个名为 `PxfExample_CustomWritable` 的Java类, 该类将对先前示例中使用的示例模式中的字段进行序列化/反序列化。然后, 您将使用此类访问通过 `hdfs:SequenceFile` 配置文件创建的可写外部表, 该表使用默认的PXF服务器。

执行以下步骤来创建Java类和可写外部表。

- 准备创建示例Java类:

```
$ mkdir -p pxfex/com/example/pxf/hdfs/writable/dataschema
$ cd pxfex/com/example/pxf/hdfs/writable/dataschema
$ vi PxfExample_CustomWritable.java
```

- 将以下文本复制并黏贴到 `PxfExample_CustomWritable.java` 文件中:

```
package com.example.pxf.hdfs.writable.dataschema;

import org.apache.hadoop.io.*;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.lang.reflect.Field;

/**
 * PxfExample_CustomWritable class - used to serialize and deserialize
 * data with
 * text, int, and float data types
 */
public class PxfExample_CustomWritable implements Writable {

    public String st1, st2;
    public int int1;
    public float ft;

    public PxfExample_CustomWritable() {
        st1 = new String("");
        st2 = new String("");
        int1 = 0;
        ft = 0.f;
    }

    public PxfExample_CustomWritable(int i1, int i2, int i3) {

        st1 = new String("short_string__" + i1);
        st2 = new String("short_string__" + i1);
        int1 = i2;
        ft = i1 * 10.f * 2.3f;
    }

}
```

```

String GetSt1() {
    return st1;
}

String GetSt2() {
    return st2;
}

int GetInt1() {
    return int1;
}

float GetFt() {
    return ft;
}

@Override
public void write(DataOutput out) throws IOException {

    Text txt = new Text();
    txt.set(st1);
    txt.write(out);
    txt.set(st2);
    txt.write(out);

    IntWritable intw = new IntWritable();
    intw.set(int1);
    intw.write(out);

    FloatWritable fw = new FloatWritable();
    fw.set(ft);
    fw.write(out);
}

@Override
public void readFields(DataInput in) throws IOException {

    Text txt = new Text();
    txt.readFields(in);
    st1 = txt.toString();
    txt.readFields(in);
    st2 = txt.toString();

    IntWritable intw = new IntWritable();
    intw.readFields(in);
    int1 = intw.get();

    FloatWritable fw = new FloatWritable();
    fw.readFields(in);
    ft = fw.get();
}

public void printFieldTypes() {
    Class myClass = this.getClass();
    Field[] fields = myClass.getDeclaredFields();

    for (int i = 0; i < fields.length; i++) {
        System.out.println(fields[i].getType().getName());
    }
}
}

```

3. 编译并为 `PxfExample_CustomWritable` 创建Java类JAR文件。为您
的Hadoop发行版提供一个包含 `hadoop-common.jar` 文件的类路径。例如,
如果您安装了 Hortonworks Data Platform Hadoop客户端：

```
$ javac -classpath /usr/hdp/current/hadoop-client/hadoop-common.jar  
PxfExample_CustomWritable.java  
$ cd ../../../../../../  
$ jar cf pxfex-customwritable.jar com  
$ cp pxfex-customwritable.jar /tmp/
```

(您的Hadoop库类路径可能不同)

4. 将 `pxfex-customwritable.jar` 文件复制到Greenplum 数据库master节点。例
如：

```
$ scp pxfex-customwritable.jar gpadmin@gpmaster:/home/gpadmin
```

5. 登录到您的Greenplum数据库master节点：

```
$ ssh gpadmin@<gpmaster>
```

6. 将 `pxfex-customwritable.jar` JAR文件复制到用户运行时类库目录中，并
记下位置。例如，如果 `PXF_CONF=/usr/local/greenplum-pxf`：

```
gpadmin@gpmaster$ cp /home/gpadmin/pxfex-customwritable.jar  
/usr/local/greenplum-pxf/lib/pxfex-customwritable.jar
```

7. 将PXF配置同步到Greenplum数据库集群。例如：

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster sync
```

8. 如[重启PXF](#)中所述，在每个Greenplum数据库segment主机上重启PXF。

9. 使用PXF `hdfs:SequenceFile` 配置文件创建Greenplum数据库可写外部表。
在 `DATA-SCHEMA` 自定义选项 `<custom-option>` 中标识您在上面创建的序列
化/反序列化Java类。创建可写外部表时，使用 `BZip2` 压缩并指定 `BLOCK`
压缩模式。

```
postgres=# CREATE EXTERNAL TABLE pxf_tbl_seqfile (location text,  
LOCATION ('pxf://data/pxf_examples/pxf_seqfile?PROFILE=hdfs:Se  
SCHEMA=com.example.pxf.hdfs.writable.dataschema.PxfExample_CustomWritable&  
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_export'));
```

注意 `'CUSTOM'` `FORMAT` 格式化属性 `<formatting-properties>` 指定为内置的
`pxfwritable_export` 格式化程序。

10. 通过直接插入 `pxf_tbl_seqfile` 表将一些数据写入 `pxf_seqfile` HDFS目

录中。例如：

```
postgres=# INSERT INTO pxf_tbl_seqfile VALUES ( 'Frankfurt', 'Mar', 777,
3956.98 );
postgres=# INSERT INTO pxf_tbl_seqfile VALUES ( 'Cleveland', 'Oct',
3812, 96645.37 );
```

11. 回想一下，Greenplum数据库不支持直接查询一个可写外部表。要读取 `pxf_seqfile` 中的数据，请创建一个引用此HDFS目录的可读外部表：

```
postgres=# CREATE EXTERNAL TABLE read_pxf_tbl_seqfile (location text, month text, number_of_orders integer, total_sales real)
LOCATION ('pxf://data/pxf_examples/pxf_seqfile?'
PROFILE=hdfs:SequenceFile&DATA-
SCHEMA=com.example.pxf.hdfs.writable.dataschema.PxfExample_CustomWritable'
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

当您通过 `hdfs:SequenceFile` 配置文件读取HDFS数据是必须指定 `DATA-SCHEMA` 自定义选项 <custom-option>。您无需提供压缩相关的选项。

12. 查询可读外部表 `read_pxf_tbl_seqfile`：

```
gpadmin=# SELECT * FROM read_pxf_tbl_seqfile ORDER BY total_sales;
```

location	month	number_of_orders	total_sales
Frankfurt	Mar	777	3956.98
Cleveland	Oct	3812	96645.4
(2 rows)			

读取记录键

当Greenplum数据库外部表引用SequenceFile或其他以键-值对存储行的数据格式时，您可以通过使用 `recordkey` 关键字作为字段名称，在Greenplum查询中访问记录键的值。

`recordkey` 字段类型必须与记录键类型相对应，就像其他字段必须与HDFS数据匹配一样。.

您可以将 `recordkey` 定义为以下任何Hadoop类型：

- BooleanWritable
- ByteWritable
- DoubleWritable
- FloatWritable
- IntWritable

- LongWritable
- Text

如果没有为行定义记录键， Greenplum数据库将返回处理该行的segment的id。

示例： 使用记录键

创建一个可读外部表，以访问您在 [示例： 将二进制数据写入HDFS](#) 创建的可写外部表 `pxf_tbl_seqfile` 的记录键。本例中将 `recordkey` 定义为 `int8` 类型。

```
postgres=# CREATE EXTERNAL TABLE read_pxf_tbl_seqfile_recordkey(recordkey
int8, location text, month text, number_of_orders integer, total_sales
real)
      LOCATION ('pxf://data/pxf_examples/pxf_seqfile?'
PROFILE=hdfs:SequenceFile&DATA-
SCHEMA=com.example.pxf.hdfs.writable.dataschema.PxfExample_CustomWritable')
      FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
gpadmin=# SELECT * FROM read_pxf_tbl_seqfile_recordkey;
```

recordkey	location	month	number_of_orders	total_sales
2	Frankfurt	Mar		777 3956.98
1	Cleveland	Oct		3812 96645.4

(2 rows)

当您将行插入到可写外部表时，您没有定义记录键，因此 `recordkey` 标识为处理该行的segment。

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ PXF介绍

□ PXF管理手册

□ 使用PXF访问hadoop

读写文本数据

读取Avro数据

读取JSON数据

读写Parquet数据

读写SequenceFile数据

将多行文本文件读入单个表行

读取Hive表数据

读取HBase表数据

□ 使用PXF访问Azure, Google云端存储, Minio和S3对象存储

使用PXF访问SQL数据库(JDBC)

PXF故障排除

□ PXF实用程序手册

Back to the Administrator Guide

Greenplum database 5管理员指南

In this topic:

- [先决条件](#)
- [Hive数据格式](#)
- [数据类型映射](#)
- [样本数据集](#)
- [Hive命令行](#)
- [查询外部Hive数据](#)
- [访问TextFile格式的Hive表](#)
- [访问RCFile格式的Hive表](#)
- [访问ORC格式的Hive表](#)
- [访问Parquet格式的Hive表](#)
- [处理复杂数据类型](#)
- [分区过滤下推](#)
 - [示例: 使用Hive配置文件访问同构分区的数据](#)
 - [示例: 使用Hive配置文件访问异构分区的数据](#)
- [使用PXF访问Hive默认分区](#)

Apache Hive是一个分布式数据仓库基础架构。Hive有助于管理支持多种数据格式的大型数据集，包括逗号分隔值(.csv)、TextFile、RCFile、ORC、和Parquet。

PXF Hive连接器读取Hive表中存储的数据。本节描述如何使用PXF Hive连接器。

先决条件

在使用PXF处理Hive表数据之前，请确保您已满足PXF Hadoop [先决条件](#)。

如果您打算将PXF过滤器下推与Hive整数类型一起使用，请确保配置参数 `hive.metastore.integral.jdo.pushdown` 存在，并且在两个配置文件中的 `hive-site.xml` 中均设置为 `true`。Hadoop集群和 `$PXF_CONF/servers/default/hive-site.xml`。有关更多信息，请参考[更新Hadoop配置](#)。

Hive数据格式

PXF Hive连接器支持多种数据格式，并定义了以下用于访问这些格式的配置文件：

文件格式	描述	配置文件
TextFile	平面文件，其数据以逗号、制表符或空格分隔的值格式或JSON表示法。	Hive, HiveText
SequenceFile	平面文件，由二进制键/值对组成。	Hive
RCFile	列式记录文件，由二进制键/值对组成；高行压缩率。	Hive, HiveRC
ORC	优化了的行列数据文件，具有stripe、footer、和postscript部分；减少数据大小。	Hive, HiveORC, HiveVectorizedORC
Parquet	压缩的列式数据。	Hive

注意：Hive 配置文件支持所有文件存储格式。它将对底层文件类型使用最佳的 Hive* 配置文件。

数据类型映射

PXF Hive连接器支持基础和复杂数据类型。

基础数据类型

要在Greenplum数据库中展示Hive数据，请将使用基础数据类型的数据值映射到相同类型的Greenplum数据列。

下表总结了Hive基本数据类型的外部映射规则。

Hive 数据类型	Greenplum 数据类型
boolean	bool
int	int4
smallint	int2
tinyint	int2
bigint	int8

float	float4
double	float8
string	text
binary	bytea
timestamp	timestamp

注意: `HiveVectorizedORC` 配置文件不支持 timestamp 数据类型。

复杂数据类型

Hive支持的数据类型，包括数组(array)，结构(struct)，映射(map)，以及混合类型。PXF 将以上复杂数据类型映射为 `text`。您可以创建Greenplum数据库函数或应用程序代码来提取这些复杂数据类型的子部分。

本章稍后提供通过 `Hive` 和 `HiveORC` 配置文件使用复杂数据类型的示例。

注意: `HiveVectorizedORC` 配置文件不支持复杂类型。

样本数据集

本主题中介绍的示例在公共数据集上运行。这个简单的数据集为零售业务建模，并包含具有以下名称和数据类型的字段：

字段名	数据类型
location	text
month	text
number_of_orders	integer
total_sales	double

准备样本数据集以备用：

- 首先，创建一个文本文件：

```
$ vi /tmp/pxf_hive_datafile.txt
```

- 将以下数据添加到 `pxf_hive_datafile.txt`；注意使用逗号 `,` 分

隔四个字段的值：

```
Prague,Jan,101,4875.33
Rome,Mar,87,1557.39
Bangalore,May,317,8936.99
Beijing,Jul,411,11600.67
San Francisco,Sept,156,6846.34
Paris,Nov,159,7134.56
San Francisco,Jan,113,5397.89
Prague,Dec,333,9894.77
Bangalore,Jul,271,8320.55
Beijing,Dec,100,4248.41
```

记住 `pxf_hive_datafile.txt` 的路径；您将在后续的练习中使用它。

Hive命令行

Hive命令行是类似 `psql` 的子系统。启动Hive命令行：

```
$ HADOOP_USER_NAME=hdfs hive
```

默认的Hive数据库名为 `default`。

示例：创建Hive表

创建一个Hive表以展现示例数据集。

- 在 `default` 数据库中创建一个名为 `sales_info` 的Hive表：

```
hive> CREATE TABLE sales_info (location string, month string,
   number_of_orders int, total_sales double)
 ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
 STORED AS textfile;
```

注意：

- `STORED AS textfile` 子句指示Hive以 Textfile (默认) 格式创建表。Hive Textfile格式支持逗号、制表符和空格分隔的值，以及用JSON表示的数据。
- `DELIMITED FIELDS TERMINATED BY` 子句定义数据记录(行)中的字段分隔符。`sales_info` 表字段分隔符是逗号(`,`)。

- 将 `pxf_hive_datafile.txt` 示例数据文件加载到您刚刚创建的

`sales_info` 表中：

```
hive> LOAD DATA LOCAL INPATH '/tmp/pxf_hive_datafile.txt'
      INTO TABLE sales_info;
```

在本章稍后的例子中，您将通过PXF直接访问 `sales_info` Hive 表。另外您还将在其他Hive文件格式类型的表中插入 `sales_info` 数据，并使用PXF直接访问这些数据。

3. 在 `sales_info` 上执行查询以验证是否成功加载了数据：

```
hive> SELECT * FROM sales_info;
```

确定Hive表的HDFS位置

如果需要指定Hive表的HDFS文件位置，请使用其HDFS文件路径引用它。您可以通过 `DESCRIBE` 命令确定Hive表在HDFS中的位置。例如：

```
hive> DESCRIBE EXTENDED sales_info;
Detailed Table Information
...
location:hdfs://<namenode>:<port>/apps/hive/warehouse/sales_info
...
```

查询外部Hive数据

您可以创建一个Greenplum数据外部表以访问Hive表中的数据。如前所述，PXF Hive 连接器定义了特定的配置文件以支持不同的文件格式。这些配置文件分别为 `Hive`、`HiveText`、`HiveRC`、`HiveORC` 和 `HiveVectorizedORC`。

`HiveText` 和 `HiveRC` 配置文件分别针对文本和RCFile格式进行了优化。`HiveORC` 和 `HiveVectorizedORC` 配置文件针对ORC文件格式进行了优化。`Hive` 配置文件为所有文件存储类型进行了优化。当Hive表底层由不同文件格式的多个分区组成时，您可以使用 `Hive` 配置文件。

使用以下语法创建引用Hive表的Greenplum数据库外部表：

```

CREATE EXTERNAL TABLE <table_name>
  ( <column_name> <data_type> [, ...] | LIKE <other_table> )
LOCATION ('pxf://<hive-db-name>.<hive-table-name>
?
PROFILE=Hive|HiveText|HiveRC|HiveORC|HiveVectorizedORC[&SERVER=
<server_name>]')
FORMAT 'CUSTOM|TEXT' (FORMATTER='pxfwritable_import' |
delimiter='<delim>')

```

`CREATE EXTERNAL TABLE` 命令中Hive连接器使用的特定关键字和值见下表中描述。

关键字	值
<hive-db-name>	Hive数据库的名称。如果省略，默认为名为 <code>default</code> 的Hive数据库
<hive-table-name>	Hive表的名称
PROFILE	<code>PROFILE</code> 关键字的值必须指定为 <code>Hive</code> , <code>HiveText</code> , <code>HiveRC</code> , <code>HiveORC</code> , 或 <code>HiveVectorizedORC</code> 之一
SERVER= <server_name>	PXF用于访问数据的命名服务器配置。可选的；如果未指定，PXF将使用 <code>default</code> 服务器。
FORMAT (<code>Hive</code> , <code>HiveORC</code> , 和 <code>HiveVectorizedORC</code> 配置文件)	<code>FORMAT</code> 子句必须指定为 <code>'CUSTOM'</code> 。 <code>CUSTOM</code> 格式需要内置的 <code>pxfwritable_import</code> <code>formatter</code>
FORMAT (<code>HiveText</code> 和 <code>HiveRC</code> 配置文件)	<code>FORMAT</code> 子句必须指定为 <code>TEXT</code> 。在 <code>delimiter = '<delim>'</code> 格式选项中指定单个ascii字符字段定界符。

访问TextFile格式的Hive表

您可以使用 `Hive` 和 `HiveText` 配置文件来访问以TextFile格式存储的Hive表数据。

示例：使用Hive配置文件

使用 `Hive` 配置文件创建一个可读的Greenplum数据库外部表，该表

引用此前您创建的Hive文本格式表 `sales_info`。

1. 创建外部表:

```
postgres=# CREATE EXTERNAL TABLE
salesinfo_hiveprofile(location text, month text, num_orders
int, total_sales float8)
LOCATION ('pxf://default.sales_info?
PROFILE=Hive')
FORMAT 'custom' (FORMATTER='pxfwritable_import');
```

2. 查询表:

```
postgres=# SELECT * FROM salesinfo_hiveprofile;
```

location	month	num_orders	total_sales
Prague	Jan	101	4875.33
Rome	Mar	87	1557.39
Bangalore	May	317	8936.99
...			

示例: 使用HiveText配置文件

使用 `HiveText` 配置文件创建一个可读的Greenplum数据库外部表，该表引用此前您创建的Hive文本格式表 `sales_info`。

1. 创建外部表:

```
postgres=# CREATE EXTERNAL TABLE
salesinfo_hivetextprofile(location text, month text,
num_orders int, total_sales float8)
LOCATION ('pxf://default.sales_info?
PROFILE=HiveText')
FORMAT 'TEXT' (delimiter=E',');
```

注意，`FORMAT` 子句 `delimiter` 值指定为单个ASCII逗号字符 `,`。
。 `E` 转义字符。

2. 查询外部表:

```
postgres=# SELECT * FROM salesinfo_hivetextprofile WHERE
location='Beijing';
```

location	month	num_orders	total_sales
Beijing	Jul	411	11600.67
Beijing	Dec	100	4248.41
(2 rows)			

访问RCFile格式的Hive表

RCFile Hive表格式用于行列格式的数据。PXF `HiveRC` 配置文件提供对RCFile数据的访问。

示例: 使用HiveRC配置文件

使用 `HiveRC` 配置文件在Hive中查询RCFile格式的数据。

- 启动 `hive` 命令行并创建一个以RCFile格式存储的Hive表:

```
$ HADOOP_USER_NAME=hdfs hive
```

```
hive> CREATE TABLE sales_info_rcfile (location string, month
string,
number_of_orders int, total_sales double)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS rcfile;
```

- 将 `sales_info` 表的数据写入到 `sales_info_rcfile`:

```
hive> INSERT INTO TABLE sales_info_rcfile SELECT * FROM
sales_info;
```

样本数据集的副本现在以RCFile格式存储在Hive
`sales_info_rcfile` 表中。

- 查询 `sales_info_rcfile` Hive 表以验证数据是否正确加载:

```
hive> SELECT * FROM sales_info_rcfile;
```

- 使用 PXF `HiveRC` 配置文件创建Greenplum数据库可读外部表, 该表引用此前步骤您创建的Hive `sales_info_rcfile` 表。例如:

```
postgres=# CREATE EXTERNAL TABLE
salesinfo_hivercprofile(location text, month text, num_orders
int, total_sales float8)
  LOCATION ('pxf://default.sales_info_rcfile?
PROFILE=HiveRC')
  FORMAT 'TEXT' (delimiter=E',');
```

5. 查询外部表:

```
postgres=# SELECT location, total_sales FROM
salesinfo_hivercprofile;
```

location		total_sales
Prague		4875.33
Rome		1557.39
Bangalore		8936.99
Beijing		11600.67
...		

访问ORC格式的Hive表

优化行列(ORC)文件格式是一种列文件格式，提供了一种高效的方式来存储和访问HDFS数据。ORC格式在压缩和性能方面都优于文本和RCFile格式。PXF支持ORC 1.2.1版本。

ORC具有类型感知能力，且专门针对Hadoop工作负载而设计。ORC文件存储文件中数据的类型和编码信息。一组行数据(也称为stripe)中的所有列一起以ORC格式文件存储于磁盘上。ORC格式类型的列性质允许进行读取投影，从而有助于避免在查询期间访问不必要的列。

ORC还支持在file, stripe, row级别使用内置索引进行谓词下推，从而将过滤操作移至数据加载阶段。

有关ORC文件格式的详细信息，请参考[Apache orc](#) 和 [Apache Hive LanguageManual ORC](#) 网站。

支持ORC文件格式的配置文件

选择支持ORC的配置文件时，请考虑以下事情：

- [HiveORC](#) 配置文件:

- 一次读取一行
- 支持列投影
- 支持复杂类型。您可以访问由数据(array), 映射(map), 结构(struct), 和联合数据类型组成的Hive表。PXF 将这些复杂类型序列化为 `text`。

- `HiveVectorizedORC` 配置文件:

- 一次最多读取1024行
- 不支持列投影
- 不支持复杂类型或 timestamp 数据类型

示例: 使用HiveORC配置文件

在接下来的例子中, 您将创建以ORC格式存储的Hive表, 并使用 `HiveORC` 配置文件查询此Hive表。

1. 用ORC文件格式创建Hive表:

```
$ HADOOP_USER_NAME=hdfs hive
```

```
hive> CREATE TABLE sales_info_ORC (location string, month
string,
    number_of_orders int, total_sales double)
STORED AS ORC;
```

2. 将 `sales_info` 表的数据写入到 `sales_info_ORC`:

```
hive> INSERT INTO TABLE sales_info_ORC SELECT * FROM
sales_info;
```

样本数据集的副本现在以ORC格式存储在 `sales_info_ORC` 中。

3. 在 `sales_info_ORC` 上执行Hive查询以验证数据是否正确加载:

```
hive> SELECT * FROM sales_info_ORC;
```

4. 启动 `psql` 子系统并打开计时:

```
$ psql -d postgres
```

```
postgres=> \timing
Timing is on.
```

5. 使用PXF `HiveORC` 配置文件创建Greenplum数据库外部表，该表引用此前您在步骤 1 创建的名为 `sales_info_ORC` 的Hive表。

`FORMAT` 子句必须指定为 `'CUSTOM'`。`HiveORC` `CUSTOM` 格式仅支持内置的 `'pxfwritable_import'` `formatter`。

```
postgres=> CREATE EXTERNAL TABLE
salesinfo_hiveORCprofile(location text, month text,
num_orders int, total_sales float8)
LOCATION ('pxf://default.sales_info_ORC?
PROFILE=HiveORC')
FORMAT 'CUSTOM'
(FORMATTER='pxfwritable_import');
```

6. 查询外部表:

```
postgres=> SELECT * FROM salesinfo_hiveORCprofile;
```

location	month	number_of_orders	total_sales
Prague	Jan	101	4875.33
Rome	Mar	87	1557.39
Bangalore	May	317	8936.99
...			

Time: 425.416 ms

示例: 使用HiveVectorizedORC配置文件

在以下示例中，您将使用 `HiveVectorizedORC` 配置文件查询您在此前示例中创建的 `sales_info_ORC` Hive表。

1. 启动 `psql` 子系统:

```
$ psql -d postgres
```

2. 使用PXF `HiveVectorizedORC` 配置文件创建Greenplum数据库外部表，该表引用您在此前示例步骤 1 中创建的名为 `sales_info_ORC` 的Hive表。`FORMAT` 子句必须指定为 `'CUSTOM'`。`HiveVectorizedORC` `CUSTOM` 格式仅支持内置的 `'pxfwritable_import'` `formatter`。

```
postgres=> CREATE EXTERNAL TABLE
salesinfo_hiveVectORC(location text, month text, num_orders
int, total_sales float8)
LOCATION ('pxf://default.sales_info_ORC?'
PROFILE=HiveVectorizedORC')
FORMAT 'CUSTOM'
(FORMATTER='pxfwritable_import');
```

3. 查询外部表:

```
postgres=> SELECT * FROM salesinfo_hiveVectORC;
```

location	month	number_of_orders	total_sales
Prague	Jan	101	4875.33
Rome	Mar	87	1557.39
Bangalore	May	317	8936.99
...			

Time: 425.416 ms

访问Parquet格式的Hive表

PXF **Hive** 配置文件支持使用Parquet存储格式的非分区和分区Hive表。使用等效的Greenplum数据库数据类型映射Hive表列。例如,如果在 **default** 模式中使用以下命令创建一个Hive表:

```
hive> CREATE TABLE hive_parquet_table (location string, month
string,
number_of_orders int, total_sales double)
STORED AS parquet;
```

定义Greenplum数据库外部表:

```
postgres=# CREATE EXTERNAL TABLE pxf_parquet_table (location
text, month text, number_of_orders int, total_sales double
precision)
LOCATION ('pxf://default.hive_parquet_table?profile=Hive')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

查询外部表:

```
postgres=# SELECT month, number_of_orders FROM
```

```
pxf_parquet_table;
```

处理复杂数据类型

示例: 使用Hive配置文件处理复杂数据类型

本示例中使用 `Hive` 配置文件以及数组和映射复杂数据类型，特别是整数数组和字符串键/值对。

此示例中的数据模式包含具有以下名称和数据类型的字段：

字段名	数据类型
index	int
name	string
intarray	整数数组
propmap	字符串键/值对映射

当您在Hive表中指定数组字段，必须为集合中的每个项目标识终止符。相似的，您还必须为映射键指定终止符。

1. 创建一个文本文件，从中加载数据：

```
$ vi /tmp/pxf_hive_complex.txt
```

2. 将以下文本添加到 `pxf_hive_complex.txt`。此数据使用逗号 `,` 分隔字段值，百分号 `%` 分隔集合项，并使用 `:` 终止映射键值：

```
3,Prague,1%2%3,zone:euro%status:up
89,Rome,4%5%6,zone:euro
400,Bangalore,7%8%9,zone:apac%status:pending
183,Beijing,0%1%2,zone:apac
94,Sacramento,3%4%5,zone:noam%status:down
101,Paris,6%7%8,zone:euro%status:up
56,Frankfurt,9%0%1,zone:euro
202,Jakarta,2%3%4,zone:apac%status:up
313,Sydney,5%6%7,zone:apac%status:pending
76,Atlanta,8%9%0,zone:noam%status:down
```

3. 创建一个Hive表来展示此数据：

```
$ HADOOP_USER_NAME=hdfs hive
```

```
hive> CREATE TABLE table_complextypes( index int, name
string, intarray ARRAY<int>, propmap MAP<string, string>
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
COLLECTION ITEMS TERMINATED BY '%'
MAP KEYS TERMINATED BY ':'
STORED AS TEXTFILE;
```

注意:

- FIELDS TERMINATED BY 将逗号标识为字段分隔符
- COLLECTION ITEMS TERMINATED BY 子句将百分号作为集合项(数组项, 映射键/值对)的终止符
- MAP KEYS TERMINATED BY 将冒号标识为映射键的终止符

4. 将 pxf_hive_complex.txt 示例数据文件加载到您刚刚创建的 table_complextypes 表中:

```
hive> LOAD DATA LOCAL INPATH '/tmp/pxf_hive_complex.txt' INTO
TABLE table_complextypes;
```

5. 在Hive表 table_complextypes 上执行查询以验证数据是否已成功加载:

```
hive> SELECT * FROM table_complextypes;
```

```
3 Prague [1,2,3] {"zone":"euro","status":"up"}
89 Rome [4,5,6] {"zone":"euro"}
400 Bangalore [7,8,9] {"zone":"apac","status":"pending"}
...
```

6. 使用 Hive 配置文件创建可读Greenplum数据库外部表, 该表引用名为 table_complextypes 的Hive表:

```
postgres=# CREATE EXTERNAL TABLE
complextypes_hiveprofile(index int, name text, intarray text,
propmap text)
LOCATION ('pxf://table_complextypes?
PROFILE=Hive')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

注意整数数组和映射复杂类型已映射为Greenplum数据库文本数据类型

7. 查询外部表:

```
postgres=# SELECT * FROM complextypes_hiveprofile;
```

index	name	intarray	propmap
3	Prague	[1,2,3]	{"zone": "euro", "status": "up"}
89	Rome	[4,5,6]	{"zone": "euro"}
400	Bangalore	[7,8,9]	{"zone": "apac", "status": "pending"}
183	Beijing	[0,1,2]	{"zone": "apac"}
94	Sacramento	[3,4,5]	{"zone": "noam", "status": "down"}
101	Paris	[6,7,8]	{"zone": "euro", "status": "up"}
56	Frankfurt	[9,0,1]	{"zone": "euro"}
202	Jakarta	[2,3,4]	{"zone": "apac", "status": "up"}
313	Sydney	[5,6,7]	{"zone": "apac", "status": "pending"}
76	Atlanta	[8,9,0]	{"zone": "noam", "status": "down"}
(10 rows)			

`intarray` 和 `propmap` 都被序列化为文本字符串

示例: 使用HiveORC配置文件处理复杂数据类型

在以下示例中，您将创建和填充以ORC格式存储的Hive表。您将使用 `HiveORC` 配置文件查询此Hive表中的复杂数据类型。

1. 用ORC存储格式创建Hive表:

```
$ HADOOP_USER_NAME=hdfs hive
```

```
hive> CREATE TABLE table_complextypes_ORC( index int, name
string, intarray ARRAY<int>, propmap MAP<string, string>)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
COLLECTION ITEMS TERMINATED BY '%'
MAP KEYS TERMINATED BY ':'
STORED AS ORC;
```

2. 将您在此前示例中创建的 `table_complextypes` 表的数据写入到 `table_complextypes_ORC` 表中:

```
hive> INSERT INTO TABLE table_complextypes_ORC SELECT * FROM
table_complextypes;
```

A copy of the sample data set is now stored in ORC format in
`table_complextypes_ORC`.

- 在 `table_complextypes_ORC` 上执行查询以验证数据是否成功加载:

```
hive> SELECT * FROM table_complextypes_ORC;
```

```
OK
3      Prague      [1,2,3]      {"zone":"euro","status":"up"}
89     Rome        [4,5,6]      {"zone":"euro"}
400    Bangalore   [7,8,9]
{"zone":"apac","status":"pending"}
...
...
```

- 启动 `psql` 子系统:

```
$ psql -d postgres
```

- 使用 PXF `HiveORC` 配置文件创建可读Greenplum数据库外部表, 该表引用您在步骤 1 中创建的名为 `table_complextypes_ORC` 的Hive表。 `FORMAT` 子句必须指定为 `'CUSTOM'`。`HiveORC CUSTOM` 格式仅支持内置的 `'pxfwritable_import'` `formatter`。

```
postgres=> CREATE EXTERNAL TABLE complextypes_hiveorc(index
int, name text, intarray text, propmap text)
LOCATION ('pxf://default.table_complextypes_ORC?
PROFILE=HiveORC')
FORMAT 'CUSTOM'
(FORMATTER='pxfwritable_import');
```

注意整数数组和映射复杂类型已映射为Greenplum数据库文本数据类型

- 查询外部表:

```
postgres=> SELECT * FROM complextypes_hiveorc;
```

index	name	intarray	propmap
-------	------	----------	---------

```

-----+
   3 | Prague      | [1,2,3]  |
{"zone":"euro","status":"up"}
   89 | Rome        | [4,5,6]  | {"zone":"euro"}
  400 | Bangalore   | [7,8,9]  |
{"zone":"apac","status":"pending"}
...

```

`intarray` 和 `propmap` 都被序列化为文本字符串

分区过滤下推

PXF Hive连接器支持Hive分区修剪和Hive分区目录结构。这样可以在包含Hive表的选定HDFS文件上排除分区。要使用分区筛选功能来减少网络流量和I/O，请使用 `WHERE` 子句在PXF外部表上运行查询，该子句引用已分区的Hive表中的特定分区列。

下面介绍了对Hive字符串和整数类型的PXF Hive连接器分区过滤支持：

- 字符串类型支持关系运算符 `=`, `<<`, `<=`, `>`, `>=` 和 `<>`。
- 整数类型支持关系运算符 `=` 和 `<>` (要对Hive整数类型使用分区过滤，必须按照[前提条件](#)中所述更新Hive配置)。
- 与上述关系运算符一起使用时，支持逻辑运算符 `AND` 和 `OR`。
- 不支持 `LIKE` 字符串运算符。

要利用PXF分区过滤下推功能，Hive和PXF分区字段名称必须相同。否则，PXF将忽略分区过滤，并且过滤将在Greenplum数据库端执行，从而影响性能。

PXF Hive连接器仅在分区列上过滤，而不在其他表属性上过滤。此外，仅以上确定的那些数据类型和运算符支持过滤器下推。

默认情况下，PXF过滤器下推处于启用状态。您可以按照如下所述配置PXF过滤器下推[关于过滤器下推](#)。

示例：使用Hive配置文件访问同构分区的数据

在此示例中，您将使用 `Hive` 配置文件来查询名为 `sales_part` 的Hive表，该表使用 `delivery_state` 和 `delivery_city` 字段分

区。然后，您创建一个Greenplum数据库外部表以查询 `sales_part`。该过程包括一些演示过滤器下推的特定示例。

1. 创建一个名为 `sales_part` 的Hive表，包含两个分区字段，`delivery_state` 和 `delivery_city`：

```
hive> CREATE TABLE sales_part (name string, type string,
    supplier_key int, price double)
        PARTITIONED BY (delivery_state string, delivery_city
    string)
        ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

2. 加载数据至Hive表并添加一些分区：

```
hive> INSERT INTO TABLE sales_part
    PARTITION(delivery_state = 'CALIFORNIA',
    delivery_city = 'Fresno')
    VALUES ('block', 'widget', 33, 15.17);
hive> INSERT INTO TABLE sales_part
    PARTITION(delivery_state = 'CALIFORNIA',
    delivery_city = 'Sacramento')
    VALUES ('cube', 'widget', 11, 1.17);
hive> INSERT INTO TABLE sales_part
    PARTITION(delivery_state = 'NEVADA', delivery_city =
    'Reno')
    VALUES ('dowel', 'widget', 51, 31.82);
hive> INSERT INTO TABLE sales_part
    PARTITION(delivery_state = 'NEVADA', delivery_city =
    'Las Vegas')
    VALUES ('px49', 'pipe', 52, 99.82);
```

3. 查询 `sales_part` 表：

```
hive> SELECT * FROM sales_part;
```

Hive分区表上的 `SELECT *` 语句显示记录末尾的分区字段。

4. 检查 `sales_part` 表Hive/HDFS的目录结构：

```
$ sudo -u hdfs hdfs dfs -ls -R /apps/hive/warehouse/sales_part
/apps/hive/warehouse/sales_part/delivery_state=CALIFORNIA/deli
/apps/hive/warehouse/sales_part/delivery_state=CALIFORNIA/deli
/apps/hive/warehouse/sales_part/delivery_state=NEVADA/deli
/apps/hive/warehouse/sales_part/delivery_state=NEVADA/deli
```

5. 创建一个PXF外部表来读取Hive分区表 `sales_part`。要利用分区过滤器的下推功能，请在 `CREATE EXTERNAL TABLE` 属性列表的末

尾定义与Hive分区字段相同的字段。

```
$ psql -d postgres
```

```
postgres=# CREATE EXTERNAL TABLE pxf_sales_part(
    item_name TEXT, item_type TEXT,
    supplier_key INTEGER, item_price DOUBLE
PRECISION,
    delivery_state TEXT, delivery_city TEXT)
LOCATION ('pxf://sales_part?Profile=Hive')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

6. 查询外部表:

```
postgres=# SELECT * FROM pxf_sales_part;
```

7. 在 `pxf_sales_part` 上执行另一个查询(不下推), 以此返回 `delivery_city` 等于 `Sacramento` 和 `item_name` 等于 `cube` 的记录:

```
postgres=# SELECT * FROM pxf_sales_part WHERE delivery_city =
'Sacramento' AND item_name = 'cube';
```

这个查询过滤了 `delivery_city` 分区 `Sacramento`。由于 `item_name` 字段不是分区列, 因此该列上的过滤不会被下推。在 `Sacramento` 分区传输完成后, 在Greenplum数据库端执行此过滤操作。

8. 查询(使用过滤下推)所有 `delivery_state` 等于 `CALIFORNIA` 的记录:

```
postgres=# SET gp_external_enable_filter_pushdown=on;
postgres=# SELECT * FROM pxf_sales_part WHERE delivery_state =
'CALIFORNIA';
```

该查询读取位于 `CALIFORNIA` `delivery_state` 分区中的所有数据, 无论城市是哪个。

示例: 使用Hive配置文件访问异构分区的数据

您可以使用PXF `Hive` 配置文件访问任何Hive文件存储类型。使用 `Hive` 配置文件, 您可以在单个Hive表中访问分区以不同文件格式存

储的异构格式数据。

本示例中，您将创建一个分区的Hive外部表。该表由之前示例创建的 `sales_info` (文本格式) 和 `sales_info_rcfile` (RC 格式) Hive表相关的HDFS数据文件组成。您将按照年份对数据进行分区，将 `sales_info` 的数据分配给2013年，而 `sales_info_rcfile` 的数据分配给2016年(这里忽略表包含相同数据的事实)。然后，您将使用PXF `Hive` 配置文件查询这个分区的Hive外部表。

1. 创建一个名为 `hive_multiformpart` 的Hive外部表，该表以名为 `year` 的字符串字段进行分区：

```
$ HADOOP_USER_NAME=hdfs hive
```

```
hive> CREATE EXTERNAL TABLE hive_multiformpart( location
string, month string, number_of_orders int, total_sales
double)
PARTITIONED BY( year string )
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

2. 描述 `sales_info` 和 `sales_info_rcfile` 表，注意每个表HDFS文件的 `location` 信息：

```
hive> DESCRIBE EXTENDED sales_info;
hive> DESCRIBE EXTENDED sales_info_rcfile;
```

3. 在 `hive_multiformpart` 表中为 `sales_info` 以及 `sales_info_rcfile` 表关联的HDFS文件位置创建分区：

```
hive> ALTER TABLE hive_multiformpart ADD PARTITION (year =
'2013') LOCATION
'hdfs://namenode:8020/apps/hive/warehouse/sales_info';
hive> ALTER TABLE hive_multiformpart ADD PARTITION (year =
'2016') LOCATION
'hdfs://namenode:8020/apps/hive/warehouse/sales_info_rcfile';
```

4. 明确的标识与 `sales_info_rcfile` 表关联的分区的文件格式：

```
hive> ALTER TABLE hive_multiformpart PARTITION (year='2016')
SET FILEFORMAT RCFILE;
```

无需指定与 `sales_info` 表相关联分区的文件格式，因为 `TEXTFILE` 格式是默认格式。

5. 查询 `hive_multiformpart` 表:

```
hive> SELECT * from hive_multiformpart;
...
Bangalore Jul 271 8320.55 2016
Beijing Dec 100 4248.41 2016
Prague Jan 101 4875.33 2013
Rome Mar 87 1557.39 2013
...
hive> SELECT * from hive_multiformpart WHERE year='2013';
hive> SELECT * from hive_multiformpart WHERE year='2016';
```

6. 查询 `hive_multiformpart` 表的分区定义并退出 `hive`:

```
hive> SHOW PARTITIONS hive_multiformpart;
year=2013
year=2016
hive> quit;
```

7. 启动 `psql` 子系统:

```
$ psql -d postgres
```

8. 使用PXF `Hive` 配置文件创建一个可读的Greenplum数据库外部表，该表引用此前步骤您在Hive中创建的 `hive_multiformpart` 外部表。

```
postgres=# CREATE EXTERNAL TABLE pxf_multiformpart(location
text, month text, num_orders int, total_sales float8, year
text)
LOCATION ('pxf://default.hive_multiformpart?
PROFILE=Hive')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

9. 查询PXF外部表:

```
postgres=# SELECT * FROM pxf_multiformpart;
```

location	month	num_orders	total_sales	year
Prague	Dec	333	9894.77	2013
Bangalore	Jul	271	8320.55	2013
Beijing	Dec	100	4248.41	2013
Prague	Jan	101	4875.33	2016
Rome	Mar	87	1557.39	2016
Bangalore	May	317	8936.99	2016

....

10. 执行第二个查询以计算2013年的订单总数：

```
postgres=# SELECT sum(num_orders) FROM pxf_multiformpart
WHERE month='Dec' AND year='2013';
sum
-----
433
```

使用PXF访问Hive默认分区

本主题描述了当Hive使用默认分区时，Hive和PXF在查询结果上的区别。当Hive启用动态分区后，分区表可能会将数据存储在默认分区中。当分区列的值与列的定义类型不匹配时（例如，当任何分区列存在NULL值时），Hive会创建一个默认分区。在Hive中，任何在分区列上的过滤查询都会排除默认分区中存储的数据。

与Hive类似，PXF将表的分区列表示为附加在表末尾的列。但是，PXF会将默认分区中的任何列值都转换为NULL值。这意味着在同一个Hive查询中，在分区列上包含 `IS NULL` 过滤器的Greenplum数据库查询可以返回不同结果。

考虑使用以下语句创建的Hive分区表：

```
hive> CREATE TABLE sales (order_id bigint, order_amount float)
PARTITIONED BY (xdate date);
```

该表加载了包含以下内容的五行数据：

1.0	1900-01-01
2.2	1994-04-14
3.3	2011-03-31
4.5	NULL
5.0	2013-12-06

插入第4行会创建一个Hive默认分区，因为分区列 `xdate` 包含NULL值。

在Hive中，对分区列进行筛选的任何查询都将忽略默认分区中的数据。例如，以下查询不返回任何行：

```
hive> SELECT * FROM sales WHERE xdate IS null;
```

但是，如果将此Hive表映射到Greenplum数据库中的PXF外部表，则所有默认分区列的值都将被转换为实际的NULL值。在Greenplum数据库中，对PXF外部表执行相同的查询将返回第4行作为结果，因为过滤器匹配NULL值。

在Hive分区表上执行 `IS NULL` 查询时，请牢记此行为。

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ PXF介绍

□ PXF管理手册

□ 使用PXF访问hadoop

读写文本数据

读取Avro数据

读取JSON数据

读写Parquet数据

读写SequenceFile数据

将多行文本文件读入单个表行

读取Hive表数据

读取HBase表数据

□ 使用PXF访问Azure, Google云端存储, Minio和S3对象存储

使用PXF访问SQL数据库(JDBC)

PXF故障排除

□ PXF实用程序手册

Back to the Administrator Guide

Greenplum database 5管理员指南

In this topic:

- [先决条件](#)
- [HBase入门](#)
- [HBase Shell](#)
- [查询外部HBase数据](#)
- [数据类型映射](#)
- [列映射](#)
- [行键](#)

Apache HBase是Hadoop上的分布式、版本化的非关系型数据库。

PXF HBase连接器读取存储在HBase表中的数据。HBase连接器支持过滤器下推。

本节介绍如何使用PXF HBase连接器

先决条件

使用HBase表数据前, 请确保您已满足:

- 将 `$GPHOME/pxf/lib/pxf-hbase-*.jar` 复制到HBase集群中的每个节点, 并且该PXF JAR文件位于 `$HBASE_CLASSPATH` 中。PXF HBase 连接器需要此配置才能支持过滤器下推。
- 满足PXF Hadoop [先决条件](#).

HBase入门

本主题假设您对以下HBase概念有基本的了解:

- HBase 列包含两个组件: 列簇和列限定词。这些组件由冒号 `:` 分隔, `<column-family>:<column-qualifier>`
- HBase 行由一个行键和一个或多个列值组成。行键是表行的唯一标识符
- HBase 表是由具有一个或多个列的数据行组成的多维映射。创建HBase表时, 可以指定一组完整的列簇
- HBase 单元由行(列簇, 列限定词, 列值)和时间戳组成。给定单元格

中的列值和时间戳表示该值的版本

有关HBase的详细信息, 请参阅[Apache HBase Reference Guide](#) 。

HBase Shell

HBase shell 是一个类似 `psql` 的子系统。要启动HBase shell:

```
$ hbase shell
<hbase output>
hbase(main):001:0>
```

HBase默认的命名空间为 `default`。

示例: 创建一个HBase表

创建一个示例HBase表。

- 在 `default` 命名空间中创建一个名为 `order_info` 的HBase表。
`order_info` 具有两个列簇: `product` 和 `shipping_info` :

```
hbase(main):> create 'order_info', 'product', 'shipping_info'
```

- `order_info` `product` 列簇具有名为 `name` 和 `location` 的列标识符。`shipping_info` 列簇具有名为 `state` 和 `zipcode` 的列标识符。将一些数据添加到 `order_info` 表中:

```
put 'order_info', '1', 'product:name', 'tennis racquet'
put 'order_info', '1', 'product:location', 'out of stock'
put 'order_info', '1', 'shipping_info:state', 'CA'
put 'order_info', '1', 'shipping_info:zipcode', '12345'
put 'order_info', '2', 'product:name', 'soccer ball'
put 'order_info', '2', 'product:location', 'on floor'
put 'order_info', '2', 'shipping_info:state', 'CO'
put 'order_info', '2', 'shipping_info:zipcode', '56789'
put 'order_info', '3', 'product:name', 'snorkel set'
put 'order_info', '3', 'product:location', 'warehouse'
put 'order_info', '3', 'shipping_info:state', 'OH'
put 'order_info', '3', 'shipping_info:zipcode', '34567'
```

在本主题后面的示例中, 您将通过PXF直接访问 `orders_info` HBase 表。

3. 显示 `order_info` 表的内容：

```
hbase(main):> scan 'order_info'
ROW      COLUMN+CELL
 1      column=product:location, timestamp=1499074825516,
value=out of stock
 1      column=product:name, timestamp=1499074825491,
value=tennis racquet
 1      column=shipping_info:state, timestamp=1499074825531,
value=CA
 1      column=shipping_info:zipcode,
timestamp=1499074825548, value=12345
 2      column=product:location, timestamp=1499074825573,
value=on floor
...
3 row(s) in 0.0400 seconds
```

查询外部HBase数据

PXF HBase 连接器支持一个名为的 `HBase` 的配置文件。

使用以下语法创建引用HBase表的Greenplum数据库外部表：

```
CREATE EXTERNAL TABLE <table_name>
  ( <column_name> <data_type> [, ...] | LIKE <other_table> )
LOCATION ('pxf://<hbase-table-name>?PROFILE=HBase')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

`CREATE EXTERNAL TABLE` 命令中HBase连接器使用的特定关键字和值见下表描述。

关键字	值
<hbase-table-name>	HBase表的名称
PROFILE	<code>PROFILE</code> 关键字必须指定为 <code>HBase</code>
SERVER=<server_name>	PXF用于访问数据的命名服务器配置。可选的；如果未指定，PXF将使用 <code>default</code> 服务器。
FORMAT	<code>FORMAT</code> 子句必须指定为 <code>'CUSTOM'</code> <code>(FORMATTER='pxfwritable_import')</code>

数据类型映射

HBase是基于字节的; 它将所有数据类型存储为字节数组。要在Greenplum数据库中表示HBase数据, 请为Greenplum数据库列选择与HBase列标识符值的底层内容匹配的数据类型。

注意: PXF不支持复杂HBase对象

列映射

您可以创建一个引用HBase表定义中全部或部分列标识符的Greenplum数据库外部表。PXF支持Greenplum数据库表列和HBase表列标识符之间的直接或间接映射。

直接映射

当您使用直接映射将Greenplum数据库外部表的列名映射到HBase标识符时, 您可以将列簇限定的HBase标识符名称指定为带引号的值。PXF HBase连接器在读取表数据时将这些列名按原样传递给HBase。

例如, 创建一个访问以下数据的Greenplum数据库外部表:

- 列簇 `product` 中的 `name` 标识符
- 列簇 `shipping_info` 中的 `zipcode` 标识符

从此前您在[示例: 创建一个HBase表](#) 创建的 `order_info` HBase 表中访问数据, 使用以下 `CREATE EXTERNAL TABLE` 语法:

```
CREATE EXTERNAL TABLE orderinfo_hbase ("product:name" varchar,
                                         "shipping_info:zipcode" int)
    LOCATION ('pxf://order_info?PROFILE=HBase')
    FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

通过查找表间接映射

当您使用间接映射将Greenplum数据库外部表列名映射到HBase标识符时, 您可以在HBase中创建的查找表中指定映射。查找表将 `<column-family>:<column-qualifier>` 映射到在创建Greenplum数据库外部表时指

定的列名别名。

您必须将HBase PXF查找表命名为 `pxflookup`。并且您必须使用一个名为 `mapping` 的单列簇定义此表。例如：

```
hbase(main):> create 'pxflookup', 'mapping'
```

尽管直接映射既快速又直观，使用间接映射可以让您为HBase `<column-family>:<column-qualifier>` 名称创建一个较短的，基于字符的别名。由于以下原因，这可以更好的使HBase标识符与Greenplum数据库保持一致：

- HBase 标识符可能非常长。Greenplum数据库列名大小限制为63各字符。
- HBase 标识符名称可以包含二进制或不可打印的字符。Greenplum数据库列名称是基于字符的。

填充 `pxflookup` HBase表时，向表中添加行，以便：

- 行键指定为HBase表名
- `mapping` 列簇标识符定义为Greenplum数据库列名，值标识为需要创建别名的HBase `<column-family>:<column-qualifier>`

例如，要在 `order_info` 表上使用间接映射，将以下条目添加到 `pxflookup` 表：

```
hbase(main):> put 'pxflookup', 'order_info', 'mapping:pname',  
'product:name'  
hbase(main):> put 'pxflookup', 'order_info', 'mapping:zip',  
'shipping_info:zipcode'
```

使用以下 `CREATE EXTERNAL TABLE` 语法创建Greenplum数据库外部表

```
CREATE EXTERNAL TABLE orderinfo_map (pname varchar, zip int)  
LOCATION ('pxf://order_info?PROFILE=HBase')  
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

行键

HBase 表行键是表行的唯一标识符。PXF 以特殊方式处理行键。

要在Greenplum数据库外部表查询中使用行键，请使用名为 `recordkey` 的PXF保留列定义外部表。`recordkey` 列名指示PXF返回每一行的HBase表记录键。

使用Greenplum数据库数据类型 `bytea` 定义 `recordkey`

例如：

```
CREATE EXTERNAL TABLE <table_name> (recordkey bytea, ... )
  LOCATION ('pxf://<hbase_table_name>?PROFILE=HBase')
  FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

在您创建外部表之后，您可以在 `WHERE` 子句中使用 `recordkey` 在一系列键值上过滤HBase表

注意：若要在 `recordkey` 上启用过滤器下推，请将字段定义为 `text`

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

PXF介绍

PXF管理手册

使用PXF访问hadoop

使用PXF访问Azure, Google云端存储, Minio和S3对象存储

About Accessing the S3 Object Store

读取和写入文本数据

读取Avro数据

读取JSON数据

读写Parquet数据

读写SequenceFile数据

将多行文本文件读入单个表行

使用S3 Select从S3读取CSV和Parquet数据

使用PXF访问SQL数据库(JDBC)

PXF故障排除

PXF实用程序手册

Back to the Administrator Guide

Greenplum database 5管理手册

In this topic:

- [先决条件](#)
- [读取文本数据](#)
 - [示例: 从S3读取文本数据](#)
- [读取含有双引号引起的换行符的文本数据](#)
 - [示例: 从S3中读取多行文本数据](#)
- [写入文本数据](#)
 - [示例: 将文本数据写入S3](#)

PXF 对象存储连接器支持普通分隔和逗号分隔的值文本数据。本节介绍如何使用PXF访问对象存储中的文本数据，包括如何创建、查询以及将数据写入引用对象存储中文件的外部表。

注意: 从对象存储中访问文本数据和访问HDFS中的文本数据非常相似。

先决条件

在尝试从对象存储中读取数据或将数据写入对象存储之前，请确保已满足PXF对象存储[先决条件](#)。

读取文本数据

当您从对象存储中读取每行都是一条记录的分隔的纯文本或.csv数据时，请使用 `<objstore>:text` 配置文件。PXF 支持以下 `<objstore>` 配置文件前缀：

对象存储	配置前缀
Azure Blob Storage	wasbs
Azure Data Lake	adl
Google Cloud Storage	gs
Minio	s3
S3	s3

以下语句创建一个引用对象存储中示例文本文件的Greenplum数据库可读外部表：

```
CREATE EXTERNAL TABLE <table_name>
  ( <column_name> <data_type> [, ...] | LIKE <other\_table> )
LOCATION ('pxf://<path-to-file>?PROFILE=<objstore>:text[&SERVER=<server_name>][&<custom-option>=<value>[...]]')
FORMAT '[TEXT|CSV]' (delimiter[=|<space>][E]<delim_value>');
```

`CREATE EXTERNAL TABLE` 命令中使用的特定关键字和值见下表中描述。

关键字	值
<path-to-file>	S3 对象存储中文件或目录的绝对路径
PROFILE= <objstore>:text	<code>PROFILE</code> 关键字必须定义为特定的对象存储。例如: <code>s3:text</code>
SERVER= <server_name>	PXF用于访问数据的命名服务器配置。可选的；如果未指定，PXF将使用 <code>default</code> 服务器。
FORMAT	当 <path-to-file> 引用分隔的纯文本数据时，请使用 <code>FORMAT 'TEXT'</code> 。 当 <path-to-file> 引用逗号分隔的值数据时，请使用 <code>FORMAT 'CSV'</code>
delimiter	数据中的分隔符。对于 <code>FORMAT 'CSV'</code> ，默认的<delim_value> 是逗号 <code>,</code> 。当分隔符为转义序列时，在<delim_value> 前面加上 <code>E</code> 。示例： <code>(delimiter=E'\t')</code> <code>, (delimiter ':')</code> 。

如果要访问S3对象存储库：

- 您可以通过 `CREATE EXTERNAL TABLE` 命令中的自定义选项提供S3凭据，如[使用DDL覆盖S3服务器配置](#)中所述。
- 如果您正在从S3中读取CSV格式的数据，则可以指示PXF使用S3 Select Amazon服务检索数据。有关用于此目的的PXF自定义选项的更多信息，请参考[使用Amazon S3选择服务](#)。

示例：从S3读取文本数据

执行以下步骤创建示例文本文件，将文件复制到S3，然后使用 `s3:text` 配置文件创建两个PXF外部表来查询数据。

要运行此示例，您必须：

- 在系统上安装了AWS CLI工具
- 知道您的AWS访问ID和密钥
- 拥有对S3存储桶的写权限

- 在S3上为PXF示例文件创建一个文件夹。例如，假设您对名为 `BUCKET` 的S3存储桶具有写权限：

```
$ aws s3 mb s3://BUCKET/pxf_examples
```

- 在本地创建一个名为 `pxf_s3_simple.txt` 的分隔的纯文本数据文件：

```
$ echo 'Prague,Jan,101,4875.33
Rome,Mar,87,1557.39
Bangalore,May,317,8936.99
Beijing,Jul,411,11600.67' > /tmp/pxf_s3_simple.txt
```

请注意使用逗号 `,` 分隔四个字段

- 将数据文件复制到您在步骤1创建的S3 目录中：

```
$ aws s3 cp /tmp/pxf_s3_simple.txt s3://BUCKET/pxf_examples/
```

- 验证文件现在位于S3中：

```
$ aws s3 ls s3://BUCKET/pxf_examples/pxf_s3_simple.txt
```

- 启动 `psql` 子系统：

```
$ psql -d postgres
```

- 使用 PXF `s3:text` 配置文件创建Greenplum数据库外部表，该表引用您刚刚创建并添加到S3中的 `pxf_s3_simple.txt` 文件。例如，假设您的服务名为 `s3srvcfg`：

```
postgres=# CREATE EXTERNAL TABLE pxf_s3_textsimple(location text, month
text, num_orders int, total_sales float8)
  LOCATION ('pxf://BUCKET/pxf_examples/pxf_s3_simple.txt?
PROFILE=s3:text&SERVER=s3srvcfg')
  FORMAT 'TEXT' (delimiter=E',');
```

- 查询外部表：

```
postgres=# SELECT * FROM pxf_s3_textsimple;
```

location	month	num_orders	total_sales
Prague	Jan	101	4875.33
Rome	Mar	87	1557.39
Bangalore	May	317	8936.99
Beijing	Jul	411	11600.67
(4 rows)			

- 创建另一个引用 `pxf_s3_simple.txt` 的外部表，这次指定 `CSV` `FORMAT`：

```
postgres=# CREATE EXTERNAL TABLE pxf_s3_textsimple_csv(location text,
month text, num_orders int, total_sales float8)
  LOCATION ('pxf://BUCKET/pxf_examples/pxf_s3_simple.txt?
PROFILE=s3:text&SERVER=s3srvcfg')
  FORMAT 'CSV';
postgres=# SELECT * FROM pxf_s3_textsimple_csv;
```

当您为逗号分隔值数据指定 `FORMAT 'CSV'` 时，不需要提供 `delimiter` 分隔符选项，因为逗号是默认的分隔符。

读取含有双引号引起的换行符的文本数据

使用 `<objstore>:text:multi` 配置文件读取数据中含有引号引起的换行符，单行或多行的分隔文本数据。以下语法创建一个引用对象存储中此类文本文件的Greenplum数据库可读外部表：

```
CREATE EXTERNAL TABLE <table_name>
  ( <column_name> <data_type> [, ...] | LIKE <other_table> )
LOCATION ('pxf://<path-to-file>?PROFILE=<objstore>:text:multi[&SERVER=
<server_name>][&<custom-option>=<value>[...]]')
FORMAT '[TEXT|CSV]' (delimiter[=|<space>][E]<delim_value>);
```

`CREATE EXTERNAL TABLE` 命令中使用的特定关键字和值见下表中描述。

关键字	值
<code><path-to-file></code>	S3数据存储中的目录或文件的绝对路径
<code>PROFILE=</code> <code><objstore>:text:multi</code>	<code>PROFILE</code> 关键字必须指定为特定的对象存储。例如： <code>s3:text:multi</code>
<code>SERVER=</code> <code><server_name></code>	PXF用于访问数据的命名服务器配置。可选的；如果未指定，PXF将使用 <code>default</code> 服务器。
<code>FORMAT</code>	当 <code><path-to-hdfs-file></code> 引用分隔的纯文本数据时，请使用 <code>FORMAT 'TEXT'</code> 。 当 <code><path-to-hdfs-file></code> 引用逗号分隔值数据时，请使用 <code>FORMAT 'CSV'</code> 。
<code>delimiter</code>	数据中的分隔符。对于 <code>FORMAT 'CSV'</code> ，默认的 <code><delim_value></code> 是逗号 <code>,</code> 。当分隔符为转义序列时，在 <code><delim_value></code> 前面加上 <code>E</code> 。示例： <code>(delimiter=E'\t')</code> , <code>(delimiter ':')</code> 。

如果要访问S3对象存储，则可以通过 `CREATE EXTERNAL TABLE` 命令中的自定义选项提供S3凭据，如[使用DDL覆盖S3服务器配置](#)中所述。

示例：从S3中读取多行文本数据

执行以下步骤创建一个样例文本文件、将文件复制到S3，并使用 `PROFILE`
`s3:text:multi` 配置文件创建一个Greenplum数据库可读外部表来查询数据。

要运行此示例，您必须：

- 在系统上安装了AWS CLI工具
- 知道您的AWS访问ID和密钥
- 拥有对S3存储桶的写权限

1. 创建第二个带分隔符的纯文本文件:

```
$ vi /tmp/pxf_s3_multi.txt
```

2. 将以下数据复制/黏贴到 `pxf_s3_multi.txt` 中:

```
"4627 Star Rd.  
San Francisco, CA 94107":Sept:2017  
"113 Moon St.  
San Diego, CA 92093":Jan:2018  
"51 Belt Ct.  
Denver, CO 90123":Dec:2016  
"93114 Radial Rd.  
Chicago, IL 60605":Jul:2017  
"7301 Brookview Ave.  
Columbus, OH 43213":Dec:2018
```

注意使用冒号 `:` 分隔三个字段。另外还要注意第一个字段(地址)周围的引号。此字段包含嵌入式换行符，用于将街道地址与城市和州分开。

3. 将文本文件复制到S3:

```
$ aws s3 cp /tmp/pxf_s3_multi.txt s3://BUCKET/pxf_examples/
```

4. 使用 `s3:text:multi` 配置文件创建一个引用S3文件 `pxf_s3_multi.txt` 的外部表，确保定义 `:` (冒号) 作为字段分隔符。例如，假设您的服务名为 `s3srvcfg` :

```
postgres=# CREATE EXTERNAL TABLE pxf_s3_textmulti(address text, month
text, year int)
LOCATION ('pxf://BUCKET/pxf_examples/pxf_s3_multi.txt?
PROFILE=s3:text:multi&SERVER=s3srvcfg')
FORMAT 'CSV' (delimiter ':');
```

注意指定 `delimiter` 的替代语法。

5. 查询 `pxf_s3_textmulti` 表:

```
postgres=# SELECT * FROM pxf_s3_textmulti;
```

address	month	year
4627 Star Rd.	Sept	2017
San Francisco, CA 94107		
113 Moon St.	Jan	2018
San Diego, CA 92093		

51 Belt Ct.	Dec	2016
Denver, CO 90123		
93114 Radial Rd.	Jul	2017
Chicago, IL 60605		
7301 Brookview Ave.	Dec	2018
Columbus, OH 43213		
(5 rows)		

写入文本数据

“<objstore>:text” 配置文件支持将单行纯文本文件写入对象存储中。 使用PXF创建可写外部表时，请指定目录名称。当您将记录写入可写外部表中，您插入的数据块将写入指定目录中的一个或多个文件中。

注意： 使用可写配置文件创建的外部表仅支持 `INSERT` 操作。 如果要查询插入的数据，则比如单独创建一个引用该目录的可读外部表。

使用以下语法创建一个引用对象存储目录的Greenplum数据库可写外部表：

```
CREATE WRITABLE EXTERNAL TABLE <table_name>
  ( <column_name> <data_type> [, ...] | LIKE <other_table> )
LOCATION ('pxf://<path-to-dir>
?PROFILE=<objstore>:text[&SERVER=<server_name>][&<custom-option>=
<value>[...]]')
FORMAT '[TEXT|CSV]' (delimiter[=|<space>][E]<delim_value>');
[DISTRIBUTED BY (<column_name> [, ... ] ) | DISTRIBUTED RANDOMLY];
```

`CREATE EXTERNAL TABLE` 命令中使用的特定关键字和值见下表中描述。

关键字	值
<path-to-dir>	S3数据存储中目录的绝对路径
PROFILE=<objstore>:text	<code>PROFILE</code> 关键字必须指定为对象存储。例如： <code>s3:text</code>
SERVER=<server_name>	PXF用于访问数据的命名服务器配置。可选的；如果未指定，PXF将使用 <code>default</code> 服务器。
<custom-option>=<value>	<custom-option> 选项见下方描述
FORMAT	将分隔的纯文本写入<path-to-dir>时，使用 <code>FORMAT 'TEXT'</code> 。 将逗号分隔值文本写入<path-to-dir>时，使用 <code>FORMAT 'CSV'</code> 。
delimiter	数据中的分隔符。数据中的分隔符。对于 <code>FORMAT 'CSV'</code> ，默认的<delim_value>是逗号 <code>,</code> 。当分隔符为转义序列时，在<delim_value>前面加上 <code>E</code> 。示例： <code>(delimiter=E'\t')</code> ， <code>(delimiter ':')</code> 。
	Greenplum

DISTRIBUTED
BY

如果您计划将现有外部表，考虑在可写外部表上使用与Greenplum表相同的分布策略或<字段名>。这样做可以避免数据加载操作中segment节点间额外的数据移动。.

使用 `<objstore>:text` 配置文件创建可写外部表时，可以选择使用记录或块压缩。PXF `<objstore>:text` 配置文件支持以下压缩编解码器：

- org.apache.hadoop.io.compress.DefaultCodec
- org.apache.hadoop.io.compress.GzipCodec
- org.apache.hadoop.io.compress.BZip2Codec

您可以通过 `CREATE EXTERNAL TABLE` `LOCATION` 子句中的自定义选项指定压缩编码器。`<objstore>:text` 配置文件支持以下自定义写入选项：

选项	值描述
COMPRESSION_CODEC	压缩编解码器Java类名。如果未提供此选项，Greenplum数据库不会执行压缩编码。支持的压缩编解码器包括： <code>org.apache.hadoop.io.compress.DefaultCodec</code> <code>org.apache.hadoop.io.compress.BZip2Codec</code> <code>org.apache.hadoop.io.compress.GzipCodec</code>
COMPRESSION_TYPE	采用的压缩类型；支持的值为 <code>RECORD</code> （默认）或 <code>BLOCK</code>
THREAD-SAFE	确定表查询是否可以在多线程模式下运行的布尔值。默认为 <code>TRUE</code> 。将此选项设置为 <code>FALSE</code> 以处理单个线程中所有非线程安全操作（例如，压缩）的请求。

如果要访问S3对象存储，则可以通过 `CREATE EXTERNAL TABLE` 命令中的自定义选项提供S3凭据，如[使用DDL覆盖S3服务器配置](#)中所述。

示例：将文本数据写入S3

这个示例引用在[示例：从S3读取文本数据](#)中介绍的数据模式。

列名	数据类型
location	text
month	text
number_of_orders	int
total_sales	float8

此示例还可以选择使用您在该练习中创建的名为 `pxf_s3_textsimple` 的Greenplum数据库外部表。

步骤

执行以下步骤，使用与上述相同的数据模式创建Greenplum数据库可写外部表，其中之一将使用压缩。你将使用 PXF `s3:text` 配置文件将数据写入S3。您还将创建一个单独的可读外部表，以读取您写入S3的数据。

1. 使用上述的数据模式创建Greenplum数据库可写外部表。写入S3 目录

`BUCKET/pxf_examples/pxfwrite_s3_textsimple1`。 创建表是指定逗号 `,` 作为分隔符。例如，假设您的服务名称为 `s3srvcfg`：

```
postgres=# CREATE WRITABLE EXTERNAL TABLE pxf_s3_writetbl_1(location
text, month text, num_orders int, total_sales float8)
LOCATION
('pxf://BUCKET/pxf_examples/pxfwrite_s3_textsimple1?
PROFILE=s3:text&SERVER=s3srvcfg')
FORMAT 'TEXT' (delimiter=','');
```

将 `FORMAT` 子句 `delimiter` 值指定为单个ASCII逗号字符 `,`

2. 通过在 `pxf_s3_writetbl_1` 上调用SQL `INSERT` 命令，将一些单独的记录写入 `pxfwrite_s3_textsimple1` S3目录：

```
postgres=# INSERT INTO pxf_s3_writetbl_1 VALUES ( 'Frankfurt', 'Mar',
777, 3956.98 );
postgres=# INSERT INTO pxf_s3_writetbl_1 VALUES ( 'Cleveland', 'Oct',
3812, 96645.37 );
```

3. (可选) 将您在 [示例: 从S3读取文本数据](#) 创建的 `pxf_s3_textsimple` 表中的数据插入到 `pxf_s3_writetbl_1`：

```
postgres=# INSERT INTO pxf_s3_writetbl_1 SELECT * FROM
pxf_s3_textsimple;
```

4. Greenplum数据库不支持直接查询可写外部表。要查询刚刚添加到S3的数据，必须创建一个引用该S3目录的Greenplum数据库可读外部表：

```
postgres=# CREATE EXTERNAL TABLE pxf_s3_textsimple_r1(location text,
month text, num_orders int, total_sales float8)
LOCATION
('pxf://BUCKET/pxf_examples/pxfwrite_s3_textsimple1?
PROFILE=s3:text&SERVER=s3srvcfg')
FORMAT 'CSV';
```

创建可读外部表时，请指定 `'CSV'` `FORMAT`，因为您创建可写外部表时使用逗号 `,` 作为分隔字符，即 `'CSV'` `FORMAT` 默认分隔符。

5. 查询可读外部表：

```
postgres=# SELECT * FROM pxf_s3_textsimple_r1 ORDER BY total_sales;
```

location	month	num_orders	total_sales
Rome	Mar	87	1557.39
Frankfurt	Mar	777	3956.98
Prague	Jan	101	4875.33
Bangalore	May	317	8936.99
Beijing	Jul	411	11600.67
Cleveland	Oct	3812	96645.37
(6 rows)			

`pxf_s3_textsimple_r1` 表包含了您单独插入的记录以及
`pxf_s3_textsimple` 表的完整内容(如果执行了可选步骤)

6. 创建第二个Greenplum数据库可写外部表，这次使用Gzip 压缩并使用冒号 `:` 作为分隔符：

```
postgres=# CREATE WRITABLE EXTERNAL TABLE pxf_s3_writetbl_2 (location text
num_orders int, total_sales float8)
LOCATION ('pxf://BUCKET/pxf_examples/pxfwrite_s3_textsimple2?'
PROFILE=s3:text&SERVER=s3srvcfg&COMPRESSION_CODEC=org.apache.hadoop.io.com
FORMAT 'TEXT' (delimiter=':'');
```

7. 通过直接写入 `pxf_s3_writetbl_2` 表将一些记录写入到 `pxfwrite_s3_textsimple2` S3目录：

```
gpadmin=# INSERT INTO pxf_s3_writetbl_2 VALUES ( 'Frankfurt', 'Mar',
777, 3956.98 );
gpadmin=# INSERT INTO pxf_s3_writetbl_2 VALUES ( 'Cleveland', 'Oct',
3812, 96645.37 );
```

8. 要从新创建的名为 `pxfwrite_s3_textsimple2` 的S3目录查询数据，您可以如上所述创建一个Greenplum数据库可读外部表，该表引用此S3目录并指定 `FORMAT 'CSV' (delimiter=':')`。

[Greenplum数据库® 6.0文档](#)

[Greenplum平台扩展框架\(PXF\)](#)

[PXF介绍](#)

[PXF管理手册](#)

[使用PXF访问hadoop](#)

[使用PXF访问Azure, Google云端存储, Minio和S3对象存储](#)

[About Accessing the S3 Object Store](#)

[读取和写入文本数据](#)

[读取Avro数据](#)

[读取JSON数据](#)

[读写Parquet数据](#)

[读写SequenceFile数据](#)

[将多行文本文件读入单个表行](#)

[使用S3 Select从S3读取CSV和Parquet数据](#)

[使用PXF访问SQL数据库\(JDBC\)](#)

[PXF故障排除](#)

[PXF实用程序手册](#)

[Back to the Administrator Guide](#)

In this topic:

- [先决条件](#)
- [使用Avro数据](#)
- [创建外部表](#)
- [示例](#)

PXF 对象存储连接器支持读取Avro格式数据。本节描述如何使用PXF访问对象存储中的Avro数据，包括创建和查询引用对象存储中Avro文件的外部表。

注意: 从对象存储访问Avro格式数据与访问HDFS中的Avro格式数据非常相似。本主题标识了读取对象存储所需的特定信息，并在通用信息的位置链接到 [PXF HDFS Avro 文档](#)。

先决条件

在您尝试从对象存储中读取数据前，确保已满足PXF 对象存储先决条件。

使用Avro数据

有关Apache Avro数据序列化框架的说明，请参考PXF HDFS Avro文档中的[使用Avro数据](#)

创建外部表

使用 `<objstore>:avro` 配置文件从对象存储中读取Avro格式文件。PXF 支持以下 `<objstore>` 配置前缀：

对象存储	配置前缀
Azure Blob Storage	wasbs
Azure Data Lake	adl
Google Cloud Storage	gs
Minio	s3

S3	s3
----	----

以下语法创建一个引用Avro格式文件的Greenplum数据库可读外部表：

```
CREATE EXTERNAL TABLE <table_name>
  ( <column_name> <data_type> [, ...] | LIKE <other_table> )
LOCATION ('pxf://<path-to-file>?PROFILE=<objstore>:avro[&SERVER=
<server_name>][&<custom-option>=<value>[...]]')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

[CREATE EXTERNAL TABLE](#) 命令中使用的特定关键字和值见下表中描述。

关键字	值
<path-to-file>	对象存储中文件或目录的绝对路径
PROFILE= <objstore>:avro	PROFILE 必须指定为特定的对象存储。例如， s3:avro
SERVER= <server_name>	PXF用于访问数据的命名服务器配置。可选的；如果未指定，PXF将使用 default 服务器。
<custom-option>= <value>	特定于Avro的自定义选项描述见 PXF HDFS Avro 文档
FORMAT 'CUSTOM'	在 <objstore>:avro 配置文件中使用 FORMAT 'CUSTOM'。 CUSTOM FORMAT 要求您指定为 (FORMATTER='pxfwritable_import')。

如果要访问S3对象存储，则可以通过[CREATE EXTERNAL TABLE]命令中的自定义选项提供S3凭据，如[使用DDL覆盖S3服务器配置](#)中所述。

示例

有关Avro示例，请参阅PXF HDFS Avro 文档中的[示例: 读取Avro数据](#)。在对象存储中运行示例，必要的修改如下：

- 将文件复制到对象存储而不是HDFS。例如，将文件复制到S3：

```
$ aws s3 cp /tmp/pxf_avro.avro s3://BUCKET/pxf_examples/
```

- 使用上述的 `CREATE EXTERNAL TABLE` 语法和 `LOCATION` 关键字及

设置。例如，假设您的服务名为 `s3srvcfg`：

```
CREATE EXTERNAL TABLE pxf_s3_avro(id bigint, username text, full  
text, relationship text, address text)  
LOCATION ('pxf://BUCKET/pxf_examples/pxf_avro.avro?  
PROFILE=s3:avro&SERVER=s3srvcfg&COLLECTION_DELIM=,&MAPKEY_DELIM=  
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import'));
```

[Greenplum数据库® 6.0文档](#)

[Greenplum平台扩展框架\(PXF\)](#)

[PXF介绍](#)

[PXF管理手册](#)

[使用PXF访问hadoop](#)

[使用PXF访问Azure, Google云端存储, Minio和S3对象存储](#)

[About Accessing the S3 Object Store](#)

[读取和写入文本数据](#)

[读取Avro数据](#)

[读取JSON数据](#)

[读写Parquet数据](#)

[读写SequenceFile数据](#)

[将多行文本文件读入单个表行](#)

[使用S3 Select从S3读取CSV和Parquet数据](#)

[使用PXF访问SQL数据库\(JDBC\)](#)

[PXF故障排除](#)

[PXF实用程序手册](#)

[Back to the Administrator Guide](#)

In this topic:

- [先决条件](#)
- [使用JSON数据](#)
- [创建外部表](#)
- [示例](#)

PXF 对象存储连接器支持读取JSON格式数据。本节描述如何使用PXF访问对象存储中的JSON数据，包括如何创建和查询引用对象存储中JSON文件的外部表。

注意: 从对象存储访问JSON数据和访问HDFS中的JSON数据非常相似。本主题标识了读取对象存储中JSON数据所需的特定信息，并在通用信息的位置链接到 [PXF HDFS JSON 文档](#)。

先决条件

在您尝试从对象存储中读取数据前，确保已满足PXF 对象存储先决条件。

使用JSON数据

有关基于JSON文本的数据交换格式的说明，请参阅PXF HDFS JSON文档中的 [使用JSON数据](#)。

创建外部表

使用 `<objstore>: json` 配置文件读取对象存储中的JSON格式文件。PXF 支持以下 `<objstore>` 配置前缀：

对象存储	配置前缀
Azure Blob Storage	wasbs
Azure Data Lake	adl
Google Cloud Storage	gs
Minio	s3

S3	s3
----	----

以下语法创建一个引用JSON格式文件的Greenplum数据库可读外部表：

```
CREATE EXTERNAL TABLE <table_name>
  ( <column_name> <data_type> [, ...] | LIKE <other_table> )
LOCATION ('pxf://<path-to-file>?PROFILE=<objstore>:json[&SERVER=
<server_name>][&<custom-option>=<value>[...]]')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

[CREATE EXTERNAL TABLE](#) 命令中使用的特定关键字和值见下表中描述。

关键字	值
<path-to-file>	对象存储中文件或目录的绝对路径
PROFILE= <objstore>:json	PROFILE 必须指定为特定的对象存储。例如： s3:json 。
SERVER= <server_name>	PXF用于访问数据的命名服务器配置。可选的；如果未指定，PXF将使用 default 服务器。
<custom-option>= <value>	JSON 支持的名为 IDENTIFIER 的自定义选项 描述见 PXF HDFS JSON 文档 。
FORMAT 'CUSTOM'	在 <objstore>:json 配置文件中使用 FORMAT 'CUSTOM' 。 CUSTOM FORMAT 要求您指定为 (FORMATTER='pxfwritable_import') 。

如果要访问S3对象存储，则可以如[使用DDL覆盖S3服务器配置](#)中所述直接在 [CREATE EXTERNAL TABLE](#) 命令的自定义选项中提供S3凭据。

示例

有关JSON示例，请参阅PXF HDFS JSON 文档中的 [将样本JSON数据加载到HDFS](#) 以及 [示例：读取单行记录的JSON文件](#)。在对象存储中运行示例，必要的修改如下：

- 将文件复制到对象存储而不是HDFS。例如，将文件复制到S3：

```
$ aws s3 cp /tmp/singleline.json s3://BUCKET/pxf_examples/
$ aws s3 cp /tmp/multiline.json s3://BUCKET/pxf_examples/
```

- 使用上述的 `CREATE EXTERNAL TABLE` 语法和 `LOCATION` 关键字及设置。例如，假设您的服务名为 `s3srvcfg`：

```
CREATE EXTERNAL TABLE singleline_json_s3(
    created_at TEXT,
    id_str TEXT,
    "user.id" INTEGER,
    "user.location" TEXT,
    "coordinates.values[0]" INTEGER,
    "coordinates.values[1]" INTEGER
)
LOCATION('pxf://BUCKET/pxf_examples/singleline.json?
PROFILE=s3:json&SERVER=s3srvcfg')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ PXF介绍

□ PXF管理手册

□ 使用PXF访问hadoop

□ 使用PXF访问Azure, Google云端存储, Minio和S3对象存储

About Accessing the S3 Object Store

读取和写入文本数据

读取Avro数据

读取JSON数据

读写Parquet数据

读写SequenceFile数据

将多行文本文件读入单个表行

使用S3 Select从S3读取CSV和Parquet数据

使用PXF访问SQL数据库(JDBC)

PXF故障排除

□ PXF实用程序手册

Back to the Administrator Guide

In this topic:

- [先决条件](#)
- [数据类型映射](#)
- [创建外部表](#)
- [示例](#)

PXF 对象存储连接器支持读写Parquet格式数据。本节描述了如何使用PXF访问对象存储中的Parquet格式数据，包括创建和查询引用对象存储中Parquet文件的外部表。

PXF Parquet写入支持是一个Beta功能。

注意: 从对象存储访问Parquet格式数据与访问HDFS中的Parquet格式数据非常相似。本主题标识了读取对象存储所需的特定信息，并在通用信息的位置链接到 [PXF HDFS Parquet 文档](#)。

先决条件

在您尝试从对象存储中读取数据前，确保已满足PXF 对象存储先决条件。

数据类型映射

有关Greenplum数据库和Parquet数据类型之间映射的说明，请参阅 [数据类型映射](#)。

创建外部表

PXF `<objstore>:parquet` 配置文件支持读取和写入Parquet格式数据。PXF 支持以下 `<objstore>` 配置前缀：

对象存储	配置前缀
Azure Blob Storage	wasbs
Azure Data Lake	adl

Greenplum database 5管理员

Google Cloud Storage	gs
Minio	s3
S3	s3

使用以下语法创建引用S3目录的Greenplum数据库外部表。当您将记录插入可写外部表中时，您插入的数据块将写入指定目录中一个或多个文件。

```
CREATE [WRITABLE] EXTERNAL TABLE <table_name>
  ( <column_name> <data_type> [, ...] | LIKE <other_table> )
LOCATION ('pxf://<path-to-dir>
?PROFILE=<objstore>:parquet[&SERVER=<server_name>][&<custom-
option>=<value>[...]]')
FORMAT 'CUSTOM'
(FORMATTER='pxfwritable_import' | 'pxfwritable_export');
[DISTRIBUTED BY (<column_name> [, ... ] ) | DISTRIBUTED
RANDOMLY];
```

`CREATE EXTERNAL TABLE` 命令中使用的特定关键字和值见下表中描述。

关键字	值
<path-to-dir>	对象存储中文件或目录的绝对路径
PROFILE=<objstore>:parquet	PROFILE 必须指定为特定的对象存储。例如, s3:parquet
SERVER=<server_name>	PXF用于访问数据的命名服务器配置。可选的; 如果未指定, PXF将使用 default 服务器。
<custom-option>=<value>	特定于Parquet的自定义选项描述见 PXF HDFS Parquet 文档
FORMAT 'CUSTOM'	使用 FORMAT 'CUSTOM' 时指定 (FORMATTER='pxfwritable_export') (写入) 或 (FORMATTER='pxfwritable_import') (读取)
DISTRIBUTED BY	如果您计划将现有Greenplum数据库表中的数据加载到可写外部表, 考虑在可写外部表上使用与Greenplum表相同的分布策略或<字段名>。这样做可以避免数据加载操作中segment节点间额外的数据移动。

如果要访问S3 对象存储, 则可以如 [覆盖S3服务配置](#) 中所述直接在 `CREATE EXTERNAL TABLE` 命令中提供S3凭据。如果要访问S3对象存

储库： - 您可以通过 `CREATE EXTERNAL TABLE` 命令中的自定义选项提供S3凭据，如[用DDL覆盖S3服务器配置](#)中所述。 - 如果您正在从S3读取Parquet数据，则可以指示PXF使用S3 Select Amazon服务检索数据。有关用于此目的的PXF自定义选项的更多信息，请参考[使用Amazon S3选择服务](#)。

示例

有关Parquet 读取/写入的示例，请参阅PXF HDFS Parquet 文档中的[示例](#)。在对象存储中运行示例，必要的修改如下：

- 使用上述的 `CREATE EXTERNAL TABLE` 语法和 `LOCATION` 关键字及设置。例如，假设您的服务名为 `s3srvcfg`：

```
CREATE WRITABLE EXTERNAL TABLE pxf_tbl_parquet_s3 (location
text, month text, number_of_orders int, total_sales double
precision)
LOCATION ('pxf://BUCKET/pxf_examples/pxf_parquet?
PROFILE=s3:parquet&SERVER=s3srvcfg')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_export');
```

- 使用上述的 `CREATE EXTERNAL TABLE` 语法和 `LOCATION` 关键字及设置。例如，假设您的服务名为 `s3srvcfg`：

```
CREATE EXTERNAL TABLE read_pxf_parquet_s3(location text,
month text, number_of_orders int, total_sales double
precision)
LOCATION ('pxf://BUCKET/pxf_examples/pxf_parquet?
PROFILE=s3:parquet&SERVER=s3srvcfg')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

- PXF介绍
- PXF管理手册
- 使用PXF访问hadoop
- 使用PXF访问Azure, Google云端存储, Minio和S3对象存储

About Accessing the S3 Object Store

读取和写入文本数据

读取Avro数据

读取JSON数据

读写Parquet数据

读写SequenceFile数据

将多行文本文件读入单个表行

使用S3 Select从S3读取CSV和Parquet数据

使用PXF访问SQL数据库(JDBC)

PXF故障排除

- PXF实用程序手册

Back to the Administrator Guide

In this topic:

- [先决条件](#)
- [创建外部表](#)
- [示例](#)

PXF 对象存储连接器支持SequenceFile格式二进制数据。本节描述如何使用PXF读写SequenceFile数据，包括如何创建、写入以及查询引用对象存储文件的外部表。

注意: 从对象存储访问SequenceFile格式数据与访问HDFS中的SequenceFile格式数据非常相似。本主题标识了读取对象存储所需的特定信息，并在通用信息的位置链接到[PXF HDFS SequenceFile 文档](#)

先决条件

在您尝试从对象存储中读取数据前，确保已满足PXF 对象存储先决条件。

创建外部表

PXF `<objstore>:SequenceFile` 配置文件支持以SequenceFile格式读取和写入数据。PXF 支持以下 `<objstore>` 配置文件前缀：

对象存储	配置前缀
Azure Blob Storage	wasbs
Azure Data Lake	adl
Google Cloud Storage	gs
Minio	s3
S3	s3

使用以下语法创建引用S3目录的Greenplum数据库外部表。当您将记录插入可写外部表中时，您插入的数据块将写入指定目录中一个或多个文件。

Greenplum database 5管理员

```
CREATE [WRITABLE] EXTERNAL TABLE <table_name>
  ( <column_name> <data_type> [, ...] | LIKE <other_table> )
LOCATION ('pxf://<path-to-dir>
?PROFILE=<objstore>:SequenceFile[&SERVER=<server_name>][&
<custom-option>=<value>[...]]')
FORMAT 'CUSTOM'
(FORMATTER='pxfwritable_import'|'pxfwritable_export')
[DISTRIBUTED BY (<column_name> [, ... ] ) | DISTRIBUTED
RANDOMLY];
```

`CREATE EXTERNAL TABLE` 命令中使用的特定关键字和值见下表中描述。

关键字	值
<path-to-dir>	对象存储中文件或目录的绝对路径
PROFILE=<objstore>:SequenceFile	<code>PROFILE</code> 必须指定为特定的对象存储。例如， <code>s3:SequenceFile</code>
SERVER=<server_name>	PXF用于访问数据的命名服务器配置。可选的；如果未指定，PXF将使用 <code>default</code> 服务器。
<custom-option>=<value>	特定于SequenceFile的自定义选项描述见 PXF HDFS SequenceFile 文档
FORMAT 'CUSTOM'	使用 <code>FORMAT 'CUSTOM'</code> 时指定 (FORMATTER='pxfwritable_export') (写入) 或 (FORMATTER='pxfwritable_import') (读取)
DISTRIBUTED BY	如果您计划将现有Greenplum数据库表中的数据加载到可写外部表，考虑在可写外部表上使用与Greenplum表相同的分布策略或<字段名>。这样做可以避免数据加载操作中segment节点间额外的数据移动。

如果要访问S3对象存储，则可以如[使用DDL覆盖S3服务器配置](#)中所述直接在 `CREATE EXTERNAL TABLE` 命令的自定义选项中提供S3凭据。

示例

有关SequenceFile读取/写入的示例，请参阅[PXF HDFS SequenceFile 文档](#)中的[示例：将二进制数据写入HDFS](#)。在对象存储中运行示例，必

要的修改如下：

- 使用上述的 `CREATE EXTERNAL TABLE` 语法和 `LOCATION` 关键字及设置。例如，假设您的服务名为 `s3srvcfg`：

```
CREATE WRITABLE EXTERNAL TABLE pxf_tbl_seqfile_s3(location text,  
LOCATION ('pxf://BUCKET/pxf_examples/pxf_seqfile?PROFILE=s3:Se  
SCHEMA=com.example.pxf.hdfs.writable.dataschema.PxfExample_Custo  
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_export'));
```

- 使用上述的 `CREATE EXTERNAL TABLE` 语法和 `LOCATION` 关键字及设置。例如，假设您的服务名为 `s3srvcfg`：

```
CREATE EXTERNAL TABLE read_pxf_tbl_seqfile_s3(location text, mon  
integer, total_sales real)  
LOCATION ('pxf://BUCKET/pxf_examples/pxf_seqfile?PROFILE=s3:Se  
SCHEMA=com.example.pxf.hdfs.writable.dataschema.PxfExample_Custo  
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import'));
```

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ PXF介绍

□ PXF管理手册

□ 配置PXF

PXF安装和配置目录

为PXF安装JAVA环境

初始化PXF

配置PXF服务器

□ 配置PXF HADOOP连接器 (可选)

配置用户模拟和代理

为安全HDFS配置PXF

配置Minio和S3对象存储的连接器 (可选)

配置Azure和Google云端存储的连接器 (可选)

配置JDBC连接器 (可选)

配置PXF客户端主机和端口(可选)

升级PXF

PXF的启动、停止和重启

In this topic:

- [配置PXF用户模拟](#)
- [配置Hadoop代理](#)
- [Hive用户模拟](#)
- [Hbase用户模拟](#)

PXF使用Greenplum数据库最终用户访问Hadoop服务。默认情况下，PXF尝试使用经过GPDB身份验证的用户身份并且通过PXF连接器配置去访问数据源服务(HDFS, Hive, HBase)。注意，PXF在访问Hadoop服务时仅使用用户的登录身份。例如，如果用户以jane身份登录Greenplum数据库，然后执行SET ROLE或SET SESSION AUTHORIZATION来假定其他用户身份，则所有PXF请求仍将使用jane身份来访问Hadoop服务。

使用默认的PXF配置，您必须显式配置每个Hadoop数据源(HDFS, Hive, HBase)来允许PXF进程所有者(通常为 `gpadmin`)充当模拟用户或组的代理。详情参阅[Configuring Hadoop Proxying](#), [Hive User Impersonation](#), and [HBase User Impersonation](#)。

或者，您可以禁用PXF用户模拟功能。禁用用户模拟后，PXF将使用PXF进程的所有者(通常为 `gpadmin`)来执行所有Hadoop服务请求。此行为与PXF的早期版本相匹配，但是它没有提供任何方法来控制Hadoop中不同于Greenplum数据库用户对Hadoop服务的访问。它要求 `gpadmin` 用户有权访问HDFS中的所有文件和目录，以及在PXF外部表定义中引用的Hive和HBase中的所有表的权限。参照[Configuring PXF User Impersonation](#) for information about disabling user impersonation。

配置PXF用户模拟

执行以下流程在你的GPDB集群上打开或者关闭用户模拟功能。如果你第一次配置PXF，默认用户模拟是打开的。你不需要执行这个过程。

1. 以管理员用户身份登录GPDB master节点

```
$ ssh gpadmin@<gpmaster>
```

2. 重新进入PXF用户配置目录 `$PGF_CONF`。用文本编辑器打开配置文

件 `$PXF_CONF/conf/pxf-env.sh`。例如：

```
gpadmin@gpmaster$ vi $PXF_CONF/conf/pxf-env.sh
```

- 在 `pxf-env.sh` 文件中找到 `PXF_USER_IMPERSONATION` 设置。将该值设置为 `true` 可打开 PXF 用户模拟，或将其设置为 `false` 可将其关闭。例如：

```
PXF_USER_IMPERSONATION="true"
```

- 使用 `pxf cluster sync` 命令拷贝更新的文件 `pxf-env.sh` 文件到 Greenplum 数据库集群。例如

```
gpadmin@gpmaster$ $GPHOME/pxf/bin/pxf cluster sync
```

- 如果先前已启动 PXF，请按照 [Restarting PXF](#) 中的说明在每个 Greenplum 数据库 segment 主机上重新启动它来使新设置生效。

配置 Hadoop 代理

启用 PXF 用户角色(默认设置)后，您必须配置 Hadoop `core-site.xml` 配置文件以打开用户模拟 PXF。步骤如下：

- 在你的 Hadoop 集群中，使用文本编辑器打开 `core-site.xml` 配置文件，或者使用 Ambari 添加或编辑此过程中描述的 Hadoop 属性值。
- 设置属性值 `hadoop.proxyuser.<name>.hosts` 指定允许代理请求的 PXF 主机名列表。将 PXF 代理用户(通常为 `gpadmin`)替换为，并在用逗号分隔的列表中提供多个 PXF 主机名。例如：

```
<property>
  <name>hadoop.proxyuser.gpadmin.hosts</name>
  <value>pxfhost1,pxfhost2,pxfhost3</value>
</property>
```

- 设置属性值 `hadoop.proxyuser.<name>.groups` 指定能被 PXF 模拟的 HDFS 组列表。您应该将此列表限制为仅需要从 PXF 访问 HDFS 数据的那些组。例如：

```
<property>
  <name>hadoop.proxyuser.gpadmin.groups</name>
```

```
<value>group1,group2</value>
</property>
```

4. 在修改过 `core-site.xml` 配置文件之后，你必须重启Hadoop使你的配置生效。
5. 将更新后的 `core-site.xml` 文件复制到master服务器上的PXF Hadoop服务器配置目录 `$PXF_CONF/servers/default` 中，并将配置同步到standby服务器和每个Greenplum数据库segment主机。

Hive用户模拟

PXF Hive连接器使用Hive MetaStore确定Hive表的HDFS位置，然后直接访问基础HDFS文件。Hive不需要特定的模拟配置，因为 `core-site.xml` 中的Hadoop代理配置也适用于以这种方式访问的Hive表。

Hbase用户模拟

为了使用户模拟功能能够与HBase一起使用，必须在HBase配置中启用 `AccessController` 并重新启动群集。有关 `hbase-site.xml` 配置设置，请参阅《Apache HBase参考指南》中的[61.3 Server-side Configuration for Simple User Access Operation](#) 以进行简单的用户访问操作。

Greenplum数据库® 6.0文档

- 管理员指南

- Greenplum数据库概念

- 关于Greenplum的架构

- 关于管理和监控工具

- 关于Greenplum数据库中的并发控制

- 管理事务ID的例子**

- 关于并行数据装载

- 关于Greenplum数据库中的冗余和故障切换

- 关于Greenplum数据库中的数据库统计信息

- 管理一个Greenplum系统

- 管理Greenplum数据库访问

- 定义数据库对象

- 分布与倾斜

- 插入, 更新, 和删除数据

- 查询数据

- 使用外部数据

- 装载和卸载数据

- 性能管理

Greenplum database 5管理员
指南

对于Greenplum数据库来说，事务ID（XID）值是一个自增的32位(2^{32})值。最大的无符号32位置是4,294,967,295，或者说大约40亿。当达到最大值后，XID值会重新从0开始。Greenplum数据库以两种特性来处理XID值的限制：

- 使用模 2^{32} 的算法来计算XID值，这允许Greenplum数据库重用XID值。取模计算基于XID决定事务的顺序，即一个事务是在另一个之前还是之后发生。
每一个XID值可以有最多二十亿 (2^{31}) 个XID值表示在它之前的事务，还有二十亿 ($2^{31} - 1$) 个XID值表示在它之后更新的事务。XID值可以被认为是一组循环没有终点的值，就像24小时的时钟一样。
使用Greenplum数据库的取模计算，只要两个XID都在彼此的 2^{31} 个事务范围之内，比较它们能够得到正确的结果。
- Greenplum数据库用一个冻结XID值作为当前（可见）数据行的XID。设置一行的XID为冻结XID完成了两个功能。
 - 当Greenplum数据库使用取模计算比较XID时，比起任何其他XID来，冻结XID总是较小的一个、较早的一个。如果一行的XID没有被设置为冻结XID并且执行了 2^{31} 个新的事务，从取模计算的角度来看，该行就像是一个在未来出现的行。
 - 当该行的XID被设置为冻结XID时，原来的XID可以被使用而不会出现重复的XID。这将会保持磁盘上分配了XID的数据行数低于 (2^{32}) 。

Note: Greenplum数据库只给涉及DDL或者DML操作的事务分配XID值，它们通常是唯一需要XID的事务。

Parent topic: [关于Greenplum数据库中的并发控制](#)

简单的MVCC例子

这是一个简单例子，它展示了MVCC数据库中的概念以及如何用事务ID管理数据和事务。这个简单的MVCC数据库例子有一个单一表构成：

- 该表是一个具有2列4行数据的简单表。
- 合法的事务ID（XID）值从0到9，在9之后XID会重新从0开始。
- 冻结XID是-2。这和Greenplum数据库的冻结XID不同
- 事务都在单一行上执行。
- 只允许插入和更新操作执行。
- 所有被更新的行都保留在磁盘上，不允许移除废弃的操作。

该例子只更新amount值。不对表做其他更改。

这个例子展示了这些概念。

- 如何用事务ID来管理一个表上的多个同时发生的事务。
- 如何用冻结XID来管理事务ID
- 取模计算怎样基于事务ID决定事务的顺序

管理同时发生的事务

这个表是磁盘上没有更新过的初始表数据。该表包含两个用于事务ID的数据库列xmin（创建该行的事务）和xmax（更新该行的事务）。在这个表中，更改会按照顺序增加到表的底端。

Table 1. 示例表

item	amount	xmin	xmax
widget	100	0	null
giblet	200	1	null
sprocket	300	2	null
gizmo	400	3	null

下一个表展示了执行了对amount值的一些更新后的表数据。

- xid = 4: update tbl set amount=208 where item = 'widget'
- xid = 5: update tbl set amount=133 where item = 'sprocket'
- xid = 6: update tbl set amount=16 where item = 'widget'

在下一个表中，粗体项是该表的当前行。其他行是废弃行，即磁盘上不是当前数据的表数据。使用xmax值，用户可以通过选择其值为null值的行来判断哪些是当前行。Greenplum数据库使用了一种有点不同的方法来决定当前的表行。

Table 2. 带更新的示例表

item	amount	xmin	xmax
widget	100	0	4
giblet	200	1	null
sprocket	300	2	5

gizmo	400	3	null
widget	208	4	6
sproket	133	5	null
widget	16	6	null

这个简单的MVCC数据库用XID值来判断该表的状态。例如，所有这些独立的事务并发执行。

- 更改sprocket的amount值为133的UPDATE命令（xmin值5）
- 返回sprocket的值的SELECT命令。

在UPDATE事务期间，数据库会返回sprocket的值是300，直到UPDATE事务完成。

管理XID和冻结XID

对于这个简单的例子，该数据库接近于耗尽可用的XID值。

当Greenplum数据库接近于耗尽可用XID值时，Greenplum数据库会采取这些行动。

- Greenplum数据库发出一个警告提示数据库快要耗尽XID值。

```
WARNING: database "database_name" must be vacuumed
within number_of_transactions transactions
```

- 在最后一个XID被分配前，Greenplum停止接受事务以防止把一个XID分配两次，并且发出下面的消息。

```
FATAL: database is not accepting commands to avoid
wraparound data loss in database "database_name"
```

为了管理事务ID以及存储在磁盘上的表数据，Greenplum数据库提供了VACUUM命令。

- 一次VACUUM操作会空出XID值，这样一个表可能有超过10行被修改xmin值为冻结XID。
- 一次VACUUM操作会处理磁盘上废弃的或者被删除的表行。这个数据库的VACUUM命令将XID值改为obsolete以表示废弃行。一次Greenplum数据库的VACUUM操作（不带FULL选项），会适时地以对性能和数据可用性最小的影响移除磁盘上的行。

对于该示例表，一次VACUUM操作已经在该表上执行。该命令更新了磁盘上的表数据。这个版本的VACUUM命令与Greenplum数据库的命令有点不同，但是概念是相同的。

- 对于磁盘上不再是当前行的widget和sprocket行，这些行会被标记为obsolete。

- 对于当前版本的gibblet和gizmo行，其xmin已经被改为冻结XID。其值仍然是当前的表值（该行的xmax值为null）。不过，该表行对于所有事务是可见的，因为在执行取模计算时，其xmin值为比所有其他XID值更老的冻结XID。

在VACUUM操作之后，XID值0、1、2和3可以被使用。

Table 3. VACUUM后的示例表

item	amount	xmin	xmax
widget	100	obsolete	obsolete
gibblet	200	-2	null
sprocket	300	obsolete	obsolete
gizmo	400	-2	null
widget	208	4	6
sproket	133	5	null
widget	16	6	null

当一个具有xmin值为-2的行被更新后，其xmax值照例被替换为该事务的XID，在任何访问该行的并发事务结束之后这个位于磁盘上的行会被认为是被废弃。

废弃的行可以被从磁盘删除。对于Greenplum数据库，带有FULL选项的VACUUM命令会做更深入的处理来回收磁盘空间。

XID取模计算实例

下一个表展示了在经过了更多UPDATE事务后磁盘上的表数据。XID值已经回卷并且重新从0开始。此时并没有执行额外的VACUUM操作。

Table 4. 带有回卷XID的示例表

item	amount	xmin	xmax
widget	100	obsolete	obsolete
gibblet	200	-2	1
sprocket	300	obsolete	obsolete
gizmo	400	-2	9
widget	208	4	6

sproket	133	5	null
widget	16	6	7
widget	222	7	null
giblet	233	8	0
gizmo	18	9	null
giblet	88	0	1
giblet	44	1	null

在执行比较XID的取模计算时，Greenplum数据库会考虑行的XID和当前可用的XID范围来判断行的XID是否已经发生了XID回卷。

对于示例表来说XID回卷已经发生。按照XID的取模计算，giblet行的XID 1是一个比widget行的XID 7更晚出现的事务，虽然XID值7大于1。

对于widget和sprocket行，XID回卷并没有发生并且XID 7是一个比XID 5更晚出现的事务。

述

Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统
 - 关于Greenplum数据库
发布版本号
 - 启动和停
止Greenplum数据库
 - 访问数据库
 - 配置Greenplum数据库
系统
 - 启用压缩
 - 启用高可用和数据持久
化特征
 - Greenplum数据库高
可用性概述
 - 在Greenplum数据库
中启用镜像
 - 检测故障的Segment
 - 恢复故障Segment
 - 恢复故障的Master
 - 备份和恢复数据库
 - 扩容Greenplum系统
 - 监控Greenplum系统
 - 日常系统维护任务

Greenplum数据库系统的高可用可以通过提供容错硬件平台实现，可以通过启用Greenplum数据库高可用特性实现，也可以通过执行定期监控和运维作业来确保整个系统所有组件保持健康来实现。

硬件平台的最终故障，可能因为常见的持久运行故障或非预期的运行环境。异常断电会导致组件临时不可用。系统可以通过为可能故障的节点配置冗余备份节点来保证异常出现时仍能够不间断提供服务。在一些情况下，系统冗余的成本高于用户的服务终端容忍度。此时，高可用的目标可以改为确保服务能在预期的时间内恢复。

Greenplum数据库的容错和高可用通过以下几种方式实现：

- 硬件级别的RAID存储保护
- 数据存储总和校验
- Greenplum segment节点镜像
- Master镜像
- 双集群
- 数据库备份和恢复

硬件级别RAID

Greenplum数据库部署最佳时间是采用硬件级别的RAID为单盘失败的情况提供高性能的磁盘冗余，避免只采用数据库级别的容错机制。该方式可以在磁盘级别提供低级别的冗余保护。

数据存储总和校验

Greenplum数据库采用总和校验机制在文件系统上验证从磁盘加载到内存的数据没有被破坏。

Greenplum数据库有两种存储用户数据的方式：堆表和追加优化表。两种存储模型均采用总和校验机制验证从文件系统读取的数据。默认配置下，二者采用总和校验机制验证错误的方式基本类似

Greenplum数据库master和segment实例在他们所管理的自有内存中更新页上的数据。当内存页被更新并刷新到磁盘时，会执行总和校验并保

Recommended

存起来。当下次该页数据从磁盘读取时，先进行总和校验，只有成功通过验证的数据才能进入管理内存。如果总和校验失败，就意味着文件系统有损坏等情况发生，此时Greenplum 数据库会生成错误并中断该事务。

默认的总和校验设置能提供最好的保护，可以防止未检测到的磁盘损坏影响到数据库实例及其镜像Segment节点。

堆表的总和校验机制在Greenplum数据库采用gpinit system初始化时默认启用。我们可以通过设置 gpinit system配置文件中的HEAP_CHECKSUM参数为off来禁用堆表的总和校验功能，但是我们强烈不推荐这么做，详见[gpinit system](#)。

一旦集群初始化完成，就不能改变堆表总和校验机制在该集群上的状态，除非重新初始化系统并重载数据库。

可以通过查看只读服务器配置参数 [data_checksums](#) 来查看 堆表的总和校验是否开启。

```
$ gpconfig -s data_checksums
```

当启动Greenplum数据库集群时，[gpstart](#)工具会检查堆表的总和校验机制在master和所有segment 上是启用了还是禁用了。如果有任何异常，集群会启动失败。详情请见[gpstart](#)。

在一些情况下，为了保证数据及时恢复有必要忽略堆表总和校验产生的错误，设置[ignore_checksum_failure](#)系统配置参数为on会使在堆表总和校验失败时只生成一个警告信息，数据页仍然可以被夹在到管理内存中。如果该页被更新并存到磁盘，损坏的数据会被复制到镜像segment节点。因为该操作会导致数据丢失，所以只有在启用数据恢复时才允许设置ignore_checksum_failure参数为on。

追加优化存储表的总和校验可以在使用CREATE TABLE命令创建表时定义。默认的存储选项在 [gp_default_storage_options](#)服务器配置参数中定义。[checksum](#) 存储选项默认被启用并且强烈不建议禁用它。

如果想要禁用追加优化表上的总和校验机制，你可以

- 在创建表时，修改gp_default_storage_options配置参数包含checksum=false 或
- 增加checksum=false选项到CREATE TABLE语句的WITH storage_options语法部分。

注意CREATE TABLE允许为每一个单独的分区表设置包

括checksums在内的存储选项。

查看[CREATE TABLE](#)命令参考

和[gp_default_storage_options](#)配置文件参考语法和示例。

Segment镜像

Greenplum数据库将数据存储在多个segment实例中，每一个实例都是Greenplum数据库的一个PostgreSQL实例，数据依据建表语句中定义的分布策略在segment节点中分布。启用segment镜像时，每个segment实例都由一对 *primary* 和 *mirror* 组成。镜像segment采用基于预写日志（WAL）流复制的方式保持与主segment的数据一致。详情请见[Segment镜像概述](#)。

镜像实例通常采用gpinitsystem或gpexpand工具进行初始化。作为最佳实践，为了保证单机失败镜像通常运行在与主segment不同的主机上。将镜像分配到不同的主机上也有不同的策略。当搭配镜像和主segment的放置位置时，要充分考虑单机失败发生时处理倾斜最小化的场景。

Master镜像

在一个高可用集群中，有两种master实例，*primary* 和 *standby*。像segment一样，master和standby 应该部署在不同的主机上，以保证集群不出现单节点故障问题。客户端只能连接到primary master并在上面执行查询。standby master采用基于预写日志（WAL）流复制的方式保持与primary master的数据一致。详情请见[Master镜像概述](#)。

如果master故障了，管理员可以通过运行gpactivatestandby工具切换standby master成为新的primary master。可以通过在master和standby上配置一个虚拟IP地址来保证当发生切换后，客户端不需要在不同的网址之间切换。如果master主机故障，虚拟IP可以漂移到新的活动master节点上继续提供服务。

双集群

可以通过维护两套Greenplum数据库集群，都存储相同的数据来提供额外的冗余。

保持双集群数据同步有两种方法，分别叫做 双 和 备份 恢复。

双ETL提供一个与主集群数据一致的热备份集群。ETL（抽取，转换和加载）大致的过程为清理数据，转换数据，验证数据和加载数据进入数据仓库。双ETL的方式，以上过程会被执行两次，每个集群一次，每次都需要被验证。这允许在两个集群上进行查询数据操作，还可以提高查询的吞吐量为原来的两倍。应用可以有效的利用两套集群的优势，页可以确保ETL成功进行并在两套集群上验证通过。

通过备份/恢复的方法来维护一个双集群，可以直接采用在主集群上创建备份，然后恢复到第二个集群上。该方法可能会比双ETL的方式花费更多的时间来同步数据，但是对应用端特定业务逻辑开发的要求几乎没有。双ETL的优势是同步频率上更快，时间间隔更小。

备份和恢复

建议经常备份数据库，可以保证一旦出现问题可以很容易的重新生成数据库集群。备份可以很好的保护误操作、软件错误和硬件错误。

使用[gpbackup](#)工具备份Greenplum数据库。gpbackup在所有segment上执行并行备份，所以备份能力随着硬件增加线性扩展。

当我们设计备份策略时，最主要的关注点是将数据存储在哪里。数据可以备份在每个segment各自的本地存储中，但是存储在该位置会导致正常segment可用生产空间锐减，更重要的是，硬件失败可能会摧毁活动segment的生产数据和备份数据。执行完备份后，备份文件应该从主集群移动到独立、安全的存储位置。另外，备份可以直接存储在独立存储中。

采用gpbackup和gprestore工具，可以从一个远程位置或存储设备发送/读取备份。目前该存储插件支持连接到Amazon S3存储服务和Dell EMC Data Domain存储设备。

采用备份/恢复存储扩展API，您可以创建gpbackup和gprestore工具可用的定制化插件，用来集成您的存储系统到Greenplum数据库。

更多如何使用gpbackup和gprestore的信息，请见[使用gpbackup和gprestore并行备份](#)。

- [Segment镜像概述](#)
- [Master镜像概述](#)

Parent topic: [启用高可用和数据持久化特征](#)

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库
发布版本号

启动和停
止Greenplum数据库

□ 访问数据库

建立一个数据库会话

支持的客户端应用

Greenplum数据库客
户端应用

用psql连接

使用PgBouncer连接
池

数据库应用接口

连接问题的发现及解
决

□ 配置Greenplum数据库
系统

启用压缩

□ 启用高可用和数据持久
化特征

□ 备份和恢复数据库

□ 扩容Greenplum系统

用户可以通过使用一种PostgreSQL兼容的客户端程序连接到Greenplum数据库，例如psql。 用户和管理员总是通过*master*连接到Greenplum数据库，Segment不能接受客户端连接。

为了建立一个到Greenplum数据库Master的连接，用户将需要知道下列连接信息并且相应地配置用户的客户端程序。

Table 1. 连接参数

连接参数	描述	环境变量
应用名称	连接到数据库的应用名称，保存在application_name连接参数中。默认值是 <i>psql</i> 。	\$PGAPPNAME
数据库名	用户想要连接的数据库名称。对于一个刚初始化的系统，第一次可使用postgres数据库来连接。	\$PGDATABASE
主机名	Greenplum数据库的Master的主机名。默认主机是本地主机。	\$PGHOST
端口	Greenplum数据库的Master实例所运行的端口号。默认为5432。	\$PGPORT
用户名	要以其身份连接的数据库用户（角色）名。这不需要和用户的操作系统用户名一样。如果用户不确定用户的数据库用户名是什么，请咨询用户的Greenplum管理员。注意每一个Greenplum数据库系统都有一个在初始化时自动创建的超级用户账号。这个账号的名称和初始化Greenplum系统的用户的操作系统用户名相同。（最有代表性的是 <i>gpadmin</i> ）。	\$PGUSER

[用psql连接](#)提供了一些连接到Greenplum数据库的示例命令。

Parent topic: 访问数据库



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库

发布版本号

启动和停

止Greenplum数据库

□ 访问数据库

建立一个数据库会话

支持的客户端应用

Greenplum数据库客
户端应用

用psql连接

使用PgBouncer连接
池

数据库应用接口

连接问题的发现及解
决□ 配置Greenplum数据库
系统

启用压缩

□ 启用高可用和数据持久
化特征

□ 备份和恢复数据库

□ 扩容Greenplum系统

用户可以使用多种客户端应用连接到Greenplum数据库：

- 用户的Greenplum安装中已经提供了一些[Greenplum数据库客户端应用](#)psql客户端应用提供了一种对Greenplum数据库的交互式命令行接口。
- 使用标准的[数据库应用接口](#), , 如ODBC和JDBC, 用户可以创建他们自己的客户端应用来接入到Greenplum数据库。因为Greenplum数据库基于PostgreSQL开发, 所以它使用标准的PostgreSQL数据库驱动。
- 大部分使用ODBC和JDBC等标准数据库接口的客户端工具都可以被配置来连接到 Greenplum数据库。

Parent topic: [访问数据库](#)



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库
发布版本号

启动和停
止Greenplum数据库

□ 访问数据库

建立一个数据库会话

支持的客户端应用

Greenplum数据库客
户端应用

用psql连接

使用PgBouncer连接
池

数据库应用接口

连接问题的发现及解
决

□ 配置Greenplum数据库
系统

启用压缩

□ 启用高可用和数据持久
化特征

□ 备份和恢复数据库

□ 扩容Greenplum系统

Greenplum数据库安装后就会带有一些客户端工具应用，它们位于用户的Greenplum数据库Master主机安装的 \$GPHOME/bin目录中。下列是最常用的客户端工具应用：

Table 1. 最常用的客户端应用

名称	用法
createdb	创建一个新数据库
createlang	定义一种新的过程语言
createuser	定义一个新的数据库角色
dropdb	移除一个数据库
droplang	移除一种过程语言
dropuser	移除一个角色
psql	PostgreSQL交互式终端
reindexdb	对一个数据库重建索引
vacuumdb	对一个数据库进行垃圾收集和分析

在使用这些客户端应用时，用户必须通过Greenplum的Master实例连接到一个数据库。用户将需要知道目标数据库的名称、Master的主机名和端口号，还有用于连接的数据库用户名。这些信息可以在命令行上分别用选项-d、-h、-p和-U来提供。如果找到不属于任何一个选项的参数，它将被首先解释为数据库名。

所有这些选项都有默认值，如果该选项没有被指定就会使用其默认值。默认主机是本地主机。默认端口号是5432。默认用户名是操作系统的用户名，同时也是默认的数据库名。注意操作系统用户名和Greenplum数据库用户名并不需要一样。

如果默认值和实际情况不同，用户可以设置环境变量PGDATABASE、PGHOST、PGPORT和PGUSER为合适的值，或者使用一个psql~/.pgpass文件来包含常用的口令。

更多有关Greenplum数据库环境变量的信息，请见Greenplum数据库参考指南。有关psql的详细信息请见，Greenplum数据库工具指南。

Parent topic: 访问数据库



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库
发布版本号启动和停
止Greenplum数据库

□ 访问数据库

建立一个数据库会话

支持的客户端应用

Greenplum数据库客
户端应用

用psql连接

使用PgBouncer连接
池

数据库应用接口

连接问题的发现及解
决□ 配置Greenplum数据库
系统

启用压缩

□ 启用高可用和数据持久
化特征

□ 备份和恢复数据库

□ 扩容Greenplum系统

依靠用户使用的默认值或者已经设置的环境变量，下面的例子展示了如何通过psql 来访问数据库：

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

```
$ psql gpdatabase
```

```
$ psql
```

如果还没有创建一个用户定义的数据库，用户可以通过连接到postgres数据库来访问系统。例如：

```
$ psql postgres
```

在连接到一个数据库后，psql提供了一个提示符，提示符由psql 当前连接的数据库名后面加上=> (如果用户是数据库超级用户则会是=#) 构成。例如：

```
gpdatabase=>
```

在提示符处，用户可以输入SQL命令。为了能把一个SQL命令发送到服务器并且执行，SQL命令必须以一个 ; (分号) 结束。例如：

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

有关使用psql客户端应用以及SQL命令及语法的信息请见Greenplum数据库参考指南。

Parent topic: [访问数据库](#)



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库
发布版本号

启动和停
止Greenplum数据库

□ 访问数据库

建立一个数据库会话

支持的客户端应用

Greenplum数据库客
户端应用

用psql连接

**使用PgBouncer连接
池**

数据库应用接口

连接问题的发现及解
决

□ 配置Greenplum数据库
系统

启用压缩

□ 启用高可用和数据持久
化特征

□ 备份和恢复数据库

□ 扩容Greenplum系统

PgBouncer工具可以管理PostgreSQL和Greenplum数据库连接的连接池。

以下话题描述了如何在Greenplum数据库上配置PgBouncer。更多有关如何在PostgreSQL上使用PgBouncer的信息，请参考 [PgBouncer网站](#)。

- [概述](#)
- [迁移PgBouncer](#)
- [配置PgBouncer](#)
- [启动PgBouncer](#)
- [管理PgBouncer](#)

Parent topic: [访问数据库](#)

概述

数据库连接池是一种数据库连接的缓存。一旦一个池子的连接被建立，连接池就能消除创建数据库连接的开销，这样客户端能更快地连接并且服务器的负载也能被降低。

PgBouncer连接池来自于PostgreSQL社区，它被包括在Greenplum数据库中。PgBouncer可以为多个数据库管理连接池，并且这些数据库可以位于不同的Greenplum数据库集群或者PostgreSQL后端。PgBouncer会为每一种数据库用户与数据库的组合建立一个池。一个被池化的连接只能被来自于同一个用户和数据库的另一个连接请求重用。当客户端断开连接后，PgBouncer会将连接归还给连接池以做下次重用。

PgBouncer有三种池模式来共享连接：

- 会话池化 – 当一个客户端连接时，只要它保持连接状态，就分配给它一个连接。当该客户端断开连接时，该连接才被放回到池中。
- 事务池化 – 在一个事务运行期间，分配一个连接给客户端。当PgBouncer发现事务完成，该连接就被放回到池中。这种模式只能被用于不使用依赖于会话的特性的应用。
- 语句池化 – 语句池化类似于事务池化，但是不允许多事务。这种模式的目标是为了在客户端强制自动提交模式，且它的定位是PostgreSQL上的PL/Proxy。

可以为PgBouncer设置一种默认的池模式，并且该模式可以被单个数据库和用户覆盖。

通过连接到一个虚拟的pgbouncer数据库，用户可以使用类SQL命令监控和管理PgBouncer。可以不重启PgBouncer就修改配置参数，只需要重新载入配置文件就能发现更改。PgBouncer支持PostgreSQL和Greenplum数据库上的标准连接接口。Greenplum数据库客户端应用（例如psql）可以连接到正在运行的PgBouncer主机和端口号，而不是Greenplum的Master主机和端口号。

PgBouncer有一个类psql的管理控制台界面。授权用户可以通过该控制台连接到一个虚拟数据库来监控和管理PgBouncer。您可以通过管理控制台来管理PgBouncer后台进程，也可以用控制台在运行时更新和重载PgBouncer配置，而不停止和重启PgBouncer进程。

PgBouncer原生支持TLS。

迁移PgBouncer

当您迁移到一套新版本的Greenplum数据库时，必须迁移PgBouncer实例到新版本的Greenplum数据库。

- 如果您正在迁移**Greenplum 5.8.x**版本数据库或更早版本，可以在不丢弃连接的情况下迁移PgBouncer。采用带有-R选项和配置文件的命令拉起新的PgBouncer进程。

```
$ pgbouncer -R -d pgbouncer.ini
```

-R（重启）选项启动新进程以Unix套接字的方式连接到旧进程并发出以下命令：

```
SUSPEND;
SHOW FDS;
SHUTDOWN;
```

当新进程检测到旧进程完成后，它接管旧连接的工作。最可能的原因是SHOW FDS命令将实际的文件定位符发送新进程。如果事务因任何原因失败，新进程会被杀掉，旧进程仍然接管原来的工作。

- 如果您正在迁移**Greenplum 5.9.0**版本数据库或更高版本，您必须关闭旧版本安装文件中的PgBouncer实例，在新数据库安装文件中重新配置并启动PgBouncer实例。
- 如果您采用stunnel加密旧安装文件的PgBouncer连接，在新环境中必须使用与PgBouncer 1.8.1 及更新版本适配的内建TLS配

置SSL/TLS。

- 如果您在旧环境中使用了LDAP授权，在新环境中必须使用与PgBouncer 1.8.1及更新版本适配的内建集成PAM配置LDAP。必须移除或替换auth_file中所有以ldap://开头的密码串。

配置PgBouncer

通过配置文件配置PgBouncer和它到Greenplum数据库的访问。配置文件通常命名为pgbouncer.ini，提供Greenplum数据库的位置信息。pgbouncer.ini文件也指定PgBouncer的进程、连接池、授权用户和授权配置。

pgbouncer.ini配置文件示例内容如下：

```
[databases]
postgres = host=127.0.0.1 port=5432 dbname=postgres
pgb_mydb = host=127.0.0.1 port=5432 dbname=mydb

[pgbouncer]
pool_mode = session
listen_port = 6543
listen_addr = 127.0.0.1
auth_type = md5
auth_file = users.txt
logfile = pgbouncer.log
pidfile = pgbouncer.pid
admin_users = gpadmin
```

有关PgBouncer配置文件格式和其支持的配置参数列表等信息，请见 [pgbouncer.ini](#) 参考页面。

当客户端连接到PgBouncer时，连接池从pgbouncer.ini配置文件中查找受访数据库（可能是实际数据库的别名）对应的主机名、端口号和数据库名。该配置文件也定义数据库的授权模式。

PgBouncer要求有一个授权文件，该文件是一个包含Greenplum数据库用户名和密码的文本文件。文件内容与pgbouncer.ini中配置的auth_type 授权类型密切相关。密码可能是干净文本或MD5编码的字符串。用户也可以配置PgBouncer通过查询目标数据库来获取授权文件中未定义的密码信息。

PgBouncer授权文件格式

PgBouncer有自己的用户授权文件。用户可以在`pgbouncer.ini`配置文件的`auth_file`属性中定义文件名。`auth_file`文本文件格式如下：

```
"username1" "password" ...
"username2" "md5abcdef012342345" ...
```

`auth_file`为每个用户占用一行。每行至少有两个域，两个都用双引号包裹（" "）。第一部分定义Greenplum数据库用户名，第二部分为明文或MD5编码的密码。PgBouncer忽略改行余下的部分。

(`auth_file`文件格式与Greenplum数据库所用的授权信息文件`pg_auth`相似。PgBouncer可以直接使用Greenplum数据库授权文件。)

使用MD5编码的密码，密码格式如下：

```
"md5" + MD5_encoded(<password><username>)
```

可以从`pg_shadow`视图中获取所有Greenplum数据库用户的MD5编码密码。

为PgBouncer配置基于HBA的授权

PgBouncer支持基于HBA的授权。要为PgBouncer配置基于HBA的授权，在`pgbouncer.ini`配置文件中设置`auth_type=hba`，并在`pgbouncer.ini`文件中配置HBA格式文件参数`auth_hba_file`。

名称为`hba_bouncer.conf`的PgBouncer HBA文件的内容如下：

local	all	bouncer	trust	
host	all	bouncer	127.0.0.1/32	trust

相关`pgbouncer.ini`配置文件中的对应配置部分：

```
[databases]
p0 = port=15432 host=127.0.0.1 dbname=p0 user=bouncer
pool_size=2
p1 = port=15432 host=127.0.0.1 dbname=p1 user=bouncer
...

[pgbouncer]
...
```

```

auth_type = hba
auth_file = userlist.txt
auth_hba_file = hba_bouncer.conf
...

```

有关PgBouncer支持的HBA授权文件格式的详细讨论，请参考[HBA file format](#)。

启动PgBouncer

用户可以在Greenplum数据库master主机或其他服务器上运行PgBouncer。如果在一台单独的服务器上安装PgBouncer，用户可以使用PgBouncer管理客户端，通过更新PgBouncer配置文件和重载配置很容易的切换客户端连接到standby master。

参考以下配置设置PgBouncer。

1. 创建PgBouncer配置文件。例如，增加以下文本到文件pgbouncer.ini中：

```

[databases]
postgres = host=127.0.0.1 port=5432 dbname=postgres
pgb_mydb = host=127.0.0.1 port=5432 dbname=mydb

[pgbouncer]
pool_mode = session
listen_port = 6543
listen_addr = 127.0.0.1
auth_type = md5
auth_file = users.txt
logfile = pgbouncer.log
pidfile = pgbouncer.pid
admin_users = gpadmin

```

该文件列出了数据库和它们连接的详细信息。该文件也配置了PgBouncer实例。有关PgBouncer配置文件内容和格式的详细信息，请参考[Refer to the pgbouncer.ini](#) 参考页面。

2. 创建授权文件。文件名应该和pgbouncer.ini文件中定义的auth_file参数名字一样，users.txt。每行包含一个用户名和一个密码。密码的格式应该匹配PgBouncer 配置文件中定义的auth_type。如果auth_type参数为plain，密码串应该为干净的文本文档密码，例如：

```
"gpadmin" "gpadmin1234"
```

。如果auth_type如下面例子中为md5，授权部分必须是MD5编码格式。 MD5编码的密码格式如下：

```
"md5" + MD5_encoded(<password><username>)
```

3. 开始启动 pgbouncer:

```
$ $GPHOME/bin/pgbouncer -d pgbouncer.ini
```

-d选项代表以后台进程的形式运行PgBouncer。 pgbouncer命令的语法和选项，请见[pgbouncer](#) 参考页面。

4. 更新客户应用连接到pgbouncer，以代替直接连接到Greenplum数据库服务器。例如，连接到上面配置的Greenplum数据库mydb，像如下方式一样运行psql：

```
$ psql -p 6543 -U someuser pgb_mydb
```

-p选项的值代表用户配置的PgBouncer实例的listen_port端口号。

管理PgBouncer

PgBouncer提供一个类psql的管理控制台，可以通过指定PgBouncer端口号 和虚拟数据库名称pgbouncer来登录到PgBouncer管理控制台。该控制台接受类SQL命令，这些命令允许用户监控、重新配置和管理PgBouncer。

有关PgBouncer管理控制台命令的完整文档，请参考[PgBouncer管理控制台](#)命令参考。

要开始使用PgBouncer管理控制台，请遵循下列步骤。

1. 用psql登录到pgbouncer虚拟数据库：

```
$ psql -p 6543 -U username pgbouncer
```

*username*必须是pgbouncer.ini配置文件中 admin_users参数中所设置的值。如果pgbouncer进程运行在用户登录的当前Unix用户的UID下，用户还可以用该用户名登录。

2. 要看PgBouncer管理控制台命令，运行SHOW help命令：

```
pgbouncer=# SHOW help;
NOTICE: Console usage
DETAIL:
    SHOW
    HELP | CONFIG | DATABASES | POOLS | CLIENTS | SERVERS | VERSION
        SHOW FDS | SOCKETS | ACTIVE_SOCKETS | LISTS | MEM
        SHOW DNS_HOSTS | DNS_ZONES
        SHOW STATS | STATS_TOTALS | STATS_AVERAGES
        SET key = arg
        RELOAD
        PAUSE [ <db> ]
        RESUME [ <db> ]
        DISABLE <db>
        ENABLE <db>
        KILL <db>
        SUSPEND
        SHUTDOWN
```

3. 如果用户对PgBouncer配置文件pgbouncer.ini做了修改， 用户可以用RELOAD命令重载它：

```
pgbouncer=# RELOAD;
```

映射PgBouncer客户端到Greenplum数据库服务器连接

要映射PgBouncer客户端到Greenplum数据库服务器连接， 可以使用PgBouncer管理控制台 命令SHOW CLIENTS和SHOW SERVERS：

1. 使用ptr和link把本地客户端连接映射到服务器连接。
2. 使用客户端连接的addr和port标识来自客户端的TCP连接。
3. 使用local_addr和local_port标识到服务器的TCP连接。

Greenplum数据库® 6.0 文档

- 管理员指南

- Greenplum数据库概念

- 管理一个Greenplum系统

- 关于Greenplum数据库
发布版本号

- 启动和停
止Greenplum数据库

- 访问数据库

- 建立一个数据库会话

- 支持的客户端应用

- Greenplum数据库客
户端应用

- 用psql连接

- 使用PgBouncer连接
池

数据库应用接口

- 连接问题的发现及解
决

- 配置Greenplum数据库
系统

- 启用压缩

- 启用高可用和数据持久
化特征

- 备份和恢复数据库

- 扩容Greenplum系统

用户可能想要开发自己的客户端应用接入到Greenplum数据
库。PostgreSQL为最常用的数据库应用编程接口（API）提供了数种数据库
驱动，这些同样也能在Greenplum数据库中使用。这些驱动作为一个独立的
下载提供。每一种驱动（除了随PostgreSQL提供的libpq）都是一个独立的
PostgreSQL开发项目并且必须被下载、安装并且配置以连接到
Greenplum数据库。可用的驱动如下：

Table 1. Greenplum数据库接口

API	PostgreSQL驱 动	下载链接
ODBC	psqlODBC	https://odbc.postgresql.org/
JDBC	pgjdbc	https://jdbc.postgresql.org/
Perl DBI	pgperl	https://metacpan.org/release/DBD-Pg
Python DBI	pygresql	http://www.pygresql.org/
libpq C 库	libpq	https://www.postgresql.org/docs/9.4/libpq.html

用API访问Greenplum数据库的一般步骤是：

1. 从合适的来源下载用户的编程语言平台以及相应的API。例如，用户可以
从 Oracle得到Java开发工具包（JDK）和JDBC API。
2. 根据API规范编写用户的客户端应用。在编程时，注意Greenplum数据库
中的SQL支持这样用户才不会使用不被支持的SQL语法。更多信息请见
Greenplum 数据库参考指南。

下载合适的驱动并且配置到Greenplum数据库Master实例的连接。

Parent topic: [访问数据库](#)



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库
发布版本号启动和停
止Greenplum数据库

□ 访问数据库

建立一个数据库会话

支持的客户端应用

Greenplum数据库客
户端应用

用psql连接

使用PgBouncer连接
池

数据库应用接口

连接问题的发现及解
决□ 配置Greenplum数据库
系统

启用压缩

□ 启用高可用和数据持久
化特征

□ 备份和恢复数据库

□ 扩容Greenplum系统

很多事情都可能阻止客户端应用成功地连接到Greenplum数据库。这个主题解释了一些常见的连接问题的原因以及如何改正它们。

Table 1. 常见连接问题

问题	解决方案
没有用于主机或者 用户的pg_hba.conf配 置	要允许Greenplum数据库接受远程客户端连 接，用户必须配置用户的Greenplum 数据库 的Master实例，这样来自于客户端主机和数 据库用户的连接才会被允许连接 到Greenplum数据库。这可以通 过在pg_hba.conf配置文件（位于Master实例 的数据目录中）中增加合适的条目就能做 到。更多详细的信息请见 允许访 问Greenplum数据库 。
Greenplum数据库没 有运行	Greenplum数据库的Master实例没有运行，用 户将无法连接。用户可以通过 在Greenplum的Master主机上运行gpstate工 具来验证Greenplum数据库系统是否正常运 行。
网络问题 Interconnect 超时	如果用户从一个远程客户端连接 到Greenplum的Master主机，网络问题 可能阻 止连接（例如， DNS主机名解析问题、主 机系统没有运行等等）。为了确认网络问题不 是原因，可尝试从远程客户端主机连接 到Greenplum的Master主机。例如： ping hostname 。

如果系统不能解析主机名和Greenplum数据库
所涉及的主机的IP地址，查询和连接将会失
败。对于某些操作，到Greenplum数据
库Master的连接会使用localhost而其他连接
使用真实的主机名，因此用户必须能解析
两者。如果用户遇到这种错误，首先确认用
户能够从Master主机通过网络连接到
Greenplum数据库阵列中的每一台主机。
在Master和所有Segment的/etc/hosts文件
中，确认有Greenplum数据库阵
及所有主机的正确的主机名和IP地址。
127.0.0.1 必须解析为localhost。

已有太多客户端连接

默认情况下，Greenplum数据库被配置为在Master和每个Segment上分别允许最多250和 750个并发用户连接。导致该限制会被超过的连接尝试将被拒绝。这个限制由Greenplum数据库Master的 `postgresql.conf` 配置文件中的 `max_connections` 参数控制。如果用户为Master更改了这个设置，用户还必须在Segment上做出适当的更改。

Parent topic: [访问数据库](#)

关于Greenplum数据库的Master参数和本地参数

Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统
- 关于Greenplum数据库发布版本号
 - 启动和停止Greenplum数据库
- 访问数据库
- 配置Greenplum数据库系统

关于Greenplum数据库的Master参数和本地参数

- 设置配置参数
 - 查看服务器配置参数设置
 - 配置参数种类
 - 启用压缩
- 启用高可用和数据持久化特征
- 备份和恢复数据库
- 扩容Greenplum系统
- 监控Greenplum系统
 - 日常系统维护任务

Recommended

服务器配置文件包含着配置服务器行为的参数。Greenplum数据库的配置文件`postgresql.conf`位于数据库实例的数据目录之下。

Master和每一个Segment实例都有自己的`postgresql.conf`文件。一些参数是本地的：每个Segment实例检查它的`postgresql.conf`文件来得到这类参数的值。在Master和每一个Segment实例上都要设置本地参数。

其他参数是用户要在Master实例上设置的`master`参数。其值会在查询运行时被向下传递到Segment实例（或者在某些情况中会被忽略）。

有关`local`以及`master`服务器配置参数的信息请见 *Greenplum数据库参考指南*。

Parent topic: [配置Greenplum数据库系统](#)



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库
发布版本号

启动和停
止Greenplum数据库

□ 访问数据库

□ 配置Greenplum数据库
系统

关于Greenplum数据
库的Master参数和本
地参数

□ 设置配置参数

查看服务器配置参数
设置

配置参数种类

启用压缩

□ 启用高可用和数据持久
化特征

□ 备份和恢复数据库

□ 扩容Greenplum系统

监控Greenplum系统

日常系统维护任务

Recommended

很多配置参数限制了谁能改变它们以及何时何处它们可以被设置。例如，要改变特定的参数，用户必须是一个Greenplum数据库超级用户。其他参数只能从`postgresql.conf`文件中在系统级别上被设置，或者还要求系统重启让设置生效。

很多配置参数是会话参数。用户可以在系统级别、数据库级别、角色级别或者会话级别设置会话参数。数据库用户可以在他们的会话中改变大部分会话参数，但是某些要求超级用户权限。

关于设置服务器配置参数的信息请见*Greenplum*数据库参考指南。

- [设置本地配置参数](#)
- [设置Master配置参数](#)

Parent topic: 配置Greenplum数据库系统



Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统

关于Greenplum数据库
发布版本号

启动和停
止Greenplum数据库

- 访问数据库
- 配置Greenplum数据库
系统

关于Greenplum数据
库的Master参数和本
地参数

- 设置配置参数

查看服务器配置参数
设置

配置参数种类

启用压缩

- 启用高可用和数据持久
化特征
- 备份和恢复数据库
- 扩容Greenplum系统

监控Greenplum系统

日常系统维护任务

Recommended

SQL命令SHOW允许用户查看当前的服务器配置参数设置。例如，要查
看所有参数的设置：

```
$ psql -c 'SHOW ALL;'
```

SHOW只列出Master实例的设置。要查看整个系统（Master和所有的Segment）中一个特定参数的值，使用gpconfig工具。例如：

```
$ gpconfig --show max_connections
```

Parent topic: [配置Greenplum数据库系统](#)



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库

发布版本号

启动和停

止Greenplum数据库

□ 访问数据库

□ 配置Greenplum数据库
系统关于Greenplum数据
库的Master参数和本
地参数

□ 设置配置参数

查看服务器配置参数
设置

配置参数种类

启用压缩

□ 启用高可用和数据持久
化特征

□ 备份和恢复数据库

□ 扩容Greenplum系统

监控Greenplum系统

日常系统维护任务

Recommended

配置参数影响着多种服务器行为，例如资源消耗、查询调节以及认
证。有关配置参数种类的细节，请见Greenplum数据库参考指南中的参
数分类部分。

Parent topic: [配置Greenplum数据库系统](#)



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 定义外部表

file://协议

gpfdist://协议

gpfdists:// 协议

pxf:// 协议

s3:// 协议

使用自定义协议

处理外部表数据中的
错误创建和使用外
部Web表□ Examples for
Creating External
Tables

gpfdist://协议被用在一个URI中引用一个正在运行的 gpfdist 实例。

gpfdist 工具从一个文件主机上的目录中把外部数据文件并行提供给 Greenplum 数据库的所有 Segment。

gpfdist 位于 Greenplum 数据库的 Master 主机以及每个 Segment 主机上的 \$GPHOME/bin 目录。

在外部数据文件所在的主机上运行 gpfdist。 gpfdist 会自动解压 gzip (.gz) 和 bzip2 (.bz2)。 用户可以使用通配符 (*) 或者其他 C 风格的模式匹配来表示要读取的多个文件。 指定的文件都被假定是相对于启动 gpfdist 实例时指定的目录。

所有的主 Segment 并行地访问外部文件， Segment 的数量服从 gp_external_max_segments 服务器配置参数中设置的数量。 在 CREATE EXTERNAL TABLE 语句中使用多个 gpfdist 数据源可以放大外部表的扫描性能。

gpfdist 支持数据转换。 你可以写一个转换进程将外部数据转入或转出到一个不被 Greenplum 数据库外部表直接支持的格式。

更多关于 gpfdist 的配置， 参考 [使用 Greenplum 并行文件服务
器\(gpfdist\)](#)。

参考 gpfdist 相关文档了解更多关于使用 gpfdist 处理外部表的信息。

Parent topic: [定义外部表](#)



Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 定义外部表

file://协议

gpfdist://协议

gpfdists:// 协议

pxf:// 协议

s3:// 协议

使用自定义协议

处理外部表数据中的错误

创建和使用外部Web表

□ Examples for Creating External Tables

gpfdists:// 协议是gpfdist:// 协议的一个安全版本。

要使用它，用户需要以--ssl 选项运行gpfdist 工具。当在一个URI中指定gpfdists:// 协议时，它启用文件服务器和Greenplum数据库的加密通信以及安全鉴定以保护其不受窃听和中间人等攻击。

gpfdists 以客户端/服务器形式实现了SSL安全性，并且有下列属性和限制：

- 客户端证书是必需的。
- 不支持多语言证书。
- 不支持证书撤销列表（CRL）。
- TLSv1协议被用于TLS_RSA_WITH_AES_128_CBC_SHA加密算法。
- SSL参数不能被更改。
- 支持SSL重新协商。
- SSL忽略被设置为false的主机失配参数。
- 对于gpfdist文件服务器（server.key）和Greenplum数据库（client.key）不支持包含密码的私钥。
- 为操作系统发行合适的证书是用户的责任。通常，支持转换<https://www.sslshopper.com/ssl-converter.html> 中所示的证书。

Note: 一个用gpfdist --ssl选项启动的服务器只能与gpfdists协议通讯。一个不带--ssl选项启动的gpfdists服务器只能与gpfdists协议通讯。

- 客户端证书文件是client.crt

- 客户端私钥文件是client.key

使用下列方法之一来调用gpfdists 协议。

- 用--ssl选项运行gpfdist，然后在CREATE EXTERNAL TABLE语句的LOCATION子句中使用 gpfdists协议。
- 使用一个SSL选项设置为真的gupload YAML控制文件。运行gupload，它会用--ssl选项启动gpfdist服务器，然后使用gpfdists协议。

使用gpfdists 要求下列客户端证书存在于每个Segment的\$PGDATA/gpfdists 目录中。

- 客户端证书文件是client.crt
- 客户端私钥文件是client.key
- 受信证书发行机构root.crt

[Example 3—Multiple](#)

一个装载数据到具有安全性的外部表中的例子可见
[gpfdists instances](#)。

服务器配置参数[verify_gpfdists_cert](#) 控制当Greenplum数据库与gpfdist 实用程序通信以从外部数据源读取数据或将数据写入外部数据源时是否启用SSL证书认证。在测试Greenplum数据库外部表与提供外部数据的gpfdist 实用程序之间的通信时，可以将参数值设置为false，以禁用身份验证。如果值为false，这些SSL异常将被忽略：

- gpfdist使用的自签名SSL证书不被Greenplum数据库信任。
- SSL证书中包含的主机名与运行gpfdist的主机名不匹配。

Warning: 禁用SSL证书身份验证会暴露安全风险，因为不会去验证gpfdists SSL证书。

Parent topic: [定义外部表](#)

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 定义外部表

file://协议

gpfdist://协议

gpfdists:// 协议

pxf:// 协议

s3:// 协议

使用自定义协议

处理外部表数据中的错误

创建和使用外部Web表

□ Examples for Creating External Tables

s3协议使用一个URL指定Amazon S3桶的位置和一个用来在桶里读写文件的前缀。

Amazon简单存储服务 (Amazon S3) 提供了安全、持久、高可扩展的对象存储。有关Amazon S3的信息请见 [Amazon S3](#)。

用户可以定义只读外部表使用S3桶中现有的数据文件作为表数据，或者创建可写外部表将来自于INSERT操作的数据存储到S3桶中的文件。Greenplum数据库使用协议URL中指定的S3 URL以及前缀来为一个只读表选择一个或更多文件，还可以在为 INSERT操作上传S3文件到可写表中时定义文件位置和文件名格式。

s3协议也支持[Dell EMC Elastic Cloud Storage](#) (ECS)，这是一种与Amazon S3兼容的服务。

这一主题包含下列小节：

- [配置和使用S3外部表](#)
- [关于S3协议URL](#)
- [关于S3数据文件](#)
- [s3协议的AWS服务端加密支持](#)
- [s3协议代理支持](#)
- [关于S3协议的配置参数](#)
- [S3协议的配置文件](#)
- [S3协议的限制](#)
- [使用gpcheckcloud工具](#)

配置和使用S3外部表

按照下面这些基本步骤来配置S3协议和使用S3外部表，更多的信息请使用相应的链接。另见[S3协议的限制](#)来更好地理解S3外部表的能力和限制：

1. 配置每一个数据库支持s3协议：

- a. 在每一个将通过s3协议访问S3桶的数据库中，为 s3协议库创建读写函数：

```
CREATE OR REPLACE FUNCTION write_to_s3() RETURNS integer AS
'$libdir/gps3ext.so', 's3_export' LANGUAGE C STABLE;
```

```
CREATE OR REPLACE FUNCTION read_from_s3() RETURNS integer AS
'$libdir/gps3ext.so', 's3_import' LANGUAGE C STABLE;
```

- b. 在每一个将要访问S3桶的数据库中，声明 s3协议并且指定前一步中创

建的读写函数：

```
CREATE PROTOCOL s3 (writefunc = write_to_s3, readfunc =
read_from_s3);
```

Note: 协议名称s3必须和创建用来访问S3资源的外部表中指定的协议一样。

相应的函数会被Greenplum数据库的每一个Segment实例调用。所有的Segment主机必须能够访问该S3桶。

2. 在每一个Greenplum数据库的Segment上，创建并且安装s3 协议的配置文件：

- a. 使用gpcheckcloud工具创建一个s3协议配置文件的模板：

```
gpcheckcloud -t > ./mytest_s3.config
```

- b. 编辑该模板文件以指定连接S3位置所需的accessid 和 secret。有关其他协议参数的配置请见[S3协议的配置文件](#)s3协议的配置文件。

- c. 为Greenplum数据库的所有Segment把该文件复制到所有主机上的相同位置。默认的文件位置是

gpseg_data_dir/gpseg_prefixN/s3/s3.conf.

*gpseg_data_dir*是Greenplum数据库的Segment的数据目录路径，*gpseg_prefix*是Segment前缀，而*N*是Segment的ID。当用户初始化一个Greenplum数据库系统时，Segment的数据目录、前缀和ID就已经被设置好了。

如果用户复制该文件到一个不同的位置或者文件名，那么用户必须用s3协议URL的config参数指定该位置。见[关于S3协议的配置参数](#)。

- d. 使用gpcheckcloud工具验证到该S3桶的连接性：

```
gpcheckcloud -c "s3://<s3-endpoint>/<s3-bucket>
config=./mytest_s3.config"
```

为用户的系统的配置文件指定正确的路径，还有用户想要检查的S3端点名和桶。gpcheckcloud会尝试连接到该S3端点并且列出S3桶中的任何文件（如果可用）。一次成功的连接将以这样的消息结束：

```
Your configuration works well.
```

用户可以有选择地使用gpcheckcloud来验证对该S3桶进行上传和下载。正如[使用gpcheckcloud工具](#)中所描述的。

3. 在完成前几个创建和配置s3协议的步骤后，用户可以在CREATE EXTERNAL TABLE命令中指定一个s3协议URL来定义S3外部表。对于只读的S3表，该URL定义组成该S3表的现有数据文件的位置和前缀。例如：

```
CREATE READABLE EXTERNAL TABLE S3TBL (date text, time text, amt
int)
LOCATION('s3://s3-us-west-
2.amazonaws.com/s3test.example.com/dataset1/normal/')
```

```
config=/home/gpadmin/aws_s3/s3.conf')
FORMAT 'csv';
```

对于可写的S3表，协议URL定义S3位置，Greenplum数据库会把该表的数据文件存储在其中，该URL还定义了一个对表做INSERT操作创建文件时使用的前缀。例如：

```
CREATE WRITABLE EXTERNAL TABLE S3WRIT (LIKE S3TBL)
LOCATION('s3://s3-us-west-
2.amazonaws.com/s3test.example.com/dataset1/normal/
config=/home/gpadmin/aws_s3/s3.conf')
FORMAT 'csv';
```

参考 [关于S3协议URL](#)获取更多信息。

关于S3协议URL

对于s3协议，用户需要在CREATE EXTERNAL TABLE命令的LOCATION子句中指定文件的位置以及一个可选的配置文件位置。下面是语法：

```
's3://S3_endpoint[:port]/bucket_name/[S3_prefix]
[region=S3_region] [config=config_file_location]'
```

s3协议要求指定S3端点和S3桶名。Greenplum数据库的每一个Segment实例必须能够访问该S3位置。可选的S3_prefix值被用来选择只读S3表的文件，或者被用作向S3可写表上传数据时的文件名前缀。

Note: Greenplum数据库的s3协议的URL必须包括S3的端点主机名。

要在LOCATION子句中指定一个ECS端点（一种Amazon S3的兼容服务），用户必须把s3 配置文件参数version设置为2。version 参数控制 region参数是否被用在 LOCATION子句中。当version参数是2的时候，用户还可以指定一个Amazon S3位置。有关 version参数更多信息，参考[S3协议的配置文件](#)。

Note: 尽管S3_prefix是这种语法中一个可选的部分，用户应该总是为可写和只读S3表包括一个S3前缀，它被用来分隔作为CREATE EXTERNAL TABLE语法一部分的数据集。

对于可写的S3表，s3协议的URL指定Greenplum数据库为该表上传数据文件的端点和桶名。对于上传文件的S3用户ID，该S3桶的权限必须是Upload/Delete。对于每一个由于向表中插入数据形成的新的上传文件，都会使用这个S3文件前缀。见[关于S3数据文件](#)。

对于只读的S3表，S3文件前缀是可选的。如果用户指定了一个S3_prefix，那么s3协议会选择所有以该前缀开始的文件作为该外部表的数据文件。s3协议不使用斜线字符(/)作为界定符，因此跟在前缀后面的一个斜线字符会被当做该前缀本身的一部分。

例如，考虑下面的5个文件，其中的每一个都有名为s3-us-west-2.amazonaws.com的 S3_endpoint以及bucket_name。 test1:

```
s3://s3-us-west-2.amazonaws.com/test1/abc
s3://s3-us-west-2.amazonaws.com/test1/abc/
s3://s3-us-west-2.amazonaws.com/test1/abc/xx
s3://s3-us-west-2.amazonaws.com/test1/abcdef
s3://s3-us-west-2.amazonaws.com/test1/abcdefff
```

- 如果提供的S3 URL是 s3://s3-us-west-2.amazonaws.com/test1/abc，那么 abc前缀会选择全部的5个文件。
- 如果提供的S3 URL是 s3://s3-us-west-2.amazonaws.com/test1/abc/，那么 abc/前缀选择文件 s3://s3-us-west-2.amazonaws.com/test1/abc/ 以及 s3://s3-us-west-2.amazonaws.com/test1/abc/xx。
- 如果提供的S3 URL是 s3://s3-us-west-2.amazonaws.com/test1/abcd，那么 abcd前缀选择文件 s3://s3-us-west-2.amazonaws.com/test1/abcdef 以及 s3://s3-us-west-2.amazonaws.com/test1/abcdefff

S3_prefix中不支持通配符字符。不过，S3前缀本来的功能就好像其后有一个通配符一样。

被S3 URL (S3_endpoint/bucket_name/S3_prefix)选中的所有文件都会被用作外部表的来源，因此它们必须有同样的格式。每个文件还必须包含有完整的数据行。一个数据行不能跨文件存在。对于访问这些S3文件的S3用户ID，文件的权限必须是 Open/Download和View。

有关Amazon S3端点的信息请

见http://docs.aws.amazon.com/general/latest/gr/rande.html#s3_region。有关S3桶和文件夹的信息请见Amazon S3的文档<https://aws.amazon.com/documentation/s3/>。有关S3文件前缀的信息请见Amazon S3的文档[Listing Keys Hierarchically Using a Prefix and Delimiter](#).

config参数指定所需的 s3协议配置文件的位置，该文件包含AWS连接证书和通信参数。见 [关于S3协议的配置参数](#).

关于S3数据文件

对于可写S3表上的每个INSERT操作，Greenplum数据库的每个Segment会使用文件名格式 <prefix><segment_id><random>. <extension>[.gz] 其中：

- <prefix>是在S3 URL中指定的前缀。
- <segment_id>是Greenplum数据库的Segment的ID。
- <random>是一个用来确保文件名唯一的随机数。

- <extension>描述文件类型 (.txt 或 .csv, 取决于用户在CREATE WRITABLE EXTERNAL TABLE的FORMAT子句中提供的值)。 gpcheckcloud工具创建的文件总是使用扩展名 .data。
- 如果为S3可写表启用了压缩 (默认启用), 会在文件名后追加.gz。

对于可写的S3表, 用户可以配置Segment上传文件用的缓冲区尺寸以及线程数量。见[S3协议的配置文件](#)。

对于只读的S3表, 所有通过S3的文件位置

(*S3_endpoint/bucket_name/S3_prefix*) 指定的文件都被用作该外部表的来源并且都必须具有相同的格式。每个文件还必须包含有完整的数据行。如果这些文件含有一个可选的头部行, 头部行中的列名不能包含新行字符 (\n) 以及回车 (\r)。另外, 列定界符也不能是新行字符(\n)或回车 (\r)。

s3协议可以识别gzip格式并且解压这类文件。只支持gzip压缩格式。

对于访问这些S3文件的S3用户ID, 文件的权限必须是Open/Download和View。可写的S3表要求该S3用户ID具有Upload/Delete权限。

对于只读的S3表, 每一个Segment一次可以从S3位置使用几个线程下载一个文件。为了利用Greenplum数据库的Segment所执行的并行处理, S3位置中的文件应该在尺寸上相似并且文件的数量应该允许多个Segment从S3位置下载数据。例如, 如果Greenplum数据库系统由16个Segment组成并且有充足的网络带宽, 在S3位置中创建16个文件允许每个Segment从该S3位置下载一个文件。相反, 如果如果该位置只包含1个或者2个文件, 就只有1个或者2个Segment能下载数据。

S3协议的AWS服务端加密支持

Greenplum数据库支持用可读和可写外部表 (用s3协议创建) 访问的AWS S3文件进行服务端加密, 服务端加密使用Amazon的S3-managed密钥 (SSE-S3)。SSE-S3在用户的对象数据被写入到磁盘时对它们加密, 并且在用户访问它们时透明地解密。

Note: s3协议只对Amazon Web Services的S3文件支持SSE-S3。在访问S3兼容服务上的文件时不支持SSE-S3。

用户的S3 accessid 和 secret权限掌管着用户对所有S3桶对象的访问, 不管该数据是否被加密。但是, 为了访问加密数据, 用户必须配置用户的客户端使用S3-managed密钥。

有关AWS服务端加密的额外信息, 请参考AWS文档中的[使用服务端加密保护数据](#)。

配置S3服务端加密

s3协议的服务端加密默认被禁用。要在使用Greenplum数据库的s3 协议写入

的AWS S3对象上利用服务端加密，用户必须在用户的s3配置文件中把配置参数server_side_encryption设置为值sse-s3：

```
server_side_encryption = sse-s3
```

当用户把一个包括了server_side_encryption = sse-s3 设置的配置文件提供给使用s3的CREATE WRITABLE EXTERNAL TABLE调用时，Greenplum数据库会在该外部表上进行INSERT操作时应用加密头部。然后S3会在写LOCATION子句中提供的URI所标识的对象时进行加密。

在通过用s3协议创建的可读外部表访问加密文件时，S3会透明地解密数据。不需要做额外的配置。

对于更细粒度的加密配置，用户可以考虑创建Amazon Web Services S3的桶策略，按照AWS文档的[Protecting Data Using Server-Side Encryption with Amazon S3-Managed Encryption Keys \(SSE-S3\)](#)部分标识用户想要加密的对象以及在那些对象上的写动作。

s3协议代理支持

您可以指定一个URL，该URL是S3用于连接数据源的代理。S3支持以下协议：HTTP和HTTPS。您可以使用s3协议配置参数代理或环境变量指定代理。如果设置了配置参数，则忽略环境变量。

要使用环境变量指定代理，请根据协议设置环境变量：http_proxy或https_proxy。您可以通过设置适当的环境变量为每个协议指定不同的URL。S3支持这些环境变量。

- all_proxy指定了当特定协议的环境变量没有被设置时的代理URL。
- no_proxy指定了一个通过环境变量设定的，不使用代理的逗号分割的机器名列表。

必须设置环境变量，并且必须可以在所有Greenplum数据库主机上访问Greenplum数据库。

有关配置参数proxy的更多信息，请参考s3协议配置文件[S3协议的配置文件](#)。

关于S3协议的配置参数

可选的config参数指定所需的s3协议配置文件的位置。该文件包含Amazon Web Services (AWS) 的连接证书和通信参数。有关该文件的信息请见[S3协议的配置文件](#)。

在Greenplum数据库的所有Segment主机上都要求这个配置文件。其默认位置是Greenplum数据库每一个Segment实例的数据目录下的一个位置。

```
gpseg_data_dir/gpseg_prefixN/s3/s3.conf
```

*gpseg_data_dir*是该Greenplum数据库Segment的数据目录路径，*gpseg_prefix*是该Segment的前缀，而*N*是该Segment的ID。Segment的数据目录、前缀还有ID都是在初始化一个Greenplum数据库系统时被设置好的。

如果在Segment主机上有多个Segment实例，用户可以通过在每个Segment主机上创建一个位置来简化配置。然后用户可以在s3协议的LOCATION子句中的config参数中指定该位置的绝对路径。这个例子指定了一个位于gpadmin主目录中的位置。

```
LOCATION ('s3://s3-us-west-2.amazonaws.com/test/my_data
config=/home/gpadmin/s3.conf')
```

该主机上的所有Segment实例都会使用文件 /home/gpadmin/s3.conf。

S3协议的配置文件

当使用s3协议时，Greenplum数据库的所有Segment都要求一个s3协议配置文件。默认位置是：

```
gpseg_data_dir/gpseg-prefixN/s3/s3.conf
```

*gpseg_data_dir*是该Greenplum数据库Segment的数据目录路径，*gpseg-prefix*是该Segment的前缀，而*N*是该Segment的ID。Segment的数据目录、前缀还有ID都是在初始化一个Greenplum数据库系统时被设置好的。

如果在Segment主机上有多个Segment实例，用户可以通过在每个Segment主机上创建一个位置来简化配置。然后用户可以在s3协议的LOCATION子句中的config参数中指定该位置的绝对路径。不过，注意只读和可写S3外部表使用相同的参数进行连接。如果用户想为只读和可写S3表配置不同的协议参数，那么用户必须使用两个不同的s3协议配置文件并且在创建每个表时在CREATE EXTERNAL TABLE语句中指定正确的文件。

这个例子指定一个单一文件位置，它位于gpadmin主目录的的s3 目录中：

```
config=/home/gpadmin/s3/s3.conf
```

主机上的所有Segment实例都使用文件 /home/gpadmin/s3/s3.conf。

s3协议配置文件是一个文本文件，它由一个 [default] 小节和参数构成。这里是一个配置文件的例子：

```
[default]
secret = "secret"
accessid = "user access id"
```

```
threadnum = 3
chunksize = 67108864
```

可以使用Greenplum数据库的gpcheckcloud工具来测试S3配置文件。见[使用gpcheckcloud工具](#)。

s3配置文件参数

accessid

必需。访问S3桶的AWS S3 ID。

secret

必需。访问S3桶的AWS S3 ID的密码。

autocompress

对于可写的S3外部表，这个参数指定是否在上传到S3之前（使用gzip）

压缩文件。如果没有指定这个参数，文件会默认被压缩。

chunksize

每个Segment线程用来读写S3服务器的缓冲区尺寸。默认是64MB。最小是8MB而最大是128MB。

在插入数据到可写S3表时，Greenplum数据库的每个Segment把数据写入到其缓冲区（使用最多threadnum个线程）中直到缓冲区被填满，之后它会把该缓冲区写入到S3桶中的一个文件里。根据需要这一处理会在每个Segment上反复执行直到插入操作完成。

因为Amazon S3允许最多10,000个部分的多部分上传，最小8MB的chunksize值就支持在Greenplum数据库的每个Segment上最大80GB的插入量。最大128MB的chunksize值支持每个Segment最大1.28TB的插入量。对于可写的S3表，用户必须确保chunksize的设置能支持表预期的大小。更多有关上传到S3的信息请见S3文档中的[Multipart Upload Overview](#)。

encryption

使用被安全套接字层（SSL）保护的连接。默认值是true。值true, t, on, yes, 以及y(大小写无关)都被当作是true。任何其他值都被当成是false。

如果在CREATE EXTERNAL TABLE命令的LOCATION子句中的URL没有指定端口，配置文件的 encryption参数会影响s3协议使用的端口

（HTTP是端口80，HTTPS是端口443）。如果指定了端口，不管encryption设置是什么都会使用那个端口。

gpcheckcloud_newline

从S3位置下载文件时，如果文件的最后一行没有EOL（行尾）字符，则gpcheckcloud实用程序会将新行字符附加到文件的最后一行。默认字符是\n（换行符）。值可以是\n, \r\n\n（回车）或\n\r（换行符/回车符）。

添加EOL字符可防止一个文件的最后一行与下一个文件的第一行连在一起。

low_speed_limit

上传/下载速度下限，以字节每秒为单位。默认速度是10240 (10K)。
如果上传或者下载速度低于该限制超过`low_speed_time`所指定的时间，那么该连接会被中止并且重试。在3次重试后，S3协议会返回一个错误。指定值为0表示没有下限。

low_speed_time

当连接速度低于`low_speed_limit`时，这个参数指定在中止对S3桶上传或者下载之前应等待的总时间（以秒为单位）。默认值是60秒。指定值为0表示没有时间限制。

proxy

指定一个URL，该URL是S3用于连接数据源的代理。S3支持以下协议：HTTP和HTTPS。这是参数的格式。

```
proxy = protocol://[user:password@]proxyhost[:port]
```

如果未设置此参数或为空字符串 (`proxy = ""`)，则S3使用由环境变量`http_proxy`或`https_proxy`（以及环境变量`all_proxy`和`no_proxy`）指定的代理。S3使用的环境变量取决于协议。有关环境变量的信息，请参阅Greenplum数据库管理向导的[S3协议代理支持](#)。

配置文件中最多只能有一个`proxy`参数。参数指定的URL是所有支持的协议的代理。

server_side_encryption

为桶配置的S3服务端加密方法。Greenplum数据库只支持使用Amazon S3-managed密钥的服务端加密，由配置参数值`sse-s3`表示。默认情况下服务端加密被禁用（值为`none`）。

threadnum

对S3桶上传或者下载数据时，一个Segment能够创建的并发线程的最大数量。默认值是4。最小值是1且最大值是8。

verifycert

控制在HTTPS上建立客户端和S3数据源之间加密通信时S3协议如何处理认证。其值可以是`true`或者`false`。默认值是`true`。

- `verifycert=false` - 忽略认证错误并且允许HTTPS之上的加密通信。
- `verifycert=true` - 要求合法的认证（一个正确的证书）用于HTTPS之上的加密通信。

在测试和开发环境中设置这个值为`false`有助于允许无需更改证书的通信。

Warning: 设置这个值为`false`会在建立客户端和S3数据存储之间的通信时忽略证书，这会暴露出安全性风险。

version

指定在CREATE EXTERNAL TABLE命令的LOCATION子句中指定的信息的版本。值可以是1或2。默认值是1。

如果值是1，则LOCATION子句支持Amazon S3 URL并且不含region参数。如果值是2，则LOCATION支持S3兼容服务并且必须包括region参数。region参数指定S3数据源的地区。对于下面这个S3 URL

`s3://s3-us-west-`

`2.amazonaws.com/s3test.example.com/dataset1/normal/`,
其AWS S3地区是us-west-2。

如果version是1或者没有指定, 下面的CREATE EXTERNAL TABLE中的LOCATION子句例子指定了一个Amazon S3端点。

```
LOCATION ('s3://s3-us-west-2.amazonaws.com/s3test.example.com/dataset1/normal/config=/home/gpadmin/aws_s3/s3.conf')
```

如果version是2, 下面的带region参数的 LOCATION(CREATE EXTERNAL TABLE命令的子句) 例子指定了一个AWS S3兼容服务。

```
LOCATION ('s3://test.company.com/s3test.company/test1/normal/region=local-test config=/home/gpadmin/aws_s3/s3.conf')
```

如果version是2, LOCATION子句还能指定一个Amazon S3端点。这个例子指定了一个使用region参数的Amazon S3端点。

```
LOCATION ('s3://s3-us-west-2.amazonaws.com/s3test.example.com/dataset1/normal/region=us-west-2 config=/home/gpadmin/aws_s3/s3.conf')
```

Note: 在上传或者下载S3文件时, Greenplum数据库在每个Segment主机上可能要求最多`threadnum * chunkszie`的内存。在配置Greenplum数据库的总内存时, 请考虑这种s3 协议的内存需求。

S3协议的限制

s3协议有一些限制:

- 只支持S3路径风格的URL。

```
s3://S3_endpoint/bucketname/[S3_prefix]
```

- 只支持S3端点。该协议不支持S3桶的虚拟主机 (绑定一个域名给一个S3桶)。
- 支持版本4的AWS签名的签名处理。
有关每种签名处理支持的S3端点的信息, 请见http://docs.aws.amazon.com/general/latest/gr/rande.html#s3_region □
- 在CREATE EXTERNAL TABLE命令的LOCATION 子句中只支持一个单一URL和可选的配置文件。
- 如果在CREATE EXTERNAL TABLE命令中没有指定NEWLINE参数, 新行字符在特定前缀的所有数据文件中都必须一样。如果在同一个前缀的某些数据文件中有不一样的新行字符, 在这些文件上的读操作可能会失败。
- 对于可写的S3外部表, 只支持INSERT操作。不支持UPDATE, DELETE, 和TRUNCATE操作。

- 因为Amazon S3允许做多10,000个部分的多部分上传，最大128MB的chunksize 值在Greenplum数据库的每个Segment上支持对可写S3表最大1.28TB的插入量。用户必须确保chunksize的设置能支持表预期的大小。更多有关上传到S3的信息请见S3文档中的[Multipart Upload Overview](#)。
- 为了利用Greenplum数据库的Segment所执行的并行处理，S3位置中的文件应该在尺寸上相似并且文件的数量应该允许多个Segment从S3位置下载数据。例如，如果Greenplum数据库系统由16个Segment组成并且有充足的网络带宽，在S3位置中创建16个文件允许每个Segment从该S3位置下载一个文件。相反，如果如果该位置只包含1个或者2个文件，就只有1个或者2个Segment能下载数据。

使用gpcheckcloud工具

Greenplum数据库的工具gpcheckcloud帮助用户创建一个s3 协议配置文件并且测试一个配置文件。用户可以用指定选项来测试用一个配置文件访问S3桶的能力，并且可以选择对该桶中的文件上传或者下载数据。

如果不用任何选项运行该工具，它会发送一个模板配置文件到STDOUT。用户可以捕捉该输出并且创建一个连接到Amazon S3的s3配置文件。

该工具被安装在Greenplum数据库的 \$GPHOME/bin 目录中。

语法

```
gpcheckcloud {-c | -d} "s3://S3_endpoint/bucketname/[S3_prefix]
[config=path_to_config_file]"

gpcheckcloud -u <file_to_upload>
"s3://S3_endpoint/bucketname/[S3_prefix]
[config=path_to_config_file]"
gpcheckcloud -t

gpcheckcloud -h
```

选项

-c

s3协议URL中指定的配置连接到指定的S3位置并且返回其中的文件信息。

如果连接失败，该工具显示关于失败的信息，例如非法的证书、前缀或者服务器地址（DNS错误）或服务器不可用。

-d

用s3协议URL中指定的配置从指定的S3位置下载数据并且把输出发送到 STDOUT。

如果文件被gzip压缩过，解压后的数据会被发送到 STDOUT。

-u

使用指定的配置文件（如果有）上传一个文件到s3协议URL中指定

的S3桶中。使用这个选项可以测试压缩和配置中的`chunkszie` 和 `autocompress` 设置。

-t

发送一个模板配置文件到STDOUT。用户可以捕捉该输出并且创建一个连接到Amazon S3的s3配置文件。

-h

显示gpcheckcloud帮助。

例子

这个例子运行不带选项的工具在当前目录下创建一个模板 s3配置文件`mytest_s3.config`。

```
gpcheckcloud -t > ./mytest_s3.config
```

这个例子尝试使用s3配置文件`s3.mytestconf` 上传一个本地文件`testdata.csv`到一个S3桶位置：：

```
gpcheckcloud -u ./testdata.csv "s3://s3-us-west-2.amazonaws.com/test1/abc config=s3.mytestconf"
```

一次成功的上传会导致在S3桶中出现一个或者更多个以文件名格式`abc<segment_id><random>.data[.gz]`命名的文件。见[关于S3数据文件](#)。

这个例子尝试使用s3配置文件`s3.mytestconf`连接到一个S3桶位置。

```
gpcheckcloud -c "s3://s3-us-west-2.amazonaws.com/test1/abc config=s3.mytestconf"
```

从该S3桶位置下载所有文件并且发送输出到STDOUT。

```
gpcheckcloud -d "s3://s3-us-west-2.amazonaws.com/test1/abc config=s3.mytestconf"
```

Parent topic: [定义外部表](#)

像

Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统
 - 关于Greenplum数据库发布版本号
 - 启动和停止Greenplum数据库
 - 访问数据库
 - 配置Greenplum数据库系统
- 启用压缩
- 启用高可用和数据持久化特征
 - Greenplum数据库高可用性概述
 - 在Greenplum数据库中启用镜像
 - 检测故障的Segment
 - 恢复故障Segment
 - 恢复故障的Master
- 备份和恢复数据库
- 扩容Greenplum系统
- 监控Greenplum系统
- 日常系统维护任务

可以在建立Greenplum数据库系统时使用`gpinitsystem`配置系统使用镜像，或者之后用`gpaddmirrors`以及`gpinitstandby`启用镜像。这个主题假定现有系统初始化时没有镜像，现在要向其中加入镜像。

- 启用**Segment**镜像
- 启用**Master**镜像

Parent topic: [启用高可用和数据持久化特征](#)



Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统
 - 关于Greenplum数据库
发布版本号
 - 启动和停
止Greenplum数据库
 - 访问数据库
 - 配置Greenplum数据库
系统
 - 启用压缩
 - 启用高可用和数据持久
化特征
 - Greenplum数据库高
可用性概述
 - 在Greenplum数据库
中启用镜像
 - 检测故障的Segment
 - 恢复故障Segment
 - 恢复故障的Master
 - 备份和恢复数据库
 - 扩容Greenplum系统
 - 监控Greenplum系统
 - 日常系统维护任务

如果启用了镜像，Greenplum数据库会在主Segment宕机后自动故障转移到一个镜像Segment上，镜像Segment承担主Segment的角色和职能，故障主Segment变成镜像，用户感觉不到segment产生了故障。当故障出现时，正在进行中的事务会回滚并在新的segment上自动重新开始。[gpstate](#)工具可以用来分辨故障segment。该工具显示的系统表信息包括[gp_segment_configuration](#)。

如果整个Greenplum数据库系统由于一个Segment故障（例如，如果没有启用镜像或者没有足够的Segment在线以访问全部用户数据）而变得无法运转，用户在尝试连接到数据库时会看到错误。返回给客户端程序的错误可能表明失效。例如：

```
ERROR: All segment databases are unavailable
```

segment故障如何被监测和管理

在Greenplum数据库master主机上，Postgres的postmaster进程会创建一个错误侦测子进程 ftsprobe。该进程也被称为FTS (Fault Tolerance Server)进程。如果FTS进程出现故障，postmaster会重启该进程。

FTS进程循环执行，每个循环之间会停顿一会。每一个循环中，FTS都会从[gp_segment_configuration](#)系统表中获取每一个主segment实例的主机名和端口号，并通过与其建立TCP套接字连接的方式连接segment实例来侦测segment的状态。如果成功连接，segment会执行一些简单的检查并报告给FTS。该检查包括在关键的segment目录上执行一个stat系统调用，并且会检查segment实例的内部错误。如果没有检测到问题，FTS会收到一个积极的反馈信号，被检查的正常segment也不会采取任何附加操作。

如果不能建立连接，或者在超时时间内没有得到反馈，FTS会尝试重新连接segment实例。如果达到了FTS的最大重试次数，FTS会去检测该segment的镜像看其是否在线，如果在线它就会修改[gp_segment_configuration](#)表，将原来的主segment标记为"down"并设置镜像承担主segment的角色。FTS会把执行过的操作更新到[gp_configuration_history](#)表中。

当系统中只有主segment正常，其对应的镜像故障时，主segment会进入*not synchronizing*状态并继续记录数据库日志变化，一旦镜像修
segment

Recommended

复，便可以将这些变化继续同步，而不用从主拷贝。

执行一个完整的

运行gpstate工具并带有-e选项可以展示单个主segment和镜像segment实例 的任何问题。gpstate的另外一些选项也能显示所有主segment或镜像segment实例的信息，例如 -m (镜像实例信息) 、 -c (主segment和镜像segment配置信息) ，另外也可以显示主segment和镜像segment的问题。

您也可以从系统表gp_segment_configuration查看当前模式： s (同步状态)n (非同步状态)，还有当前状态 u (在线) or d (离线)。

[gprecoverseg](#)工具可以用于将离线的镜像segment重新加入集群。默认情况下， gprecoverseg会执行增量恢复，首先将镜像放入同步模式，接下来会重放主segment上 记录下来的日志变化到镜像segment。如果增量恢复失败，整个恢复便会失败。此时可以执行gprecoverseg 并带有-F选项，以执行一个全量恢复，该操作会复制主segment上的所有数据到镜像segment上。

segment实例恢复后， gpstate -e命令可以列出所有切换了的主segment和镜像segment实例信息。该信息表示系统目前处于不均衡状态（主实例和镜像实例不在他们的初始配置角色）。如果系统目前处于不均衡状态，在segment 主机系统上可能存在活动主segment实例分布不均匀的结果。

系统表gp_segment_configuration包含列role和 preferred_role。该列上的值为p代表主segment， m 代表镜像segment。role列展示segment实例当前角色， preferred_role 列展示segment实例原来的角色。

在一个平衡的系统中，所有segment实例的role和preferred_role列都是对应一致的。当某一个列存在不一样的情况时，表明系统是不平衡的。为了重新平衡集群并将所有的segment恢复到他们本来的角色，可以运行gprecoverseg命令并带有-r选项。

简单的故障转移和恢复示例

本例子已经假设一个单独的主-镜像segment实例对中主segment故障并切换到镜像segment。以下表格展示在恢复到正常的主segment之前， segment实例在gp_segment_configuration 表中的本来角色，当前角色，模式，和状态：

也可以通过执行gpstate -e来显示主segment和镜像segment的状态

信息

	本来角色	当前角色	模式	状态
Primary	p (primary)	m (mirror)	n (not synchronizing)	d (down)
Mirror	m (mirror)	p (primary)	n (not synchronizing)	u (up)

segment实例不在他们原来的最佳角色，主segment故障。镜像segment上线成为主segment，目前为非同步模式，因为他的镜像（之前失败的主segment）处于故障状态。当修复了故障问题后，使用gprecoverseg重新将失败的实例拉起并在主segment和镜像segment之间同步差异数据。

一旦gprecoverseg操作完成，segment的状态如下表展示，主segment和镜像segment转换成为他们之前的最佳角色。

	本来角色	当前角色	模式	状态
Primary	p (primary)	m (mirror)	s (synchronizing)	u (up)
Mirror	m (mirror)	p (primary)	s (synchronizing)	u (up)

gprecoverseg -r命令通过将segment的角色切换回他们之前的最佳角色来重新平衡整个集群系统。

	本来角色	当前角色	模式	状态
Primary	p (primary)	p (primary)	s (synchronized)	u (up)
Mirror	m (mirror)	m (mirror)	s (synchronized)	u (up)

配置FTS行为

有一系列的服务器配置参数会影响FTS行为：

gp_fts_probe_interval

开始下一个FTS循环的时间间隔，以秒为单位。例如将该参数设置为60，检测循环持续10秒，那么FTS进程沉睡50秒。如果设置该参数为60，检测循环持续75秒，FTS进程沉睡0秒。该参数默认值为60，最大为3600。

gp_fts_probe_timeout

master到segment之间的检测超时时间，以秒为单位。默认20秒，最大3600秒。

gp_fts_probe_retries

检测segment失败后的重试次数，例如如果设置为5，那么首次失败后，会再进行4次检测尝试。默认为5。

gp_log_fts

FTS日志级别。可设置的值有："off", "terse", "verbose", or "debug"。"verbose"设置可以用在生产上，能为问题定位提供一些有用的数据。"debug"设置不能用在生产上。默认为："terse"。

gp_segment_connect_timeout

允许镜像响应的最大时间，以秒为单位。默认为：600（10分钟）

除了可以使用FTS执行错误检查外，不能发送数据到其镜像segment的主segment可以改变镜像的状态为down。主segment会将同步数据暂时排队处理，如果gp_segment_connect_timeout时间超时，此时意味着镜像故障，系统会标记镜像为down并将主segment变为变化追踪（change tracking）模式。

- [检测故障Segment](#)
- [检查日志文件](#)

Parent topic: [启用高可用和数据持久化特征](#)

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库
发布版本号启动和停
止Greenplum数据库

□ 访问数据库

□ 配置Greenplum数据库
系统

启用压缩

□ 启用高可用和数据持久
化特征□ Greenplum数据库高
可用性概述□ 在Greenplum数据库
中启用镜像

□ 检测故障的Segment

□ **恢复故障Segment**

□ 恢复故障的Master

□ 备份和恢复数据库

□ 扩容Greenplum系统

监控Greenplum系统

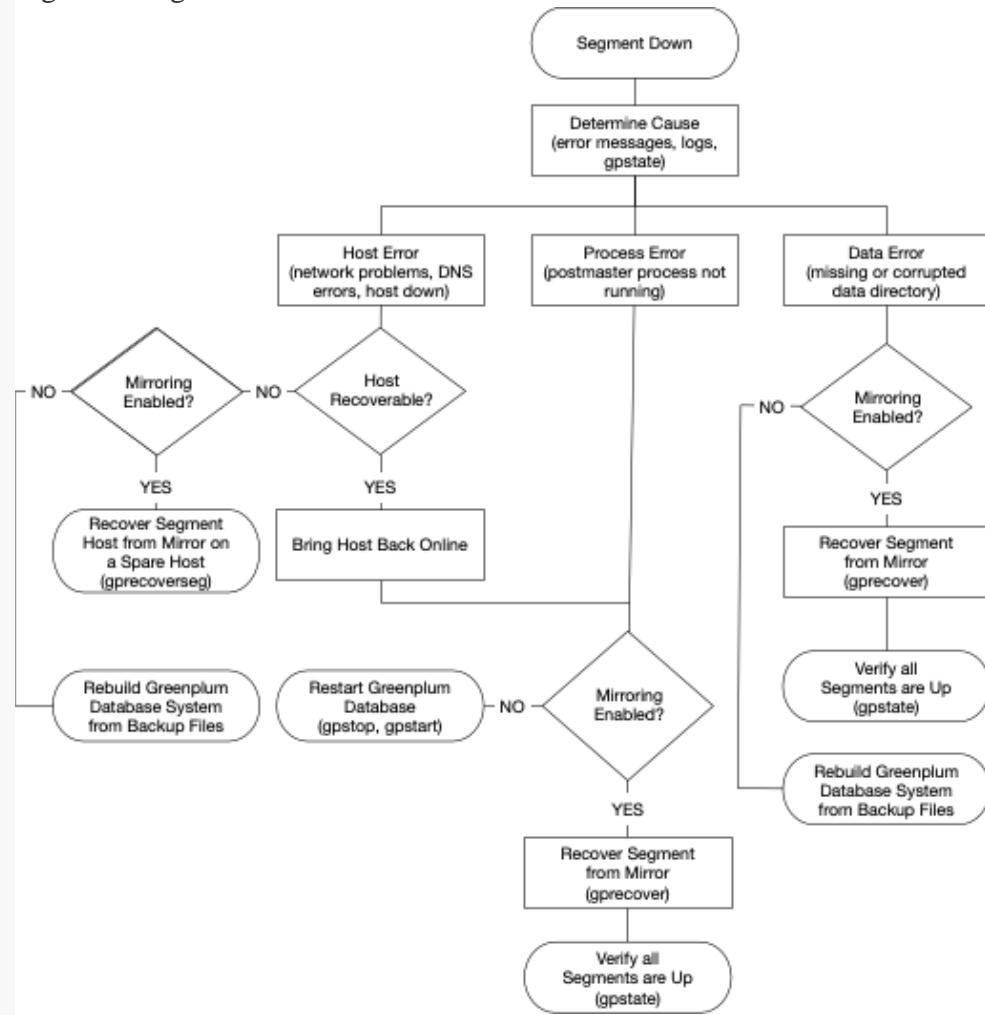
日常系统维护任务

如果Master无法连接到一个Segment实例，它会在Greenplum数据库的系统表中把该Segment标记为“down”。该Segment实例会保持离线状态直到管理员采取步骤让它重新回到线上。恢复一个失效Segment实例或者主机的处理取决于失效原因以及是否启用了镜像。一个Segment实例的故障原因多种多样：

- Segment主机不可用，例如由于网络或者硬件失效。
- Segment实例没有运行，例如没有postgres数据监听器进程。
- Segment实例的数据目录损坏或者丢失，例如数据不可访问、文件系统损坏或者磁盘失效。

[Figure 1](#) 展示了前述失效场景的高层排查步骤。

Figure 1. Segment失效故障排查矩阵



- [从Segment故障中恢复](#)

Parent topic: [启用高可用和数据持久化特征](#)



Greenplum数据库® 6.0文档

□ 管理员指南

- Greenplum数据库概念
- 管理一个Greenplum系统

关于Greenplum数据库
发布版本号启动和停
止Greenplum数据库

- 访问数据库
- 配置Greenplum数据库
系统
- 启用压缩
- 启用高可用和数据持久
化特征

□ Greenplum数据库高
可用性概述□ 在Greenplum数据库
中启用镜像

□ 检测故障的Segment

□ 恢复故障Segment

□ 恢复故障的Master

□ 备份和恢复数据库

□ 扩容Greenplum系统

监控Greenplum系统

日常系统维护任务

如果主Master故障，Greenplum数据库不可访问且WAL复制会停止。使用[gpactivatestandby](#)来激活后备Master。在激活后备Master过程中，Greenplum数据库会重构Master主机为最后一次成功提交事务时的状态。

以下步骤假设系统中已经配置了standby master主机。详见[启用Master镜像](#)。

激活standby master

1. 在standby master主机上运行gpactivatestandby工具 来激活它。例如：

```
$ gpactivatestandby -d /data/master/gpseg-1
```

此处-d选项指定正在激活的master主机的数据目录。
激活standby后，状态变为*active*或*primary* master

2. 上面工具执行完成后，运行gpstate带有-b 选项来显示系统汇总信息：

```
$ gpstate -b
```

master实例状态应该为Active。如果没有配置standby master，该命令会显示standby master的状态为 No master standby configured。如果配置了standby master，它的状态为 Passive。

3. 在切换到最新的活动Master主机后，在其上运行 ANALYZE 例如：

```
$ psql dbname -c 'ANALYZE;'
```

4. 可选：如果激活之前的standby master时没有配置一个新的standby master。可以运行 gpinitstandby工具来配置激活一个新的standby master。

Important: 必须初始化一个新的standby master以继续为master提供镜像。

关于恢复原来master和standby master的详细配置方法，请见[在恢复后还原Master镜像](#)。

Recommended

- 在恢复后还原**Master**镜像

Parent topic: 启用高可用和数据持久化特征

Greenplum数据库® 6.0文档

- 管理员指南
 - Greenplum数据库概念
 - 管理一个Greenplum系统

关于Greenplum数据库
发布版本号启动和停
止Greenplum数据库

- 访问数据库
- 配置Greenplum数据库
系统
- 启用压缩
- 启用高可用和数据持久
化特征
- 备份和恢复数据库

备份和恢复概述

- 使用
gpbackup和gprestore并
行备份
 - 扩容Greenplum系统
- 监控Greenplum系统
- 日常系统维护任务
- Recommended Monitoring and Maintenance Tasks
- 管理Greenplum数据库访问

Greenplum数据库支持并行和非并行的方法来备份和恢复数据库。并行操作的规模不受系统中Segment数量的影响，因为每台Segment主机都同时把其数据写入到本地的磁盘存储上。如果使用非并行备份和恢复操作，数据必须通过网络从Segment被发送到Master，后者把所有的数据写入它的存储中。除了把I/O限制在一台主机上之外，非并行备份要求Master拥有足够的本地磁盘存储以保存整个数据库。

使用gpbackup和gprestore做并行备份

gpbackup和gprestore是Greenplum数据库的备份和恢复工具。gpbackup在每个独立的表级别使用ACCESS SHARE锁，而不是在pg_class catalog表里加EXCLUSIVE锁。这使得你可以在backup期间执行DML语句，如CREATE, ALTER, DROP和TRUNCATE操作，只要这些操作没有执行在备份的数据上。

使用gpbackup创建的备份文件旨在提供将来恢复单个数据库对象及其依赖项的功能，例如函数和所需的用户定义数据类型。参考[使用gpbackup和gprestore并行备份](#)获取更多信息。

使用pg_dump做非并行备份

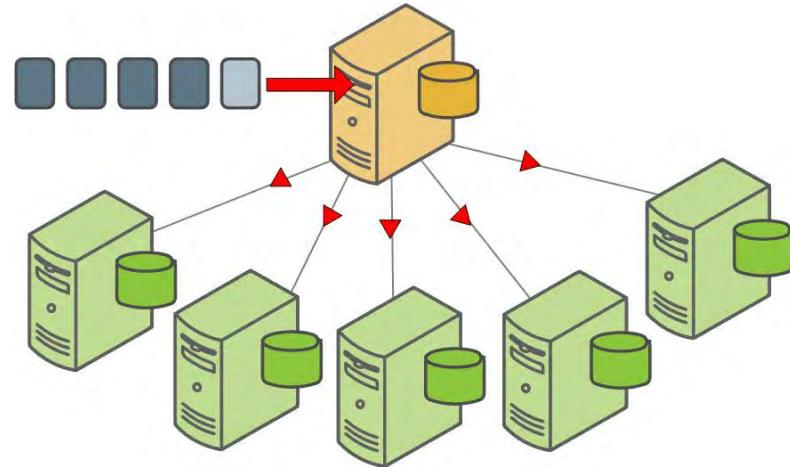
PostgreSQL pg_dump和pg_dumpall非并行备份工具可以用来在master主机上创建一个单独的，包含所有节点数据的dump文件。

PostgreSQL的非并行工具应该在特殊场合使用。它们比使用Greenplum backup工具要慢得多，因为所有数据都必须通过master。此外，通常情况是master主机没有足够的磁盘空间来保存整个分布式Greenplum数据库的备份。

pg_restore工具需要pg_dump或pg_dumpall创建的压缩dump文件。在开始还原之前，应修改dump文件中的CREATE TABLE语句以包含Greenplum DISTRIBUTED子句。如果不包含DISTRIBUTED子句，Greenplum数据库会分配默认值，这可能不是最佳值。有关详细信息，请参阅Greenplum数据库参考指南中的CREATE TABLE。

使用并行的备份文件来做一个非并行的恢复，可以从每一个segment节点拷贝备份文件到master节点，然后通过master加载它们。

Figure 1. 使用并行的备份文件来做一个非并行的恢复



备份Greenplum数据库数据的另一种非并行方法是使用COPY TO SQL命令将数据库中的全部或部分表复制到master主机上的分隔文本文件。

Parent topic: [备份和恢复数据库](#)

Greenplum数据库® 6.0文档

- 管理员指南

- Greenplum数据库概念

- 管理一个Greenplum系统

- 关于Greenplum数据库发布版本号

- 启动和停止Greenplum数据库

- 访问数据库

- 配置Greenplum数据库系统

- 启用压缩

- 启用高可用和数据持久化特征

- 备份和恢复数据库

- 备份和恢复概述

- 使用gpbackup和gprestore并行备份

- 扩容Greenplum系统

- 监控Greenplum系统

- 日常系统维护任务

- Recommended Monitoring and Maintenance Tasks

- 管理Greenplum数据库访问

- 定义数据库对象

- 分布与倾斜

- 插入、更新、和删除数据

gpbackup和gprestore是Greenplum数据库工具，可为Greenplum数据库创建和恢复备份集。默认情况下，gpbackup仅在Greenplum数据库master数据目录中存储备份的对象元数据文件和DDL文件。Greenplum数据库节点使用COPY ... ON SEGMENT命令将备份表的数据存储在位于每个节点的backups目录中的压缩CSV数据文件中。

元数据备份文件包含gprestore并行恢复完整备份集所需的所有信息。备份元数据还提供了在gprestore的未来版本中仅用于还原数据集中的单个对象以及任何依赖对象的框架。（参考[理解备份文件](#)获取更新信息。）将表数据存储在CSV文件中还提供了使用其他恢复工具（如gpload）在同一群集或其他群集中加载数据的机会。默认情况下，为节点上的每个表创建一个文件。您可以使用gpbackup指定--leaf-partition-data选项，以便为分区表的每个叶子分区创建一个数据文件，而不是单个文件。此选项还允许您按叶分区筛选备份集。

每个gpbackup任务都使用Greenplum数据库中的单个事务。在此事务期间，元数据将在master主机上备份，并且每个节点主机上的每个表的数据将使用COPY ... ON SEGMENT命令并行写入CSV备份文件。备份进程在备份的每个表上获取ACCESS SHARE锁。

关于gpbackup和gprestore工具的更多选项，参考[gpbackup](#) 和 [gprestore](#)。

- [需求和限制](#)

- [备份或还原中包含的对象](#)

- [执行基本备份和还原操作](#)

- [过滤备份或恢复的内容](#)

- [配置邮件通知](#)

- [理解备份文件](#)

- [使用gpbackup和gprestore创建增量备份](#)

- [将gpbackup和gprestore与BoostFS一起使用](#)

- [使用gpbackup存储插件](#)

- [备份/恢复存储插件API \(Beta版\)](#)

Parent topic: [备份和恢复数据库](#)

需求和限制



gpbackup和gprestore工具在Greenplum数据库5.5.0和之后的版本可用。

gpbackup和gprestore有如下限制：

- 如果在父分区表上创建索引，则gpbackup不会在父分区的子分区表上备份相同的索引，因为在子分区上创建相同的索引会导致错误。但是，如果您交换分区，则gpbackup不会检测到交换分区上的索引是从新父表继承的。在这种情况下，gpbackup备份冲突的CREATE INDEX语句，这会在还原备份集时导致错误。
- 您可以执行gpbackup的多个实例，但每次执行都需要不同的时间戳。
- 数据库对象过滤目前仅限于schema和表。
- 如果使用gpbackup --single-data-file选项将表备份组合到每个节点的单个文件中，则无法使用gprestore执行并行还原操作（无法将--jobs设置为大于1的值）。
- 您不能将--exclude-table-file与--leaf-partition-data一起使用。虽然您可以在使用--exclude-table-file指定的文件中指定叶子分区名称，但gpbackup会忽略分区名称。
- 在运行DDL命令的同时使用gpbackup备份数据库可能会导致gpbackup失败，以确保备份集内的一致性。例如，如果在备份操作开始后删除了表，则gpbackup将

退出并显示错误消息ERROR: relation <schema.table> does not exist。

由于表锁定问题，在备份操作期间删除表时，gpbackup可能会失败。

gpbackup生成要备份的表列表，并在表上获取ACCESS SHARED锁。如果表上保留了EXCLUSIVE LOCK，则gpbackup会在释放现有锁后获取ACCESS SHARED锁。如果gpbackup尝试获取表上的锁时表不再存在，则gpbackup将退出并显示错误消息。

对于可能在备份期间删除的表，可以使用gpbackup表过滤选项（例如--exclude-table或--exclude-schema）从备份中排除表。

- 使用gpexpand创建的备份只能还原到与源集群具有相同数量的节点实例的Greenplum数据库集群。如果运行gpexpand将节点添加到集群，则在扩容完成后无法恢复在扩容之前所做的备份。

Parent topic: [使用gpbackup和gprestore并行备份](#)

备份或还原中包含的对象

下表列出了使用gpbackup和gprestore备份和还原的对象。使用--dbname选项为您指定备份的数据库对象。默认情况下也会备份全局对象（Greenplum数据库系统对象），但只有在gprestore中包含--with-globals选项时才会还原它们。

Table 1. 备份和还原的对象

数据库(使用--dbname指定的数据库)	全局(需要指定--with-globals选项来还原)
<ul style="list-style-type: none"> 会话级别的配置参数设置(GUCs) Schemas, 参考Note 过程语言扩展 序列 注释 表 Indexes 所有者 可写外部表(DDL only) 可读外部表(DDL only) 函数 聚合 类型转换 类型 视图 协议 触发器。（虽然Greenplum数据库不支持触发器，但会备份和恢复任何存在的触发器定义。） 规则 域 运算符，运算符族和运算符类 转换 扩展 文本搜索解析器，词典，模板和配置 	<ul style="list-style-type: none"> 表空间 数据库 数据库范围的配置参数设置(GUCs) Resource group定义 Resource queue定义 角色 GRANT分配给数据库的角色

Note: 这些schemas不包含在备份中。

- gp_toolkit
- information_schema
- pg_aoseg
- pg_bitmapindex
- pg_catalog
- pg_toast*
- pg_temp*

还原到已存在数据库时，`gprestore`假定在将对象还原到public schema时存在public schema。还原到新数据库（使用`--create-db`选项）时，`gprestore`会在使用`CREATE DATABASE`命令创建数据库时自动创建public schema。该命令使用包含public schema的`template0`数据库。

也可以参考[理解备份文件](#)。

Parent topic: [使用gpbackup和gprestore并行备份](#)

执行基本备份和还原操作

要执行数据库的完整备份以及Greenplum数据库系统元数据，请使用以下命令：

```
$ gpbackup --dbname <database_name>
```

例如：

```
$ gpbackup --dbname demo
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-
Starting backup of database demo
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-
Backup Timestamp = 20180105112754
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-
Backup Database = demo
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-
Backup Type = Unfiltered Compressed Full Backup
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-
Gathering list of tables for backup
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-
Acquiring ACCESS SHARE locks on tables
Locks acquired: 6 / 6
[=====
100.00% 0s
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-
Gathering additional table metadata
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-
Writing global database metadata
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-
Global database metadata backup complete
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-
Writing pre-data metadata
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-
Pre-data metadata backup complete
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-
Writing post-data metadata
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-
Post-data metadata backup complete
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-
Writing data to file
Tables backed up: 3 / 3
[=====] 100.00%
0s
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-
Data backup complete
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-
Found neither /usr/local/greenplum-db/.bin/gp_email_contacts.yaml nor
/home/gpadmin/gp_email_contacts.yaml
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-
Email containing gpbackup report /gpmaster/seg-
```

```
1/backups/20180105/20180105112754/gpbackup_20180105112754_report will not
be sent
20180105:11:27:55 gpbackup:gadmin:centos6.localdomain:002182-[ INFO] :-
Backup completed successfully
```

上面的命令在Greenplum数据库master主机的默认目录中创建一个包含全局和数据库特定元数据的文件，`$MASTER_DATA_DIRECTORY/backups/<YYYYMMDD>/<YYYYYMMDDHHMMSS>/`。

例如：

```
$ ls /gpmaster/gpsne-1/backups/20180105/20180105112754
gpbackup_20180105112754_config.yaml    gpbackup_20180105112754_report
gpbackup_20180105112754_metadata.sql    gpbackup_20180105112754_toc.yaml
```

默认情况下，每个节点用一个独立的压缩CSV文件

在`<seg_dir>/backups/<YYYYMMDD>/<YYYYYMMDDHHMMSS>/`存储备份的每个表的数据：

```
$ ls /gpdata1/gpsne0/backups/20180105/20180105112754/
gpbackup_0_20180105112754_17166.gz  gpbackup_0_20180105112754_26303.gz
gpbackup_0_20180105112754_21816.gz
```

要将所有备份文件合并到一个目录中，请包含`--backup-dir`选项。请注意，您必须使用此选项指定绝对路径：

```
$ gpbackup --dbname demo --backup-dir /home/gpadmin/backups
20171103:15:31:56 gpbackup:gadmin:0ee2f5fb02c9:017586-[ INFO] :-
Starting
backup of database demo
...
20171103:15:31:58 gpbackup:gadmin:0ee2f5fb02c9:017586-[ INFO] :-
Backup
completed successfully
$ find /home/gpadmin/backups/ -type f
/home/gpadmin/backups/gpseg0/backups/20171103/20171103153156/gpbackup_0_201*
/home/gpadmin/backups/gpseg0/backups/20171103/20171103153156/gpbackup_0_201*
/home/gpadmin/backups/gpseg1/backups/20171103/20171103153156/gpbackup_1_201*
/home/gpadmin/backups/gpseg1/backups/20171103/20171103153156/gpbackup_1_201*
...
/home/gpadmin/backups/gpseg-
1/backups/20171103/20171103153156/gpbackup_20171103153156_config.yaml
/home/gpadmin/backups/gpseg-
1/backups/20171103/20171103153156/gpbackup_20171103153156_podata.sql
/home/gpadmin/backups/gpseg-
1/backups/20171103/20171103153156/gpbackup_20171103153156_global.sql
/home/gpadmin/backups/gpseg-
1/backups/20171103/20171103153156/gpbackup_20171103153156_postdata.sql
/home/gpadmin/backups/gpseg-
1/backups/20171103/20171103153156/gpbackup_20171103153156_report
/home/gpadmin/backups/gpseg-
1/backups/20171103/20171103153156/gpbackup_20171103153156_toc.yaml
```

执行备份操作时，可以在多个文件的额外开销可能过高的情况下使用`--single-data-file`。例如，如果您使用第三方存储解决方案，例如带备份的Data Domain。

从备份恢复

使用gprestore从备份集合恢复，必须使用`--timestamp`选项指定准确的时间戳值(YYYYMMDDHHMMSS)。包括`--create-db`选项，如果数据库未在集群中创建。例如：

```
$ dropdb demo
$ gprestore --timestamp 20171103152558 --create-db
20171103:15:45:30 gprestore:gadmin:0ee2f5fb02c9:017714-[ INFO] :-
Restore
Key = 20171103152558
20171103:15:45:31 gprestore:gadmin:0ee2f5fb02c9:017714-[ INFO] :-
Creating
database
```

```

20171103:15:45:44 gprestore:gadmin:0ee2f5fb02c9:017714-[INFO]:-Database
creation complete
20171103:15:45:44 gprestore:gadmin:0ee2f5fb02c9:017714-[INFO]:-Restoring
pre-data metadata from /gpmaster/gpsne-
1/backups/20171103/20171103152558/gpbackup_20171103152558_pedata.sql
20171103:15:45:45 gprestore:gadmin:0ee2f5fb02c9:017714-[INFO]:-Pre-data
metadata restore complete
20171103:15:45:45 gprestore:gadmin:0ee2f5fb02c9:017714-[INFO]:-Restoring
data
20171103:15:45:45 gprestore:gadmin:0ee2f5fb02c9:017714-[INFO]:-Data
restore complete
20171103:15:45:45 gprestore:gadmin:0ee2f5fb02c9:017714-[INFO]:-Restoring
post-data metadata from /gpmaster/gpsne-
1/backups/20171103/20171103152558/gpbackup_20171103152558_postdata.sql
20171103:15:45:45 gprestore:gadmin:0ee2f5fb02c9:017714-[INFO]:-Post-data
metadata restore complete

```

如果指定自定义的--backup-dir来合并备份文件，使用gprestore时指定相同的--backup-dir选项来定位备份文件：

```

$ dropdb demo
$ gprestore --backup-dir /home/gadmin/backups/ --timestamp
20171103153156 --create-db
20171103:15:51:02 gprestore:gadmin:0ee2f5fb02c9:017819-[INFO]:-Restore
Key = 20171103153156
...
20171103:15:51:17 gprestore:gadmin:0ee2f5fb02c9:017819-[INFO]:-Post-data
metadata restore complete

```

gprestore默认不会为Greenplum系统尝试恢复全局元数据。如果这个是必须的，需要--with-globals参数。

默认情况下，gprestore使用一个连接去恢复表数据和元数据。如果备份集非常大，可以使用--jobs选项来提高并发连接数从而提升恢复性能。例如：

```

$ gprestore --backup-dir /home/gadmin/backups/ --timestamp
20171103153156 --create-db --jobs 8

```

测试备份集的并行连接数，以确定快速数据恢复的理想数量。

Note: 如果备份使用gpbackup的--single-data-file选项将表备份组合到每个节点的单个文件中，则无法使用gprestore执行并行还原操作。

报告文件

当执行备份和恢复操作时，gpbackup和gprestore会生成一个报告文件。当配置了邮件通知，发送的邮件内容会包含报告文件。关于邮件通知的信息，参考[配置邮件通知](#)。

报告文件在Greenplum数据库master的备份目录中。报告文件名包含操作的时间戳。这些是gpbackup和gprestore报告文件名的格式。

```

gpbackup_<backup_timestamp>_report
gprestore_<backup_timestamp>_<restore_timestamp>_report

```

对于这些报告文件名的样例，20180213114446是备份的时间戳，20180213115426是恢复操作的时间戳。

```

gpbackup_20180213114446_report
gprestore_20180213114446_20180213115426_report

```

这个在Greenplum数据库master主机上的备份目录包含了gpbackup和gprestore的报告文件。

```
$ ls -l /gpmaster/seg-1/backups/20180213/20180213114446
total 36
-r--r--r--. 1 gpadmin gpadmin 295 Feb 13 11:44
gpbackup_20180213114446_config.yaml
-r--r--r--. 1 gpadmin gpadmin 1855 Feb 13 11:44
gpbackup_20180213114446_metadata.sql
-r--r--r--. 1 gpadmin gpadmin 1402 Feb 13 11:44
gpbackup_20180213114446_report
-r--r--r--. 1 gpadmin gpadmin 2199 Feb 13 11:44
gpbackup_20180213114446_toc.yaml
-r--r--r--. 1 gpadmin gpadmin 404 Feb 13 11:54
gprestore_20180213114446_20180213115426_report
```

报告文件的内容类似。这是gprestore报告文件内容的示例。

```
Greenplum Database Restore Report

Timestamp Key: 20180213114446
GPDB Version: 5.4.1+dev.8.g9f83645 build
commit:9f836456b00f855959d52749d5790ed1c6efc042
gprestore Version: 1.0.0-alpha.3+dev.73.g0406681

Database Name: test
Command Line: gprestore --timestamp 20180213114446 --with-globals --
createdb

Start Time: 2018-02-13 11:54:26
End Time: 2018-02-13 11:54:31
Duration: 0:00:05

Restore Status: Success
```

历史文件

执行备份操作时，gpbackup会将备份信息附加到Greenplum数据库master数据目录中的gpbackup历史文件gpbackup_history.yaml中。该文件包含备份时间戳，有关备份选项的信息以及增量备份的备份集信息。gpbackup不备份此文件。

使用--incremental选项运行gpbackup时，gpbackup使用文件中的信息查找增量备份的匹配备份，并且不指定--from-timestamp选项以指示要用作最新备份的备份在增量备份集中。有关增量备份的信息，参考[使用gpbackup和gprestore创建增量备份](#)。

返回值

在gpbackup或gprestore完成时会返回下面的一个返回值。

- **0** – 备份或恢复完成没有问题
- **1** – 备份或恢复完成，但有非致命错误。查看日志文件获取更多信息。
- **2** – 备份或恢复因为致命错误失败。查看日志文件获取更多信息。

Parent topic: [使用gpbackup和gprestore并行备份](#)

过滤备份或恢复的内容

gpbackup备份指定数据库中的所有schema和表，除非您使用schema级别或表级别过滤器选项排除或包含单个schema或表对象。

schema级别选项是--include-schema或--exclude-schema命令行选项到gpbackup。例如，如果“demo”数据库仅包含两个schema “wikipedia”和“twitter”，则以下两个命令仅备份“wikipedia” schema：

```
$ gpbackup --dbname demo --include-schema wikipedia
$ gpbackup --dbname demo --exclude-schema twitter
```

你可以在gpbackup中包含多个--include-schema选项，或多个--exclude-schema选项。例如：

```
$ gpbackup --dbname demo --include-schema wikipedia --include-schema
twitter
```

要筛选包含在备份集中的表或从备份集中排除表，请使用--include-table选项或--exclude-table选项指定各个表。该表必须是限定schema的，<schema-name>.<table-name>。可以多次指定各个表过滤选项。但是，--include-table和--exclude-table不能同时用于同一命令。

您可以在文本文件中创建指定的表名称列表。列出文件中的表时，文本文件中的每一行都必须使用格式<schema-name>.<table-name>定义单个表。该文件不得包含尾随行。例如：

```
wikipedia.articles
twitter.message
```

如果表或schema名称使用小写字母，数字或下划线字符以外的任何字符，则必须在双引号中包含该名称。例如：

```
beer."IPA"
"Wine".riesling
"Wine"."sauvignon blanc"
water.tonic
```

创建文件后，您可以使用gpbackup选项--include-table-file或--exclude-table-file来包含或排除表。例如：

```
$ gpbackup --dbname demo --include-table-file /home/gpadmin/table-
list.txt
```

您可以将--include-schema与--exclude-table或--exclude-table-file结合使用以进行备份。此示例使用--include-schema和--exclude-table来备份单个表之外的schema。

```
$ gpbackup --dbname demo --include-schema mydata --exclude-table
mydata.addresses
```

您不能将--include-schema与--include-table或--include-table-file结合使用，并且不能将--exclude-schema与任何表过滤选项（如--exclude-table或--include-table）结合使用。

使用--include-table或--include-table-file时，不会自动备份或还原依赖对象，您必须显式指定所需的依赖对象。例如，如果备份或还原视图，则还必须指定视图使用的表。如果备份或还原使用序列的表，则还必须指定序列。

根据叶子分区过滤

默认情况下，gpbackup为节点上的每个表创建一个文件。您可以指定--leaf-partition-data选项，以便为分区表的每个叶子分区创建一个数据文件，而不是单个文件。您还可以通过在包含的文本文件中列出叶子分区名称来过滤备份特定叶子分区。例如，考虑使用以下语句创建的表：

```
demo=# CREATE TABLE sales (id int, date date, amt decimal(10,2))
DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
( PARTITION Jan17 START (date '2017-01-01') INCLUSIVE ,
```

```

PARTITION Feb17 START (date '2017-02-01') INCLUSIVE ,
PARTITION Mar17 START (date '2017-03-01') INCLUSIVE ,
PARTITION Apr17 START (date '2017-04-01') INCLUSIVE ,
PARTITION May17 START (date '2017-05-01') INCLUSIVE ,
PARTITION Jun17 START (date '2017-06-01') INCLUSIVE ,
PARTITION Jul17 START (date '2017-07-01') INCLUSIVE ,
PARTITION Aug17 START (date '2017-08-01') INCLUSIVE ,
PARTITION Sep17 START (date '2017-09-01') INCLUSIVE ,
PARTITION Oct17 START (date '2017-10-01') INCLUSIVE ,
PARTITION Nov17 START (date '2017-11-01') INCLUSIVE ,
PARTITION Dec17 START (date '2017-12-01') INCLUSIVE
END (date '2018-01-01') EXCLUSIVE );
NOTICE: CREATE TABLE will create partition "sales_1_prt_jan17" for table
"sales"
NOTICE: CREATE TABLE will create partition "sales_1_prt_feb17" for table
"sales"
NOTICE: CREATE TABLE will create partition "sales_1_prt_mar17" for table
"sales"
NOTICE: CREATE TABLE will create partition "sales_1_prt_apr17" for table
"sales"
NOTICE: CREATE TABLE will create partition "sales_1_prt_may17" for table
"sales"
NOTICE: CREATE TABLE will create partition "sales_1_prt_jun17" for table
"sales"
NOTICE: CREATE TABLE will create partition "sales_1_prt_jul17" for table
"sales"
NOTICE: CREATE TABLE will create partition "sales_1_prt_aug17" for table
"sales"
NOTICE: CREATE TABLE will create partition "sales_1_prt_sep17" for table
"sales"
NOTICE: CREATE TABLE will create partition "sales_1_prt_oct17" for table
"sales"
NOTICE: CREATE TABLE will create partition "sales_1_prt_nov17" for table
"sales"
NOTICE: CREATE TABLE will create partition "sales_1_prt_dec17" for table
"sales"
CREATE TABLE

```

要仅备份一年中最后一个季度的数据，首先要创建一个文本文件，列出那些叶子分区名称而不是完整的表名称：

```

public.sales_1_prt_oct17
public.sales_1_prt_nov17
public.sales_1_prt_dec17

```

然后使用`--include-table-file`选项指定文件，以便为每个叶子分区生成一个数据文件：

```

$ gpbackup --dbname demo --include-table-file last-quarter.txt --leaf-
partition-data

```

指定`--leaf-partition-data`时，`gpbackup`在备份分区表时为每个叶子分区生成一个数据文件。例如，此命令为每个叶子分区生成一个数据文件：

```

$ gpbackup --dbname demo --include-table public.sales --leaf-partition-
data

```

备份叶子分区时，将备份叶子分区数据以及整个分区表的元数据。

Note: 您不能将`--exclude-table-file`选项与`--leaf-partition-data`一起使用。虽然您可以在使用`--exclude-table-file`指定的文件中指定叶子分区名称，但`gpbackup`会忽略分区名称。

使用gprestore过滤

使用`gpbackup`创建备份集后，可以使用`gprestore`的`--include-schema`和`--include-table-file`选项过滤要从备份集还原的schema和表。这些选项的工作方式与其`gpbackup`对应方式相同，但具有以下限制：

- 您尝试还原的表不能存在于数据库中。
- 如果尝试还原备份集中不存在的schema或表，则gprestore不会执行。
- 如果使用--include-schema选项，则gprestore无法还原对多个schema具有依赖性的对象。
- 如果使用--include-table-file选项，则gprestore不会创建角色或设置表的所有者。该工具恢复表索引和规则。触发器也被恢复，但Greenplum数据库不支持。
- 使用--include-table-file指定的文件不能包含叶子分区名称，因为在使用gpbackup指定此选项时可以使用该名称。如果在备份集中指定了叶子分区，请指定分区表以还原叶子分区数据。
还原包含来自分区表的某些叶子分区的数据的备份集时，将恢复分区表以及叶子分区的数据。例如，使用gpbackup选项--include-table-file创建备份，文本文件列出分区表的一些叶子分区。还原备份会创建分区表，并仅为文件中列出的叶分区还原数据。

Parent topic: [使用gpbackup和gprestore并行备份](#)

配置邮件通知

gpbackup和gprestore可以在备份或还原完成后发送邮件通知。

要让gpbackup或gprestore发送状态电子邮件通知，您必须将名为gp_email_contacts.yaml的文件放在运行gpbackup或gprestore的用户的主目录中，与工具（\$GPHOME/bin）位于同一目录中。如果工具无法在任一位置找到gp_email_contacts.yaml文件，则会发出消息。如果两个位置都包含.yaml文件，则该工具将使用用户\$HOME中的文件。

电子邮件主题行包括工具名称，时间戳，状态和Greenplum数据库master的名称。这是gpbackup电子邮件的示例主题行。

```
gpbackup 20180202133601 on gp-master completed
```

该电子邮件包含有关操作的摘要信息，包括选项，持续时间以及备份或还原的对象数。有关通知电子邮件内容的信息，参考[报告文件](#)。

Note: UNIX邮件实用程序必须在Greenplum数据库主机上运行，并且必须配置为允许Greenplum超级用户（gpadmin）发送电子邮件。还要确保通过gpadmin用户的\$PATH可以找到邮件程序可执行文件。

Parent topic: [使用gpbackup和gprestore并行备份](#)

gpbackup和gprestore邮件文件格式

gpbackup和gprestore电子邮件通知的YAML文件gp_email_contacts.yaml使用缩进（空格）来确定文档层次结构以及这些部分之间的关系。使用空白区域非常重要。不应仅将白色空间用于格式化目的，并且根本不应使用制表符。

Note: 如果未正确指定status参数，则工具不会发出警告。例如，如果success参数拼写错误并设置为true，则不会发出警告，并且在成功操作后不会向电子邮件地址发送电子邮件。要确保正确配置电子邮件通知，请运行配置了电子邮件通知的测试。

这是gpbackup电子邮件通知的gp_email_contacts.yaml YAML文件的格式：

```
contacts:
  gpbackup:
    - address: user@domain
      status:
        success: [true | false]
        success_with_errors: [true | false]
```

```

failure: [true | false]
gprestore:
- address: user@domain
status:
  success: [true | false]
  success_with_errors: [true | false]
  failure: [true | false]

```

电子邮件YAML文件部分

contacts

必须的。包含gpbackup和gprestore部分的部分。 YAML文件可以包含gpbackup部分， gprestore部分或每个部分中的一个。

gpbackup

可选的。开始gpbackup电子邮件部分。

address

必须的。必须至少指定一个电子邮件地址。可以指定多个电子邮件地址参数。每个地址都需要一个status部分。

*user@ domain*是一个有效的电子邮件地址。

status

必须的。指定工具何时向指定的电子邮件地址发送电子邮件。默认设置是不发送电子邮件通知。

您可以根据备份或还原操作的完成状态指定发送电子邮件通知。必须至少指定其中一个参数，每个参数最多只能出现一次。

success

可选的。如果操作完成且没有错误，请指定是否发送电子邮件。如果值为true，则在操作完成且没有错误的情况下发送电子邮件。如果值为false（默认值），则不会发送电子邮件。

success_with_errors

可选的。如果操作完成但有错误，请指定是否发送电子邮件。如果值为true，则在操作完成且出错时将发送电子邮件。如果值为false（默认值），则不会发送电子邮件。

failure

可选的。指定操作失败时是否发送电子邮件。如果值为true，则在操作失败时发送电子邮件。如果值为false（默认值），则不会发送电子邮件。

gprestore

可选的。开始gprestore电子邮件部分。此部分包含用于在gprestore操作后发送电子邮件通知的address和status参数。语法与gpbackup部分相同。

示例

此示例YAML文件指定根据操作的成功或失败向电子邮件地址发送电子邮件。对于备份操作，根据备份操作的成功或失败，将电子邮件发送到不同的地址。对于还原操作，仅当操作成功或带错完成时才会向gpadmin@example.com发送电子邮件。

```

contacts:
  gpbackup:
    - address: gpadmin@example.com
      status:
        success:true
    - address: my_dba@example.com
      status:
        success_with_errors: true
        failure: true
  gprestore:
    - address: gpadmin@example.com
      status:
        success: true
        success_with_errors: true

```

理解备份文件

Warning: 所有gpbackup元数据文件都是使用只读权限创建的。切勿删除或修改gpbackup备份集的元数据文件。这样做会使备份文件无法正常运行。

gpbackup的完整备份集包括多个元数据文件，支持文件和CSV数据文件，每个文件都指定了创建备份的时间戳。

默认情况下，元数据和支持文件存储在Greenplum数据库master主机上\$MASTER_DATA_DIRECTORY/backups/YYYYMMDD/YYYYMMDDHHMMSS/目录中。如果指定自定义备份目录，则会将此相同文件路径创建为备份目录的子目录。下表描述了元数据和支持文件的名称和内容。

Table 2. gpbackup元文件(master)

文件名	描述
gpbackup_<YYYYMMDDHHMMSS>_metadata.sql	<p>包含全局和数据库特定的元数据：</p> <ul style="list-style-type: none"> • 用于Greenplum数据库集群全局对象的DDL，不属于集群中的特定数据库。 • 用于备份数据库中的对象的DDL（使用--dbname指定），必须在还原实际数据之前创建，以及在还原数据后必须创建的对象的DDL。 <p>全局对象包括：</p> <ul style="list-style-type: none"> • 表空间 • 数据库 • 数据库范围的配置参数设置(GUCs) • Resource group定义 • Resource queue定义 • Roles • GRANT分配给数据库的角色 <p>Note:默认情况下不会还原全局元数据。必须在gprestore命令中包含--with-globals选项才能还原全局元数据。</p> <p>在还原实际数据之前必须创建的数据库特定的对象包括：</p> <ul style="list-style-type: none"> • Session级别的配置参数设置(GUCs) • Schemas • 过程语言扩展 • 类型 • 序列 • 函数 • 表 • 协议 • 操作符和操作符类 • 转换 • 聚合 • 类型转换 • 视图

		约束 在还原实际数据之后必须创建的数据库特定的对象包括： <ul style="list-style-type: none">索引规则触发器。(虽然Greenplum数据库不支持触发器，但会备份和恢复任何存在的触发器定义。)
gpbackup_<YYYYMMDDHHMMSS>_toc.yaml		包含用于在_predatadata.sql和_postdata.sql文件中查找对象DDL的元数据。此文件还包含用于在每个节点上创建的CSV数据文件中查找相应表数据的表名和OID。参考 节点数据文件 。
gpbackup_<YYYYMMDDHHMMSS>_report		包含有关用于填充备份完成后发送的电子邮件通知(如果已配置)的备份操作的信息。此文件包含以下信息： <ul style="list-style-type: none">提供的命令行选项备份的数据库数据库版本备份类型 参考 配置邮件通知 。
gpbackup_<YYYYMMDDHHMMSS>_config.yaml		包含有关特定备份任务执行的元数据，包括： <ul style="list-style-type: none">gpbackup版本数据库名称Greenplum数据库版本附加选项配置，如--no-compression, --compression-level, --metadata-only, --data-only和--with-stats。
gpbackup_history.yaml		包含有关使用gpbackup创建备份时使用的选项的信息，以及有关增量备份的信息。 存储在Greenplum数据库master数据目录中的Greenplum数据库主机上。 这个文件不是通过gpbackup备份的。 关于增量备份的信息，参考 使用gpbackup和gprestore创建增量备份 。

节点数据文件

默认情况下，每个节点为在节点上备份的每个表创建一个压缩CSV文件。您可以选择指定--single-data-file选项，以在每个节点上创建单个数据文件。文件存储在<seg_dir>/backups/YYYYMMDD/YYYYMMDDHHMMSS/。

如果指定自定义备份目录，则节点数据文件将复制到与备份目录的子目录相同的文件路径中。如果包含--leaf-partition-data选项，则gpbackup为分区表的每个叶子分区创建一个数据文件，而不是为表创建一个文件。

每个数据文件使用文件吗格式

gpbackup_<content_id>_<YYYYMMDDHHMMSS>_<oid>.gz, 其中:

- <content_id>是节点的content ID。
- <YYYYMMDDHHMMSS>是gpbackup操作的时间戳。
- <oid>是表的object ID。元数据文件gpbackup_<YYYYMMDDHHMMSS>_toc.yaml引用此<oid>以查找schema中特定的表的数据。

您可以选择使用--compression-level选项指定gzip压缩级别（1-9），或使用--no-compression完全禁用压缩。如果未指定压缩级别，则gpbackup默认使用压缩级别1。

Parent topic: [使用gpbackup和gprestore并行备份](#)

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库
发布版本号启动和停
止Greenplum数据库

□ 访问数据库

□ 配置Greenplum数据库
系统

启用压缩

□ 启用高可用和数据持久
化特征

□ 备份和恢复数据库

□ 扩容Greenplum系统

系统扩容概述

规划Greenplum系统
扩容

准备并增加节点

初始化新节点

重分布表

移除扩容Schema

监控Greenplum系统

日常系统维护任务

用户可以最小化停机时间，通过增加节点实例和节点主机来扩容Greenplum数据库。

随着收集额外数据并且现有数据的定期增长，数据仓库通常会随着时间的推移而不断增长。有时，有必要增加数据库能力来联合不同的数据仓库到一个数据库中。数据仓库也可能需要额外的计算能力（CPU）来适应新增加的分析项目。在系统被初始定义时就留出增长的空间是很好的，但是即便用户预期到了高增长率，提前太多在资源上投资通常也不明智。因此，用户应该寄望于定期地执行一次数据库扩容项目。

当用户扩容数据库时，会期待如下特性：

- 可伸缩的容量和性能。当用户向一个Greenplum数据库增加资源时，得到的容量和性能就好像该系统一开始就用增加后的资源实现一样。
- 扩容期间不中断服务。常规负载（计划中的或者临时安排的）不会被中断。
- 事务一致性。
- 容错。在扩容期间，标准的容错机制（例如Segment镜像）保持活动、一致并且有效。
- 复制和灾难恢复。在扩容期间，任何现有的复制机制都继续发挥作用。在失败或者灾难事件中需要的恢复机制也保持有效。
- 处理透明。扩容处理利用了标准的Greenplum数据库机制，因此管理员能够诊断并且排查任何问题。
- 可配置的处理。扩容可能会是一个长时间运行的处理，但它可以变成按计划执行的一系列操作。扩容Schema的表允许管理员指定表被重新分布的优先级，而且扩容活动可以被暂停并且继续。

扩容项目的规划和实体层面比扩容数据库本身更重要。需要一个多学科团队来规划和执行项目。对于内部部署安装，必须为新服务器获取和准备空间。必须指定，获取，安装，连接，配置和测试服务器。对于云部署，也应该做类似的计划。[规划新的硬件平台](#)描述了部署新硬件的常规考虑。

在准备好新的硬件平台并且设置好它们的网络之后，配置它们的操作系统并且使用Greenplum的工具运行性能测试。Greenplum数据库软件发布包括一些工具，这些工具有助于在开始扩容的软件阶段之前对新的服务器进行测试和拷机。为Greenplum数据库准备新节点的步骤请见[准备并增加节点](#)。

一旦新服务器被安装并且测试，Greenplum数据库扩容处理的软件阶段

就开始了。软件阶段被设计为尽量少被打断、事务一致、可靠并且灵活。

- 扩容过程的软件阶段的第一步是准备Greenplum数据库系统：添加新节点的主机并初始化新的节点实例。此阶段可以安排在低活动期间进行，以避免中断正在进行的业务运营。在初始化过程中，执行以下任务：

- Greenplum数据库软件已被安装。
- 在新的Segment主机的新节点实例上创建了数据库和数据库对象。
- *gpexpand* schema在postgres数据库里被创建。可以用schema里的表和视图来监控和管理扩容。

在系统被更新后，新节点主机上的新节点实例已经可以使用了。

- 新节点立即生效并参与到新查询和数据加载里。但是现有数据会发生倾斜。它们集中在原节点上并且需要根据新节点总量做重分布。
- 因为有些表的数据已经倾斜了，所以部分查询性能会下降，因为可能需要更多的Motion操作了。
- 软件阶段的最后一步就是重新分布表数据。使用*gpexpand* schema中的扩容控制表作为指导，重新分配表。对每一个表：
 - *gpexpand* 工具基于分布策略在所有新老机器上重新分布表数据。
 - 扩容控制表中的表状态会被更新。
 - 在数据重分布之后，查询优化器会基于不倾斜的数据创建更高效的查询计划。

当所有表都完成了重分布，扩容也就完成了。

Important: *gprestore*工具不能恢复在扩容之前使用*gpbackup*工具备份的数据。所以在扩容完成之后立即备份你的数据。

重新分布数据是一个长时间运行的处理，它会创建大量的网络和磁盘活动。可能需要数天时间来重新分布某些非常大型的数据库。为了最小化这些活动对业务操作的影响，系统管理员可以随意或者根据一个预定好的计划暂停并且继续扩容活动。数据集也可以被定义优先级，这样关键的应用可以从扩容中首先获益。

在一种典型的操作中，在完整的扩容处理过程中，用户需要用不同的选项运行*gpexpand*工具四次。

1. 创建扩容输入文件：

```
gpexpand -f hosts_file
```

2. 初始化Segment并且创建扩容schema:

```
gpexpand -i input_file
```

gpexpand会创建一个数据目录、从现有的数据库复制表到新的Segment上并且为扩容方案中的每个表捕捉元数据用于状态跟踪。在这个处理完成后，扩容操作会被提交并且不可撤回。

3. 重新分布表数据:

```
gpexpand -d duration
```

在初始化时，gpexpand增加并初始化新节点实例。必须运行gpexpand在新增节点实例上重分布数据来完成系统扩容。取决于系统的大小和规模，重新分布可能在一个单一会话中经过数个利用率较低的小时才会完成，或者用户可以把该处理划分成一个长时段上的批处理。每个表或分区在扩容期间是无法进行读写操作的。由于每一个表都会被重新分布在新的Segment上，数据库性能应该会逐步提升直到超过扩容前的性能水平。

在需要多个重新分布会话的大型系统中，可能需要多次运行gpexpand来完成扩容。gpexpand可以从显式的表重新分布排名获益，参考[规划表的重新分布](#)。

用户可以在初始化期间访问Greenplum数据库，但是在严重依赖表的哈希分布的系统上，它们可能会出现性能下降。尽管用户可能会遇到较慢的响应时间，但ETL作业，用户查询和报告等正常操作仍可继续。

4. 移除扩容schema:

```
gpexpand -c
```

有关gpexpand工具和其他用于系统扩容的工具的信息可见[Greenplum数据库工具指南](#)。

Parent topic: 扩容Greenplum系统

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库

发布版本号

启动和停

止Greenplum数据库

□ 访问数据库

□ 配置Greenplum数据库
系统

启用压缩

□ 启用高可用和数据持久
化特征

□ 备份和恢复数据库

□ 扩容Greenplum系统

系统扩容概述

规划Greenplum系统
扩容

准备并增加节点

初始化新节点

重分布表

移除扩容Schema

监控Greenplum系统

日常系统维护任务

细心的规划将帮助确保一个成功的Greenplum扩容项目。

这一节中的主题将帮助确保用户已经准备好执行一次系统扩容。

- [系统扩容检查列表](#)是一份用户可以用来准备和执行系统扩容处理的检查列表。
- [规划新的硬件平台](#)包括为获得和设置新硬件做规划的内容。
- [规划新Segment初始化](#)提供了有关规划使用gpexpand初始化新的Segment主机的信息。
- [规划表的重新分布](#)提供了有关规划在新Segment主机初始化完以后重新分布数据的信息。

Parent topic: [扩容Greenplum系统](#)

系统扩容检查列表

这个检查列表摘要了一次Greenplum数据库系统扩容包括的任务。

Table 1. Greenplum数据库系统扩容检查列表

扩容前的在线任务	
* 系统已启动并且可用	
<input type="checkbox"/>	制定并执行订购，构建和联网新硬件平台或配置云资源的计划。
<input type="checkbox"/>	设计一个数据库扩容计划。映射每个主机上的Segment数量、安排用于性能测试和创建扩容方案所需的停机时段并且安排用于表重新分布的时段。
<input type="checkbox"/>	执行一个完整的方案转储。
<input type="checkbox"/>	在新主机上安装Greenplum数据库的二进制文件。
<input type="checkbox"/>	复制SSH密钥到新主机 (gpssh-exkeys)。
<input type="checkbox"/>	验证新旧硬件或者云资源的操作系统环境 (gpcheckosk)。
<input type="checkbox"/>	验证新硬件或者云资源的磁盘I/O和内存带宽 (gpcheckperf)。

<input type="checkbox"/>	验证Master的数据目录中在pg_log或者gpperfmon/data目录下面没有极大的文件。
扩容前的离线任务	
* 在此处理期间系统对所有用户活动不可用。	
<input type="checkbox"/>	验证没有catalog问题 (gpcheckcat)。
<input type="checkbox"/>	验证组合的现有和新硬件或云资源的操作系统环境(gpcheck)。
<input type="checkbox"/>	验证组合的现有和新硬件或云资源的磁盘I/O和内存带宽(gpcheckperf)。
在线节点实例初始化	
* 系统已启动并且可用	
<input type="checkbox"/>	准备扩容输入文件(gpexpand)。
<input type="checkbox"/>	将新节点初始化为数组并创建扩容Schema (gpexpand -i <i>input_file</i>)。
在线扩容和表重新分布	
* 系统已启动并且可用	
<input type="checkbox"/>	在用户开始表重新分布之前，停止任何自动的快照处理或者其他消耗磁盘空间的进程。
<input type="checkbox"/>	在扩容后的系统中重新分布表 (gpexpand)。
<input type="checkbox"/>	移除扩容方案(gpexpand -c)。
<input type="checkbox"/>	运行analyze来更新分布统计信息。 在扩容期间使用gpexpand -a，在扩容之后使用analyze。
备份数据库	
* 系统已启动并且可用	
<input type="checkbox"/>	使用gpbackup工具备份数据库。在开始系统扩容之前创建的备份无法还原到新扩容的系统，因为gprestore实用程序只能将备份还原到具有相同数量节点的Greenplum数据库系统。

规划新的硬件平台

一个深思熟虑的、周密的部署兼容硬件的方法会大大地最小化扩容处理的风险。

新节点主机的硬件资源和配置应该与现有主机一致。

规划和设置新硬件平台的步骤在每一次部署中都有不同。下面是一些相关的考虑：

- 为新硬件准备物理空间，考虑冷却、电力供应和其他物理因素。
- 确定连接新旧硬件所需的物理网络和布线。
- 为扩容后的系统映射现有的IP地址空间和开发中的网络规划。
- 从现有的硬件捕捉系统配置（用户、配置文件、NIC等等），这将被用作订购新硬件时的清单。
- 为在特定站点和环境中用期望的配置部署硬件创建一个自定义的建设计划。

在选择并且增加新硬件到网络环境中后，确保执行[验证OS设置](#)中描述的任务。

规划新Segment初始化

当系统启动并可用时，可以执行扩容Greenplum数据库。运行`gpexpand`来初始化新节点到阵列中并创建扩容schema。

所需要的时间取决于Greenplum系统中的方案对象的数量以及其他与硬件性能相关的因素。在大部分环境中，新Segment的初始化不超过30分钟。

下列工具不能在`gpexpand`在做节点初始化期间执行。

- `gpbackup`
- `gpcheck`
- `gpcheckcat`
- `gpconfig`
- `gppkg`
- `gprestore`

Important: 在用户开始初始化新Segment之后，用户就不再能用扩容系统之前创建的备份文件来恢复系统。当初始化成功完成后，扩容会被提交且不能被回滚。

规划Mirror节点

如果现有的阵列有Mirror节点，新的节点也必须有Mirror配置。如果现有的节点没有配置Mirror，则不能用gpexpand工具给新主机增加Mirror。有关节点Mirror的更多信息，请参考[关于Segment镜像](#)。

对于带有Mirror节点的Greenplum数据库阵列，确保增加了足够的新主机来容纳新的Mirror节点。所需的新主机数量取决于Mirror策略：

- 散布镜像 — 向阵列中增加比每个主机上的节点数量至少多一台的主机。要确保平均散布，阵列中独立主机的数量必须大于每台主机上的节点实例数量。散布镜像会把每台主机的镜像散布到集群中剩余的主机上并且要求集群中的主机数量比每个主机上的Primary节点数量更多。
- 组镜像 — 增加至少两台新主机，这样第一台主机的镜像可以被放在第二台主机上，并且第二台主机的镜像可以被放在第一台上。如果在系统初始化阶段启用了节点镜像，这是默认的Mirroring策略。
- 块镜像 — 添加一个或多个主机系统块。例如，添加一个包含四个或八个主机的块。块镜像是一种自定义镜像配置。关于块镜像的更多信息请参考*Greenplum*数据库最佳实践指导中的[Segment镜像](#)。

对每个主机增加新节点

默认情况下，新主机上初始化后会有和现有主机上数量相同的主节点。可以增加每台主机上的节点或者向现有主机上增加新的节点。

例如，如果现有主机当前在每台主机上有两个节点，可以用gpexpand在现有主机上初始化两个额外的节点来得到总共四个节点，这样将在新主机上有四个新的节点。

创建扩容输入文件的交互式处理会提示这个选项，还可以在该输入配置文件中手工指定新节点的目录。更多信息请参考[为系统扩容创建一个输入文件](#)。

关于扩容Schema

在初始化阶段，`gpexpand`工具在`postgres`数据库中创建扩容schema `gpexpand`。

扩容Schema存储了系统中每个表的元数据，因此在扩容处理的全过程中能跟踪其状态。扩容Schema由两个表和一个跟踪扩容操作进度的视图组成：

- `gpexpand.status`
- `gpexpand.status_detail`
- `gpexpand.expansion_progress`

通过修改`gpexpand.status_detail`可以控制扩容处理的方方面面。例如，从这个表中移除一个记录会阻止系统在新节点上扩容该表。通过更新一个记录的rank值，可以控制表在重新分布过程中被处理的顺序。更多信息请参考[表重分布排名](#)。

规划表的重新分布

表重新分布会在系统在线时被执行。对于很多Greenplum系统，表重新分布会在一个安排在低利用率时期执行的单一`gpexpand`会话中完成。更大的系统可能会要求多个会话并且设置表重新分布的顺序来最小化对性能影响。如果可能的话，尽量在一个会话中完成表重新分布。

Important: 要执行表重新分布，节点主机必须具有足够多的磁盘空间来临时地保存最大表的一份拷贝。在重新分布期间，所有的表对于读写操作都不可用。

表重新分布的性能影响取决于一个表的尺寸、存储类型以及分区设计。对于任何给定的表，用`gpexpand`重新分布需要消耗和一次`CREATE TABLE AS SELECT`操作相同的时间。在重新分布一个T级别的表时，扩容工具会使用许多可用的系统资源，这可能会影响其他数据库负载的查询性能。

在大规模Greenplum系统中管理重新分布

在规划重新分布阶段时，要考虑重新分布期间在每个表上取得的`ACCESS EXCLUSIVE`锁的影响。一个表上的用户活动可能会延迟它的重新分布，而且表在重新分布期间对用户活动不可用。

可以通过调整`ranking`来控制表重分布的顺序。参考[表重分布排名](#)。

操作重分布顺序有助于调整有限的磁盘空间，并尽快恢复高优先级查询的最佳查询性能。

表重分布方法

Greenplum数据库扩容的时候有两种方法重分布数据。

- `rebuild` - 建立一个新表，把所有数据从旧表复制到新表，然后替换旧表。这是默认行为。`rebuild`方法和使用`CREATE TABLE AS SELECT`命令创建一个新表类似。在数据重分布期间，会在表上加`ACCESS EXCLUSIVE`锁。
- `move` - 扫描所有数据并做`UPDATE`操作来移动需要移动的行到不同节点实例上。在数据重分布过程中，会在表上加`ACCESS EXCLUSIVE`锁。一般来说，这个方法需要更少的磁盘空间，但是它在表里留下了很多被淘汰的行，可能在数据重分布后需要`VACUUM`操作。另外，此方法一次更新一行索引，这会使它比使用`CREATE INDEX`命令重建索引更慢。

磁盘空间充裕的系统

如果系统由充裕的空闲磁盘空间（要求用来存储最大表的一份拷贝），可以通过首先重新分布查询使用最多的重点表来尽快恢复最优的查询性能。为这些表分配高的排名，并且在系统使用量低的时候安排重新分布操作。一次运行一个重新分布处理，直到大的或者关键的表被重新分布完。

磁盘空间有限的系统

如果现有的主机磁盘空间有限，可以先重新分布较小的表（例如维度表）来腾出空间存储最大表的一份拷贝。由于每个表会被重新分布到被扩容后的阵列上，在原始节点上的可用磁盘空间会增加。当在所有节点上存在足够的空闲空间以存储最大表的一份拷贝后，就可以重新分布大的或者关键的表了。大表的重新分布要求排他锁，请把这种过程安排在非峰值时段。

还要考虑下面的事情：

- 在非峰值时段运行多个并行的重新分布处理以最大化可用的系统资源。
- 在运行多个处理时，处理的数量不能高于Greenplum系统的连接限制。有关限制并发连接的信息请见[限制并发连接](#)。

重新分布追加优化和压缩过的表

gpexpand以与堆表不同的速率重新分布未压缩和压缩的追加优化表。 压缩和解压数据所需的CPU能力会导致增加对系统性能影响。 对于具有类似数据的类似尺寸的表，可以发现如下的总体性能区别：

- 扩展未压缩的追加优化表比堆表快10%。
- 定义为使用数据压缩的追加优化表以比未压缩的追加优化表的扩展速率更慢，可能慢80%。
- 使用了ZFS/LZJB等数据压缩的系统需要更长时间重新分布。

Important: 如果系统节点使用数据压缩，在新结点上使用相同的压缩来避免磁盘空间短缺。

重新分布分区表

因为该扩展工具可以处理一个大型表上的每个分区，一个有效的分区设计能降低表重新分布的性能影响。 只有一个分区表的子表会不被设置为随机分布策略。 一次只对一个子表应用重新分布所需的读/写锁。

重新分布建有索引的表

因为gpexpand工具必须在重新分布之后重新索引每个被索引的表，一次高级的索引会带来很大的性能影响。 在有很多索引要建立的系统中，表的重新分布要慢很多。

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库
发布版本号启动和停
止Greenplum数据库

□ 访问数据库

□ 配置Greenplum数据库
系统

启用压缩

□ 启用高可用和数据持久
化特征

□ 备份和恢复数据库

□ 扩容Greenplum系统

系统扩容概述

规划Greenplum系统
扩容

准备并增加节点

初始化新节点

重分布表

移除扩容Schema

监控Greenplum系统

日常系统维护任务

验证用户的新节点已经准备好整合到现有的Greenplum系统中。

要为扩展准备新的系统节点，需要安装Greenplum数据库软件的二进制文件、交换必要的SSH密钥并且运行性能测试。

先在新节点上运行性能测试然后再在所有结点上测试。在所有节点上运行测试时应该让系统离线，这样用户活动就不会使结果失真。

通常来说，当管理员修改了节点网络或者系统中出现其他特殊情况时，用户应该运行性能测试。例如，如果用户将在两个网络集群上运行扩展后的系统，应在每一个集群上都运行测试。

Parent topic: [扩容Greenplum系统](#)

增加新结点到受信主机环境

新主机必须与现有主机交换SSH密钥，以使Greenplum管理实用程序能够在没有密码提示的情况下连接到所有节点。使用[gpssh-exkeys](#)工具执行两次密钥交换过程。

为了管理便利，第一次作为root执行该过程，然后为管理工具作为用户gpadmin再执行一次。按顺序执行下列任务：

1. [用root交换SSH密钥](#)
2. [创建gpadmin用户](#)
3. [使用gpadmin用户交换SSH密钥](#)

Note: Greenplum数据库的Segment主机命名习惯是sdwN，其中sdw是一个前缀而N是一个整数（sdw1、sdw2等）。对于具有多个接口的主机，命名习惯是对该主机名追加一个破折号（-）以及数字。例如，sdw1-1和sdw1-2是主机sdw1的两个接口名。

用root交换SSH密钥

1. 使用用户的阵列中的现有主机名创建一个主机文件以一个含有新扩展主机名的单独的主机文件。对于现有的主机，用户可以使用在系统中设置SSH密钥的同一个主机文件。在这些文件中，列出所

Master	Master	Segment
--------	--------	---------

有的主机（**主**、**备份** 和 **辅助** 主机），每一行一个主机名但是不要有额外的行或者空白。如果用户使用的是一个多NIC配置，请使用为一个给定的主机用配置好的主机名交换SSH密钥。在这个例子中，mdw被配置为有一个NIC，并且sdw1、sdw2和sdw3被配置为有4个NIC：

```
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
```

2. 用**root**登录Master主机，并且从用户的Greenplum安装中source `greenplum_path.sh`文件。

```
$ su -
# source /usr/local/greenplum-db/greenplum_path.sh
```

3. 运行`gpssh-exkeys`工具引用主机列表文件。例如：

```
# gpssh-exkeys -e /home/gpadmin/existing_hosts_file -x
/home/gpadmin/new_hosts_file
```

4. `gpssh-exkeys`会检查远程主机并且在所有的主机之间执行密钥交换。在提示时输入**root**用户的密码。例如：

```
***Enter password for root@hostname: <root_password>
```

创建gpadmin用户

1. 使用`gpssh`在所有的新Segment主机（如果该用户还不存在）上创建gpadmin用户。使用用户创建的新主机列表来做密钥交换。例如：

```
# gpssh -f new_hosts_file '/usr/sbin/useradd gpadmin -d
/home/gpadmin -s /bin/bash'
```

- 为新的gpadmin用户设置一个密码。在Linux上，用户可以使用gpssh同时在所有的Segment在主机上做这件事情。例如：

```
# gpssh -f new_hosts_file 'echo gpadmin_password |  
passwd  
gpadmin --stdin'
```

- 通过查找gpadmin用户的home目录来验证该用户已经被创建：

```
# gpssh -f new_hosts_file ls -l /home
```

使用gpadmin用户交换SSH密钥

- 使用gpadmin登入，运行gpssh-exkeys工具并引用主机列表文件。例如：

```
# gpssh-exkeys -e /home/gpadmin/existing_hosts_file -x  
/home/gpadmin/new_hosts_file
```

- gpssh-exkeys将会检查远程主机并且在所有的主机之间执行密钥交换。在提示时输入gpadmin用户的密码。例如：

```
***Enter password for gpadmin@hostname:  
<gpadmin_password>
```

验证OS设置

使用gpcheck工具来验证用户的阵列中所有的新主机都有正确的OS设置以运行Greenplum数据库软件。

运行gpcheck

- 使用将要运行Greenplum数据库系统的用户登录master主机（例如：gpadmin）。

```
$ su - gpadmin
```

- 使用用户的新主机的主机文件运行gpcheck工具。例如：

```
$ gpcheck -f new_hosts_file
```

确认磁盘I/O和内存带宽

使用[gpcheckperf](#)工具来测试磁盘I/O和内存带宽。

运行gpcheckperf

1. 使用新主机的主机文件运行gpcheckperf工具。在每一台主机上使用-d选项指定用户想要测试的文件系统。用户必须对这些目录具有写访问权限。例如：

```
$ gpcheckperf -f new_hosts_file -d /data1 -d /data2 -v
```

2. 该工具可能会花长时间来执行测试，因为它需要在主机之间拷贝非常大的文件。当它完成时，用户将会看到磁盘写、磁盘读以及流测试的摘要结果。

对于划分成子网的网络，用每个子网单独的主机文件重复这个过程。

整合新硬件到系统中

在用新Segment初始化系统之前，先用gpstop关闭系统以防止用户活动使性能测试结果倾斜。然后，使用包括所有节点（现有的和新加的）的主机文件重复性能测试：

- 验证OS设置
- 确认磁盘I/O和内存带宽

Greenplum数据库® 6.0文档

□ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库

发布版本号

启动和停止Greenplum数据库

□ 访问数据库

□ 配置Greenplum数据库系统

启用压缩

□ 启用高可用和数据持久化特征

□ 备份和恢复数据库

□ 扩容Greenplum系统

系统扩容概述

规划Greenplum系统
扩容

准备并增加节点

初始化新节点

重分布表

移除扩容Schema

监控Greenplum系统

日常系统维护任务

使用gpexpand工具创建并初始化新节点实例，并创建扩容schema。

第一次用一个合法文件运行`gpexpand`工具，它会创建并初始化节点实例并创建扩容schema。这些过程完成后，运行`gpexpand`检测扩容schema是否被创建，如果成功则做表重分布。

- 为系统扩容创建一个输入文件
- 运行`gpexpand`初始化新节点
- 回滚一个失败的扩容设置

Parent topic: 扩容Greenplum系统

为系统扩容创建一个输入文件

要开始扩容，`gpexpand`要求一个包含有关新节点和主机信息的输入文件。如果用户运行`gpexpand`但不指定输入文件，该工具会显示一个交互式的问讯来收集所需的信息并且自动创建一个输入文件。

如果用户使用交互式问讯创建输入文件，用户可以在问讯提示符中指定一个含有扩容主机列表的文件。如果用户的平台或者命令shell限制主机列表的长度，使用`-f`指定主机就是唯一的办法。

在交互模式中创建一个输入文件

在用户运行`gpexpand`在交互模式中创建一个输入文件之前，确保用户了解：

- 新主机的数量（或者一个主机文件）
- 新主机名（或者一个主机文件）
- 现有主机中使用的镜像策略（如果有）
- 每个主机要增加的Segment数量（如果有）

该工具会基于这些信息、`dbid`、`content ID`以及`gp_segment_configuration`中存储的数据目录值自动生成一个输入文件，并将该文件保存在当前目录中。



在交互模式中创建一个输入文件

1. 使用将要运行Greenplum数据库的用户登录master主机；例如，`gpadmin`。
2. 运行`gpexpand`。该工具显示关于如何准备一次扩容操作的消息，并且它会提示用户退出或者继续。
可以选择用`-f`指定一个主机文件。例如：

```
$ gpexpand -f /home/gpadmin/new_hosts_file
```

3. 在交互中，输入`Y`以继续。
 4. 除非用户用`-f`指定了一个主机文件，用户会被提示输入主机名。可以输入新扩容主机的主机名组成的由逗号分隔的列表。不要包括接口主机名。例如：
- ```
> sdw4, sdw5, sdw6, sdw7
```
- 如果只对现有主机增加节点，在这个提示符处留一个空行。不要指定`localhost`或者任何现有的主机名。
5. 输入在用户的系统中使用的镜像策略（如果有）。选项是`spread|grouped|none`。默认设置是`grouped`。确保用户有足够的主机用于选中的组策略。更多有关镜像的信息请见[规划Mirror节点](#)。
  6. 输入要增加的主节点的数量（如果有）。默认情况下，新主机会被用与现有主机相同数量的主节点初始化。输入一个大于零的数字可以为每个主机增加节点数量。用户输入的数字将是在所有主机上初始化的额外节点的数量。例如，如果现有每个主机当前有两个节点，输入一个值2会在现有主机上多初始化两个节点，而在新主机上会初始化四个节点。
  7. 如果用户在增加新的主节点，为这些新的节点输入新的主数据目录的根目录。不要指定真实的数据目录名称，目录会由`gpexpand`基于现有数据目录名称自动创建。  
例如，如果用户的现有数据目录像下面这样：

```
/gpdata/primary/gp0
/gpdata/primary/gp1
```

那么输入下面的内容（每一个提示输入一个）来指定两个新的主节点的数据目录：

```
/gpdata/primary
```

```
/gpdata/primary
```

当初始化运行时，该工具会在`/gpdata/primary`下面创建新目录`gp2`以及`gp3`。

- 如果用户在增加新的镜像节点，为这些新的节点输入新的镜像数据目录的根目录。不要指定数据目录名称，目录会由`gpexpand`基于现有数据目录名称自动创建。

例如，如果用户的现有数据目录像下面这样：

```
/gpdata/mirror/gp0
/gpdata/mirror/gp1
```

输入下面的内容（每一个提示输入一个）来指定两个新的镜像节点的数据目录：

```
/gpdata/mirror
/gpdata/mirror
```

当初始化运行时，该工具会在`/gpdata/mirror`下面创建新目录`gp2`以及`gp3`。

这些新节点的主节点和镜像节点根目录必须在主机上存在，并且运行`gpexpand`的用户必须具有在其中创建目录的权限。

在用户已经输入所有要求的信息后，该工具会生成一个输入文件并且把它保存在当前目录中。例如：

```
gpexpand_inputfile_yyyyymmdd_145134
```

## 扩容输入文件格式

除非用户的扩容场景有非典型的需求，请使用交互式问讯过程来创建用户自己的输入文件。

扩容输入文件格式：

```
hostname:address:port:datadir:dbid:content:preferred_role
```

例如：

```
sdw5:sdw5-1:50011:/gpdata/primary/gp9:11:9:p
sdw5:sdw5-2:50012:/gpdata/primary/gp10:12:10:p
sdw5:sdw5-2:60011:/gpdata/mirror/gp9:13:9:m
sdw5:sdw5-1:60012:/gpdata/mirror/gp10:14:10:m
```

对于每一个新的节点，这种格式的扩容输入文件要求下列内容：

Table 1. 用于扩容配置文件的数据

| 参数             | 合法值                           | 描述                                                                                                                                                                       |
|----------------|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hostname       | 主机名                           | 节点主机的主机名。                                                                                                                                                                |
| port           | 一个可用的端口号                      | 节点的数据库监听器端口，在现有节点的端口基数上增加。                                                                                                                                               |
| datadir        | 目录名                           | 根据gp_segment_configuration系统catalog的节点数据目录位置。                                                                                                                            |
| dbid           | 整数。不能与现有的 <i>dbid</i> 值冲突。    | 用于该Segment的数据库ID。用户输入的值应该是从系统目录gp_segment_configuration中显示的现有 <i>dbid</i> 值开始顺序增加的值。例如，要在一个现有10个Segment的阵列 ( <i>dbid</i> 值从1-10) 中增加四个节点，应列出新的 <i>dbid</i> 值11、12、13和14。 |
| content        | 整数。不能和现有的 <i>content</i> 值冲突。 | Segment的内容ID。一个主节点和它的镜像应该具有相同的内容ID，并且从现有的值开始顺序增加。更多信息请见gp_segment_configuration的参考信息中的 <i>content</i> 。                                                                  |
| preferred_role | p   m                         | 决定这个节点是主节点还是镜像。指定p表示主节点，指定m表示镜像。                                                                                                                                         |

## 运行gpexpand初始化新节点

在用户创建了一个输入文件之后，运行gpexpand以初始化新的节点。

### 用一个输入文件运行gpexpand

1. 使用将要运行Greenplum数据库的用户登录master主机；例如，gpadmin。
2. 运行gpexpand工具，通过-i指定输入文件。例如：

```
$ gpexpand -i input_file
```

工具检测扩容schema是否存在。如果*gpexpand schema*存在，在开始一个新扩容操作前通过*gpexpand -c*删除它。参考[移除扩容Schema](#)。

当新的节点被初始化并且扩容schema被创建时，该工具会打印一个成功消息并且退出。

当初始化完成，可以连到数据库查看扩容schema。*gpexpand schema*存在于postgres数据库中。更多的信息请参考[关于扩容Schema](#)。

## 回滚一个失败的扩容设置

可以回滚一个扩容操作（增加节点实例和主机），仅当它失败的时候。

如果扩容在初始化步骤失败，而数据库没有启动，用户必须首先通过运行*gpstart -m*命令以master-only模式重启数据库。

用下列命令人回滚失败的扩容：

```
gpexpand --rollback
```

## 监控集群扩容状态

在任何时候，可以运行带有-x标记的*gpstate*命令来检查集群扩容状态：

```
$ gpstate -x
```

如果扩容schema存在于postgres数据库，*gpstate -x*报告扩容的进度。在扩容的第一阶段，*gpstate*报告节点初始化的进度。在第二阶段，*gpstate*报告表重分布的进度，和重分布是否暂停或运行。

还可以通过查询扩容schema来查看扩容状态。参考[监控表的重分布](#)获取更多信息。

## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号

启动和停  
止Greenplum数据库

## □ 访问数据库

□ 配置Greenplum数据库  
系统

启用压缩

□ 启用高可用和数据持久  
化特征

## □ 备份和恢复数据库

## □ 扩容Greenplum系统

系统扩容概述

规划Greenplum系统  
扩容

准备并增加节点

初始化新节点

## 重分布表

移除扩容Schema

监控Greenplum系统

日常系统维护任务

重新分布表让现有数据在新扩容后的集群上得以平衡。

在创建了扩容schema后，可以使用[gpexpand](#)在整个系统重分布表。应该在低峰期运行，这样该工具的CPU使用率和表锁对其他操作的影响会最小。还要对表进行排名，先对最大或者最重要的表进行重新分布。

**Note:** 在重新分布数据时，Greenplum数据库必须运行在生产模式中。Greenplum数据库不能处于受限模式或者Master模式中。不能指定[gpstart](#)选项-R或者-m来启动Greenplum数据库。

当表重新分布正在进行中时，任何新创建的表会按照普通操作那样被分布在所有的节点上。查询也能访问所有的节点，即使相关的数据还没有被完全分布在新节点上的表中也行。正在被重新分布的表或者分区会被锁定并且不可读写。当其重新分布完成后，常规操作才会继续。

- [表重分布排名](#)
- [使用gpexpand重分布表](#)
- [监控表的重分布](#)

**Parent topic:** 扩容Greenplum系统

## 表重分布排名

对于大型的系统，用户可以控制表重新分布的顺序。调整扩容schema里的rank值可以优先做重度使用的表并最小化对性能的影响。可用的空闲磁盘空间可能会影响表的排名，参见[在大规  
模Greenplum系统中管理重新分布](#)。

要通过在[gpexpand.status\\_detail](#)中更新rank值来为重新分布对表排名，用psql或者另一个被支持的客户端连接到Greenplum数据库。用这样的命令更新[gpexpand.status\\_detail](#):

```
=> UPDATE gpexpand.status_detail SET rank=10;
=> UPDATE gpexpand.status_detail SET rank=1 WHERE rq_name =
'public.lineitem';
=> UPDATE gpexpand.status_detail SET rank=2 WHERE rq_name =
'public.orders';
```

这些命令把所有表的优先级降低到10，然后把排名1分配给`lineitem`并且把排名2分配给`orders`。当表的重新分布开始时，`lineitem`会首先被重新分布，接着是`orders`和`gpexpand.status_detail`中所有的其他表。要从重新分布中排除一个表，将它从`gpexpand.status_detail`中移除。

## 使用gpexpand重分布表

### 使用gpexpand重分布表

1. 作为将要运行用户的Greenplum数据库系统的用户（例如`gpadmin`）登入到Master主机。
2. 运行`gpexpand`工具。用户可以使用`-d`或者`-e`选项来定义扩展会话时限。例如，要运行该工具最多60个连续的小时：

```
$ gpexpand -d 60:00:00
```

该工具会重分布表，直到schema中的最后一个表完成或者它达到了指定的持续时间或者结束时间。当一个会话开始并且结束时，`gpexpand`会在`gpexpand.status`中更新状态和时间。

## 监控表的重分布

用户可以在表重新分布处理的过程中查询扩展方案。视图`gpexpand.expansion_progress`提供了一个当前进度摘要，包括估计的表重新分布率以及估计的完成时间。用户可以查询表`gpexpand.status_detail`来得到每个表的状态信息。

关于使用`gpstate`工具监控扩容整体流程，请参考[监控集群扩容状态](#)。

## 查看扩容状态

在第一个表完成重新分布后，`gpexpand.expansion_progress`会计算其估计值并且基于所有表的重新分布比率刷新它们。每次用户用`gpexpand`开始一次表重新分布会话时计算都会重新开始。要监控

进度，可以用或者另一种受支持的客户端连接到数据  
库，然后用下面这样的命令查询*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)
```

## 查看表状态

表*gpexpand.status\_detail*存储着方案中每一个表有关的状态、最近更新时间和更多情况。要查看一个表的状态，可以用psql或者另一种受支持的客户端连接到Greenplum数据库，然后查询*gpexpand.status\_detail*:

```
=> SELECT status, expansion_started, source_bytes FROM
 gpexpand.status_detail WHERE fq_name = 'public.sales';
 status | expansion_started | source_bytes
-----+-----+-----+
 COMPLETED | 2017-02-20 10:54:10.043869 | 4929748992
(1 row)
```

Greenplum数据库® 6.0文档

- 管理员指南

- Greenplum数据库概念

- 管理一个Greenplum系统

- 关于Greenplum数据库  
发布版本号

- 启动和停  
止Greenplum数据库

- 访问数据库

- 配置Greenplum数据库  
系统

- 启用压缩

- 启用高可用和数据持久  
化特征

- 备份和恢复数据库

- 扩容Greenplum系统

- 系统扩容概述

- 规划Greenplum系统  
扩容

- 准备并增加节点

- 初始化新节点

- 重分布表

- 移除扩容Schema**

- 监控Greenplum系统

- 日常系统维护任务

要在扩容Greenplum集群后进行清理，需要移除扩容Schema。

在扩容操作完成并且验证后，用户可以安全地移除扩容Schema。为了在一个Greenplum系统上运行另一次扩容操作，首先要移除现有的扩容Schema。

## 移除扩容Schema

1. 以运行Greenplum数据库系统的用户身份登录Master(例如，gpadmin)。
2. 用-c选项运行gpexpand工具。例如：

```
$ gpexpand -c
```

Note: 一些系统要求用户按回车两次。

**Parent topic:** 扩容Greenplum系统



## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

关于Greenplum的查询  
处理

## □ 关于GPORCA

## GPORCA概述

## 启用和禁用GPORCA

## 收集根分区统计信息

使用GPORCA时的考  
虑

## GPORCA特性和增强

## GPORCA改变的行为

## GPORCA的限制

判断被使用的查询优  
化器

## 关于统一多级分区表

## 定义查询

GPORCA扩展了Greenplum数据库传统优化器的规划和优化能力。

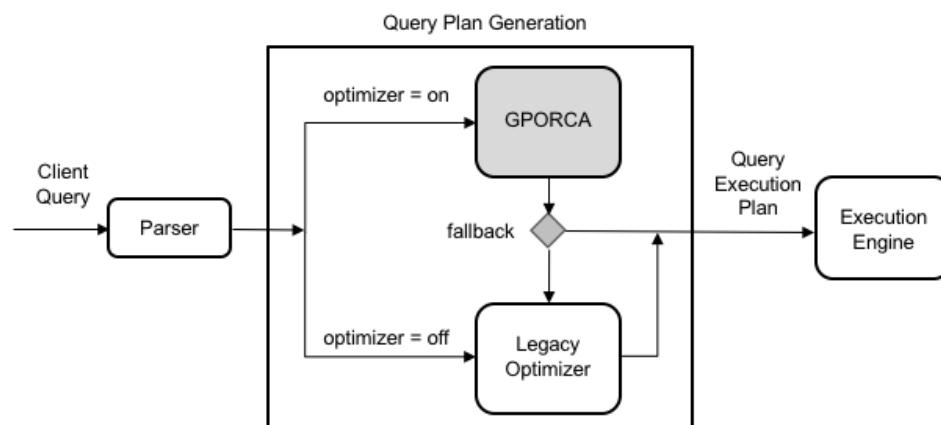
GPORCA是可扩展的，它能在多核环境中获得更好的性能。Greenplum数据库默认使用GPORCA来生成查询计划。

GPORCA也在下列领域增强了Greenplum数据库的查询性能调优：

- 针对分区表的查询
- 包含公共表表达式 (CTE) 的查询
- 包含子查询的查询

在Greenplum数据库中，GPORCA与传统查询优化器并存。默认情况下，Greenplum数据库使用GPORCA。如果无法使用GPORCA，则会使用传统查询优化器。

下图展示了GPORCA如何融合到查询规划架构中。



Note: GPORCA会忽略所有的传统查询优化器服务器参数。然而，如果Greenplum数据库回退到传统查询优化器，优化器服务器配置参数将影响查询计划的生成。关于传统查询优化器的服务器参数，查看[查询调节参数](#)。

**Parent topic:** [关于GPORCA](#)



## Greenplum数据库® 6.0文档

## □ 管理员指南

- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 配置客户端认证

使用  
带TLS/SSL的LDAP认  
证

使用Kerberos认证

为Linux客户端进  
行Kerberos配置为Windows客户端进  
行Kerberos配置

管理角色与权限

- 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

- 查询数据

- 使用外部数据

- 装载和卸载数据

- 性能管理

Greenplum database 5管理员  
指南

可以使用LDAP服务器控制对Greenplum数据库的访问。此外，通过对`pg_hba.conf`文件条目增加参数，可使用加密来保护访问连接。

Greenplum数据库支持在带TLS/SSL协议的LDAP认证，用以加密与LDAP服务器的通信：

- 带STARTTLS和TLS协议的LDAP认证 – STARTTLS以明文连接（无加密）开始，并将其升级为安全连接（使用加密）。
- 带安全连接与TLS/SSL的LDAP认证（LDAPS） – Greenplum数据库可以使用基于LDAP服务器协议的TLS或SSL协议。

如果未指定协议，Greenplum Database将使用明文连接与LDAP服务器通信。

要使用LDAP认证，必须将Greenplum数据库的master节点配置为LDAP客户端。有关配置LDAP客户端的更多信息，请参阅LDAP文档。

## 启用带STARTTLS和TLS协议的LDAP认证

要启用带TLS协议的STARTTLS，在`pg_hba.conf`文件中，增加一个`ldap`行，并制定`ldaptls`参数值为1。默认端口是389。在此示例中，认证方法参数包括`ldaptls`参数。

```
ldap ldapserver=myldap.com ldaptls=1 ldapprefix="uid="
ldapsuffix=",ou=People,dc=example,dc=com"
```

使用指定非默认端口`ldapport`参数。在此示例中，认证方法参数包括`ldaptls`参数，包括指定550端口的`ldapport`参数。

```
ldap ldapserver=myldap.com ldaptls=1 ldapport=500
ldapprefix="uid=" ldapsuffix=",ou=People,dc=example,dc=com"
```

## 启用带安全连接与TLS/SSL的LDAP认证

要启用带TLS/SSL的安全连接，将以`ldaps://`为前缀的LDAP服务器

名称，指定到ldapserver参数上。默认端口是636。

在此示例中，ldapserver参数为LDAP服务器myldap.com指定安全连接及TLS/SSL协议。

```
ldapserver=ldaps://myldap.com
```

要指定非默认端口，请在LDAP服务器名称后添加冒号（:）和端口号。在此示例中example ldapserver参数包括ldaps://前缀及非默认端口550。

```
ldapserver=ldaps://myldap.com:550
```

## 使用系统级的OpenLDAP系统配置认证

如果有一个系统级的OpenLDAP系统，并且在pg\_hba.conf文件中配置使用带TLS或者SSL的LDAP登录，则登录可能会失败并显示以下消息：

```
could not start LDAP TLS session: error code '-11'
```

要使用现有的OpenLDAP系统进行身份验证，Greenplum数据库必须被设置为使用该LDAP服务器的CA证书来验证用户证书。在master主机和standby主机上执行以下步骤以配置Greenplum数据库：

- 将base64编码的根CA链文件从活动目录或LDAP服务器复制到Greenplum数据库的master主机和standby主机。在此示例中，使用/etc/pki/tls/certs目录。
- 切换到复制CA证书文件的目录，并以root用户身份为OpenLDAP生成哈希：

```
cd /etc/pki/tls/certs
openssl x509 -noout -hash -in <ca-certificate-file>
ln -s <ca-certificate-file> <ca-certificate-file>.0
```

- 使用指定的CA证书目录和证书文件为Greenplum数据库配置OpenLDAP配置文件。  
以root用户身份编辑OpenLDAP配置文件/etc/openldap/ldap.conf内容：

```
SASL_NOCANON on
URI ldap://ldapA.example.priv
ldaps://ldapB.example.priv ldaps://ldapC.example.priv
BASE dc=example,dc=priv
TLS_CACERTDIR /etc/pki/tls/certs
TLS_CACERT /etc/pki/tls/certs/<ca-certificate-file>
```

**Note:** 要使证书验证成功，证书中的主机名必须与URI属性中的主机名匹配。否则，您还必须添加TLS\_REQCERT allow参数到文件中。

4. 以gpadmin用户身份，编辑/usr/local/greenplum-db/greenplum\_path.sh并添加以下行。

```
export LDAPCONF=/etc/openldap/ldap.conf
```

## 注释

如果在pg\_hba.conf文件的条目中指定了以下内容，则Greenplum数据库会记录错误：

- 同时指定了ldaps://前缀，和ldaptls=1参数
- 同时指定了ldaps://前缀，和ldapport参数

为LDAP认证启用加密通信，仅加密Greenplum数据库和LDAP服务器之间的通信。

有关加密客户端连接的信息，参考[加密客户端/服务器连接](#)所示。

## 示例

如下是一个pg\_hba.conf文件中的示例条目。

此示例指定Greenplum数据库和LDAP服务器之间没有加密的LDAP认证。

```
host all plainuser 0.0.0.0/0 ldap ldapserver=myldap.com
ldapprefix="uid=" ldapsuffix=",ou=People,dc=example,dc=com"
```

此示例指定Greenplum数据库和LDAP服务器之间的带STARTTLS和TLS协议的LDAP认证。

```
host all tlsuser 0.0.0.0/0 ldap ldapserver=myldap.com
ldaptls=1 ldapprefix="uid="
ldapsuffix=",ou=People,dc=example,dc=com"
```

此示例指定Greenplum数据库和LDAP服务器之间的带安全连接与TLS/SSL的LDAP认证。

```
host all ldapsuser 0.0.0.0/0 ldap
ldapserver=ldaps://myldap.com ldapprefix="uid="
ldapsuffix=",ou=People,dc=example,dc=com"
```

**Parent topic:** [配置客户端认证](#)

## Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 配置客户端认证

使用  
带TLS/SSL的LDAP认  
证

### 使用Kerberos认证

- 为Linux客户端进  
行Kerberos配置
- 为Windows客户端进  
行Kerberos配置
- 管理角色与权限
- 定义数据库对象
- 分布与倾斜
- 插入, 更新, 和删除数据
- 查询数据
- 使用外部数据
- 装载和卸载数据
- 性能管理

Greenplum database 5管理员  
指南

您可以使用Kerberos认证服务器控制对Greenplum数据库的访问。

Greenplum数据库支持使用Kerberos认证的通用安全服务应用程序接口 (GSSAPI)。GSSAPI为支持它的系统提供了自动身份认证（单点登录）功能。可以在Greenplum数据库配置文件`pg_hba.conf`中指定需要Kerberos身份认证的Greenplum数据库用户或角色。当角色尝试登录Greenplum数据库时，如果Kerberos身份认证不可用，则登录失败。

Kerberos提供一种安全的、加密的认证服务。它不加密客户端和数据库之间传输的数据，不提供授权服务。如果要网络传输的数据，可以使用SSL连接。如果要管理Greenplum数据库及对象的访问权限，可以在`pg_hba.conf`文件中进行设置，为数据库中的Greenplum数据库用户和角色的授权。更多有关Kerberos的信息，可以参考[管理角色与权限](#)。

For more information about Kerberos, see <http://web.mit.edu/kerberos/> □。

## 先决条件

在为Greenplum数据库配置Kerberos身份认证之前，请确保

- 您可以识别用于Kerberos身份认证的KDC服务器和用于Greenplum数据系统的Kerberos域。
  - 如果您计划使用MIT Kerberos的KDC服务器但尚未配置它，参阅[安装和配置Kerberos KDC服务器](#)的示例说明
  - 如果您使用现有的Active Directory KDC服务器，须确保具有：
    - 在AD KDC服务器上安装所有Active Directory服务角色。
    - 启用LDAP服务。
- 系统时间在Kerberos密钥分发中心（KDC）服务器和Greenplum数据库主服务器之间是同步的。例如，在两台服务器上安装NTPntp软件包。
- 在KDC服务器和Greenplum数据库master主机之间存在网络连接。
- 所有Greenplum数据库主机上都安装了Java 1.7.0\_17或更高版本。在Red Hat Enterprise Linux 6.x或7.x上使用要使用Kerberos认证的JDBC，需要安装Java 1.7.0\_17或更高版本。

## 步骤

以下是完成为Greenplum数据库设置Kerberos身份认证的任务步骤。

[KDC](#)

[Greenplum](#)

[Principal](#)

## 在 ..... 数据库中创建 ..... 数据库的主体 (.....)

- 在master主机安装Kerberos客户端
- 使用Kerberos身份认证配置Greenplum数据库
- 将Kerberos主体映射为Greenplum数据库角色
- 为Greenplum数据库配置JDBC的Kerberos身份认证
- 在Windows上为Greenplum数据库客户端配置Kerberos
- 使用活动目录配置客户端身份认证

**Parent topic:** 配置客户端认证

# 在KDC数据库中创建Greenplum数据库的主体 (Principal)

创建服务主体，为Greenplum数据库服务和Kerberos管理主体，用于以gpadmin用户身份管理KDC数据库。

1. 以root用户身份登录Kerberos KDC服务器。

```
$ ssh root@<kdc-server>
```

2. 为Greenplum数据库服务创建一个主体。

```
kadmin.local -q "addprinc -randkey postgres/mdw@GPDB.KRB"
```

-randkey选项可防止命令提示输入密码。

主体名称中postgres部分，匹配Greenplum数据库krb\_srvname配置参数的值，默认情况值为postgres即可。

主体名称中主机名部分（译者注，示例中的mdw部分），必须匹配Greenplum数据库master主机上hostname命令的输出结果。如果hostname命令显示的是完全限定域名（FQDN），例如显示mdw.example.com，应用到主体名称则为postgres/mdw.example.com@GPDB.KRB。

主体名称中GPDB.KRB部分，是Kerberos域（realm）的名称

3. 为gpadmin/admin角色创建一个主体。

```
kadmin.local -q "addprinc gpadmin/admin@GPDB.KRB"
```

此主体允许您以gpadmin身份登录时，管理KDC数据库。确保Kerberos的kadm.ac1配置文件包含用于向此主体授予权限的ACL。例如，此ACL为GPDB.KRB域中所有管理员用户，授予所有权限。

```
* /admin@GPDB.KRB *
```

4. 使用kadmin.local创建密钥表（keytab）文件。以下示例在当前目录中创建密钥表文件gpdb-kerberos.keytab，并包含Greenplum数据库服务主体和gpadmin/dmin主体的身份认证信息。

```
kadmin.local -q "ktadd -k gpdb-kerberos.keytab
postgres/mdw@GPDB.KRB gadmin/admin@GPDB.KRB"
```

5. 将密钥表文件复制到master主机。

```
scp gpdb-kerberos.keytab gadmin@mdw:~
```

## 在master主机安装Kerberos客户端

在Greenplum数据库master服务器上安装Kerberos客户端实用程序和依赖库。

1. 在Greenplum数据库master服务器上安装Kerberos软件包。

```
$ sudo yum install krb5-libs krb5-workstation
```

2. 从KDC服务器拷贝/etc/krb5.conf文件到Greenplum数据库master主机相同目录下。

## 使用Kerberos身份认证配置Greenplum数据库

使用Kerberos配置Greenplum数据库。

1. 以gadmin用户身份登录Greenplum数据库master主机。

```
$ ssh gadmin@<master>
$ source /usr/local/greenplum-db/greenplum_path.sh
```

2. 设置从KDC服务器复制的密钥表文件的所有者和权限。

```
$ chown gadmin:gadmin /home/gadmin/gpdb-kerberos.keytab
$ chmod 400 /home/gadmin/gpdb-kerberos.keytab
```

3. 通过设置Greenplum数据库krb\_server\_keyfile服务器配置参数，配置密钥表文件的位置。该gpconfig命令指定将/home/gadmin文件

夹为为密钥表文件gpdb-kerberos.keytab的位置。

```
$ gpconfig -c krb_server_keyfile -v '/home/gpadmin/gpdb-kerberos.keytab'
```

- 修改Greenplum数据库的pg\_hba.conf文件，启用Kerberos支持。例如，添加以下行到pg\_hba.conf文件，为来自同一网络上的所有用户和主机的连接请求添加GSSAPI和Kerberos身份认证支持。

```
host all all 0.0.0.0/0 gss include_realm=0
krb_realm=GPDB.KRB
```

设置krb\_realm选项为某个域名称，确保只有该域下的用户才能使用Kerberos成功进行身份认证。设置include\_realm选项为0为身份认证通过的用户名去除域部分。关于pg\_hba.conf文件更多信息，参阅PostgreSQL文档中的[关于pg\\_hba.conf文件](#) 内容。

- 更新krb\_server\_keyfile和pg\_hba.conf后，重启Greenplum数据库。

```
$ gpstop -ar
```

- 创建Greenplum数据库超级用户角色gpadmin/admin。

```
$ createuser gpadmin/admin
Shall the new role be a superuser? (y/n) y
```

此数据库角色的Kerberos密钥文件可以从KDC服务器复制。

- 使用kinit创建票据，使用klist显示Kerberos票证缓存中所有票据。

```
$ LD_LIBRARY_PATH= kinit -k -t /home/gpadmin/gpdb-kerberos.keytab gpadmin/admin@GPDB.KRB
$ LD_LIBRARY_PATH= klist
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: gpadmin/admin@GPDB.KRB

Valid starting Expires Service principal
06/13/2018 17:37:35 06/14/2018 17:37:35 krbtgt/GPDB.KRB@GPDB.KRB
```

LD\_LIBRARY\_PATH环境变量设置为包含Greenplum数据库LIB目录，包括Kerberos库。这可能会导致Kerberos应用程序命令，如使用kinit和klist的由于版本冲突而失败。解决方案是在您获取Kerberos应用程序之前运行Kerberos应用程序greenplum-db\_path.sh文件或暂时取消设置LD\_LIBRARY\_PATH执行Kerberos应用程序时的变量，如示例中所示。

Note: 通过加载greenplum-db\_path.sh脚本来设置Greenplum数据库环境时，环境变量LD\_LIBRARY\_PATH被设置为包含Greenplum数据库

的lib目录，目录中包含Kerberos库文件。可能会导致Kerberos工具命令，如kinit和klist因为版本冲突而执行失败。解决方案是在加载greenplum-db\_path.sh脚本之前运行Kerberos工具程序，或暂时取消设置LD\_LIBRARY\_PATH变量再执行如上Kerberos工具程序。

#### 8. 作为测试，使用gpadmin/admin角色登录postgres数据库：

```
$ psql -U "gpadmin/admin" -h mdw postgres
psql (9.4.20)
Type "help" for help.

postgres=# select current_user;
current_user

gpadmin/admin
(1 row)
```

Note: 当您在master主机启动psql时，必须包含-h <master-hostname>选项，强制使用TCP连接，因为使用本地连接时Kerberos认证不生效。

如果Kerberos主体不是Greenplum数据库用户，用户尝试登录数据库时，则会显示类似的以下内容的消息显示在的psql命令行：

```
psql: krb5_sendauth: Bad response
```

必须将主体添加为Greenplum数据库用户。

## 将Kerberos主体映射为Greenplum数据库角色

要连接到启用了Kerberos身份认证的Greenplum数据库系统，用户首先使用kinit命令从KDC服务器请求授予票证，期间需要输入密码或者提供密钥表文件。随后当用户连接到启用Kerberos的Greenplum数据库系统时，用户的Kerberos主题名称将是Greenplum数据库角色名称。具体名称根据Greenplum数据库pg\_hba.conf中受到在gss选项进行转换：

- 如果krb\_realm=<realm>选项存在，Greenplum数据库只接收指定域名下的Kerberos主体。
- 如果include\_realm=0选项存在，the Greenplum数据库角色名是Kerberos主体名去除Kerberos域后部分。如果include\_realm=1选项被指定，Kerberos域不会从Greenplum数据库角色名中被剔除。角色名必须使用Greenplum数据库的CREATE ROLE命令创建。
- 如果map=<map-name>选项被指定，将Kerberos主体名称与\$MASTER\_DATA\_DIRECTORY/pg\_ident.conf中指定的<map-

name>标记匹配，根据第一个匹配项替换为Greenplum数据库角色名。

### 用户名映射被定义

在\$MASTER\_DATA\_DIRECTORY/pg\_ident.conf配置文件。此示例定义了一个名为的映射MyMap中有两个条目。本例中使用两条mymap记录，定义了两个名称映射。

```
MAPNAME SYSTEM-USERNAME GP-USERNAME
mymap /^admin@GPDB.KRB$ gpadmin
mymap /^(.*)_gp@GPDB.KRB$ \1
```

名称映射被指定在pg\_hba.conf文件的Kerberos记录的可选项中

```
host all all 0.0.0.0/0 gss include_realm=0 krb_realm=GPDB.KRB
map=mymap
```

第一个映射条目匹配Kerberos主体admin@GPDB.KRB，并替换为Greenplum数据库的gpadmin角色名。第二个条目使用通配符匹配GPDB.KRB域下名称是\_gp结尾的的Kerberos主体，并替换为Kerberos主体名称的开始部分（译者注：\_gp前面的部分）。Greenplum数据库使用pg\_ident.conf文件中第一个匹配的映射条目，因此条目的顺序很重要。

有关使用用户名映射的更多信息，请参阅PostgreSQL的文档的[用户名映射](#)。

## 为Greenplum数据库配置JDBC的Kerberos身份认证

使用JDBC访问启用Kerberos证的Greenplum数据库。

您可以配置Greenplum数据库使用Kerberos运行用户定义的Java函数。

1. 确保在Greenplum数据库主机中，已经安装并配置了Kerberos。参阅[在master主机安装Kerberos客户端](#)。
2. 在/home/gpadmin文件夹下创建.java.login.config文件，并在文件中添加如下内容：

```
pgjdbc {
 com.sun.security.auth.module.Krb5LoginModule required
 doNotPrompt=true
 useTicketCache=true
 debug=true
 client=true;
};
```

- 创建一个Java应用，使用Kerberos认证连接到Greenplum数据库。如下示例中数据库连接URL使用PostgreSQL的JDBC驱动，并指定了用于Kerberos认证的参数：

```
jdbc:postgresql://mdw:5432/mytest?
kerberosServerName=postgres
&jaasApplicationName=pg jdbc&user=gpadmin/gpdb-kdc
```

指定的参数名和参数值取决于Java应用如何执行Kerberos认证。

- 运行Java应用示例，测试使用Kerberos登录Greenplum数据库。

## 安装和配置Kerberos KDC服务器

在Red Hat Enterprise Linux上，为Greenplum设置Kerberos密钥分发中心(KDC)服务器的步骤。

如果您没有KDC，参考如下步骤，在Red Hat Enterprise Linux主机上安装配置一个KDC服务，其域为GPDB.KRB。本示例中KDC服务器的主机名是gpdb-kdc。

- 安装Kerberos服务器客户端软件包：

```
$ sudo yum install krb5-libs krb5-server krb5-workstation
```

- 编辑/etc/krb5.conf配置文件。如下示例展示的包含默认GPDB.KRB域配置的Kerberos服务。

```
[logging]
default = FILE:/var/log/krb5libs.log
kdc = FILE:/var/log/krb5kdc.log
admin_server = FILE:/var/log/kadmind.log

[libdefaults]
default_realm = GPDB.KRB
dns_lookup_realm = false
dns_lookup_kdc = false
ticket_lifetime = 24h
renew_lifetime = 7d
forwardable = true
default_tgs_enctypes = aes128-cts des3-hmac-sha1 des-cbc-crc des-cbc-md5
default_tkt_enctypes = aes128-cts des3-hmac-sha1 des-cbc-crc des-cbc-md5
permitted_enctypes = aes128-cts des3-hmac-sha1 des-cbc-crc des-cbc-md5

[realms]
GPDB.KRB = {
 kdc = gpdb-kdc:88
```

```

admin_server = gpdb-kdc:749
default_domain = gpdb.krb
}

[domain_realm]
.gpdb.krb = GPDB.KRB
gpdb.krb = GPDB.KRB

[appdefaults]
pam = {
 debug = false
 ticket_lifetime = 36000
 renew_lifetime = 36000
 forwardable = true
 krb4_convert = false
}

```

在[realms]分布，kdc和admin\_server键指定了Kerberos服务运行的域名（gpdb-kdc）和端口。可以用IP地址来代替主机名。

如果该Kerberos服务器还管理者其他域的认证，需要在kdc.conf文件[realms]和[domain\_realm]部分添加GPDB.KRB域配置。关于kdc.conf文件更多信息，参阅[Kerberos文档](#)。

### 3. 运行kdb5\_util，创建Kerberos数据库。

```
kdb5_util create -s
```

其kdb5\_util的create子命令创建数据库，为KDC服务器所管理的Kerberos域存储密钥。其-s选项创建一个隐藏文件。如果没有这个隐藏文件，每次启动KDC服务时都需要输入密码

### 4. 使用kadmin.local工具增加一个管理员用户到KDC数据库中。因为工具本身并不依赖于Kerberos认证，kadmin.local工具允许您为本地的Kerberos服务器增加一个初始的管理员用户。运行如下命令，添加用户gpadmin作为管理员用户到KDC数据库中：

```
kadmin.local -q "addprinc gpadmin/admin"
```

大部分用户不需要对Kerberos服务器的管理访问。他们可以使用户kadmin来管理自己的主体（例如，更改自己的密码）。关于kadmin的信息，参阅[Kerberos文档](#)。

### 5. 如果需要，编辑 /var/kerberos/krb5kdc/kadm5.acl文件，为gpadmin授予适当的权限。

### 6. 启动Kerberos守护进程：

```
/sbin/service krb5kdc start#
/sbin/service kadmin start
```

## 7. Kerberos

设置

自启动:

```
/sbin/chkconfig krb5kdc on
/sbin/chkconfig kadmin on
```

# 行Kerberos配置

Greenplum数据库® 6.0文档

- 管理员指南
- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问

## 配置客户端认证

使用  
带TLS/SSL的LDAP认  
证

使用Kerberos认证

为Linux客户端进  
行Kerberos配置

为Windows客户端进  
行Kerberos配置

管理角色与权限

## 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

## 查询数据

## 使用外部数据

## 装载和卸载数据

## 性能管理

Greenplum database 5管理员  
指南

您可以配置Linux client客户端应用来连接一个已配置Kerberos认证的Greenplum数据库系统。

如果在Red Hat Enterprise Linux, JDBC连接Greenplum数据库时, 使用Kerberos认证。在客户端系统必须被配置为使用Kerberos认证。如果不使用Kerberos认证连接Greenplum数据库, 在客户端系统中不需要Kerberos。

- [要求](#)
- [设置客户端系统使用Kerberos身份认证](#)
- [运行Java应用程序](#)

有关在Greenplum数据库中启用Kerberos认证的信息, 参阅Greenplum数据库管理员指南中的“使用Kerberos认证”部分。

**Parent topic:** [配置客户端认证](#)

## 要求

如下是从客户端系统使用JDBC应用连接到已启用Kerberos认证的Greenplum数据库的要求。

- [先决条件](#)
- [客户端机器上的必需软件](#)

## 先决条件

- 在Greenplum数据库master主机上必须安装并配置好Kerberos。  
Important: Greenplum数据库必须被配置完毕, 这样远程用户才能使用Kerberos认证连接到Greenplum数据库。授权访问Greenplum Database数据库, 通过pg\_hba.conf文件控制。关于细节, 参阅Greenplum数据库管理员指南中的“编辑pg\_hba.conf文件”部分。
- The client system requires the Kerberos configuration file [krb5.conf](#) from the Greenplum Database master.
- 客户端系统从Greenplum数据库的Master请求Kerberos配置文件krb5.conf。客户端系统需要Kerberos密钥表文件, 该文件包含用

于登录数据库的Greenplum数据库用户的身份认证凭据。

- 客户端机器必须能够连接到Greenplum数据库master主机。  
如有必要，将Greenplum数据库master主机名和IP地址添加到系统的hosts文件。在Linux系统上，hosts文件在/etc目录下。

## 客户端机器上的必需软件

- 在客户端机器上需要Kerberos的kinit工具。该kinit在安装安装Kerberos软件包时即可获得：
  - krb5-libs
  - krb5-workstation

Note: 安装Kerberos软件包时，可以使用其他Kerberos的klist工具显示Kerberos票证信息。

Java应用程序需要如下额外软件：

- Java JDK  
Red Hat Enterprise Linux 6.x支持Java JDK 1.7.0\_17。
- 确保将JAVA\_HOME设置为Java JDK支持的安装目录。

## 设置客户端系统使用Kerberos身份认证

要使用Kerberos身份认证连接到Greenplum数据库，需要Kerberos票证。在客户端系统上，使用kinit工具生成Kerberos的密钥表文件生成票证并存储在缓存文件中。

1. 安装Greenplum数据库master主机中Kerberos配置文件krb5.conf的副本。该文件可用于Greenplum数据库客户端软件和Kerberos实用程序。  
安装krb5.conf文件到/etc目录。  
如果需要，增加default\_ccache\_name参数到krb5.ini文件的[libdefaults]部分，用于指定Kerberos票据缓存文件在客户端系统中位置。
2. 获取包含Greenplum数据库用户的身份认证凭据的Kerberos密钥表文件。
3. 运行kinit指定密钥表文件在客户端计算机上创建票证。对于此示例，密钥表文件gpdb-kerberos.keytab在当前目录中。票

凭证缓存文件位于gpadmin用户家目录。

```
> kinit -k -t gpdb-kerberos.keytab -
c /home/gpadmin/cache.txt
gpadmin/kerberos-gpdb@KRB.EXAMPLE.COM
```

## 运行psql

您可以远程系统访问启用了Kerberos身份认证的Greenplum数据库。

## 使用psql连接到Greenplum数据库

1. 作为gpadmin用户打开命令窗口。
2. 从命令窗口启动psql，指定与Greenplum数据库的连接，使用Kerberos身份验证配置的用户。  
例如，使用Kerberos主体gpadmin/kerberos-gpdb，以gpadmin用户身份，登录到计算机上的Greenplum数据库：

```
$ psql -U "gpadmin/kerberos-gpdb" -h kerberos-gpdb
postgres
```

## 运行Java应用程序

使用Java身份验证和授权服务 (JAAS)，通过Java应用程序访问已启用Kerberos身份验证的Greenplum数据库。

1. 在用户家目录创建.java.login.config文件控制。  
例如，在Linux系统，家目录类似于/home/gpadmin。  
添加如下内容到文件中：

```
pgjdbc {
 com.sun.security.auth.module.Krb5LoginModule required
 doNotPrompt=true
 useTicketCache=true
 ticketCache = "/home/gpadmin/cache.txt"
 debug=true
 client=true;
};
```

## 2. 创建Java应用程序连接到已启用Kerberos身份验证的Greenplum数据库的，并以用户身份运行应用程序。

此示例数据库连接URL使用PostgreSQL的JDBC驱动程序并指定Kerberos身份验证的参数。

```
jdbc:postgresql://kerberos-gpdb:5432/mytest?
kerberosServerName=postgres&jaasApplicationName=pgjdbc&
user=gpadmin/kerberos-gpdb
```

指定的参数名称和值取决于Java应用程序如何执行Kerberos身份验证。

# 运行Kerberos配置

Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 配置客户端认证

使用  
带TLS/SSL的LDAP认  
证

使用Kerberos认证

为Linux客户端进  
行Kerberos配置

为Windows客户端进  
行Kerberos配置

管理角色与权限

- 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

- 查询数据

- 使用外部数据

- 装载和卸载数据

- 性能管理

Greenplum database 5管理员  
指南

您可以配置微软的Windows客户端应用程序来连接一个已配  
置Kerberos认证的Greenplum数据库系统。

- [在Windows上为Greenplum数据库客户端配置Kerberos](#)
- [使用活动目录配置客户端身份认证](#)

有关在Greenplum数据库中启用Kerberos认证的信息，参阅[使  
用Kerberos认证](#)。

**Parent topic:** [配置客户端认证](#)

## 在Windows上为Greenplum数据库客 户端配置Kerberos

当一个Greenplum数据库系统被配置为启用Kerberos作认证后，您可以  
为微软Windows系统上的Greenplum数据库客户端工  
具gplload和psql配置Kerberos认证。Greenplum数据库直接使  
用Kerberos认证，而不使用微软的活动目录（AD）。

本节包含以下信息。

- [在Windows系统上安装Kerberos](#)
- [运行psql工具](#)
- [gupload示例YAML文件](#)
- [创建一个Kerberos的密钥表文件](#)
- [问题及可能的解决方案](#)

这些主题假设Greenplum数据库系统已经被配置为启用Kerberos和微软  
的活动目录认证。参阅[使用活动目录配置客户端身份认证](#)。

## 在Windows系统上安装Kerberos

要对Windows系统上的Greenplum数据库客户端使用Kerberos认证，该  
系统上必须安装MIT Kerberos Windows客户端。可以  
从<http://web.mit.edu/kerberos/dist/index.html> 下载安装MIT Kerberos

for Windows 4.0.1 (for krb5)。

在Windows系统上，可以使用Kerberos的kinit工具管理Kerberos的票据。

Kerberos服务没有被设置为自启动。该服务不能被用来认证Greenplum数据库。

从Greenplum数据库master主机拷贝/etc/krb5.conf，并放置到Windows系统Kerberos默认路径C:\ProgramData\MIT\Kerberos5\krb5.ini。在文件的[libdefaults]部分，移除Kerberos的票据缓冲的位置default\_ccache\_name。

在Windows系统上，使用环境变量KRB5CCNAME指定Kerberos票据的位置。环境变量的值是一个文件而不是目录，并且它应该对服务器上的每个登录都是唯一的。

如下示例是配置文件，其中default\_ccache\_name被移除，[logging]部分也被移除。

```
[libdefaults]
debug = true
default_etypes = aes256-cts-hmac-sha1-96
default_realm = EXAMPLE.LOCAL
dns_lookup_realm = false
dns_lookup_kdc = false
ticket_lifetime = 24h
renew_lifetime = 7d
forwardable = true

[realms]
EXAMPLE.LOCAL =
 kdc =bocdc.example.local
 admin_server = bocdc.example.local
}

[domain_realm]
.example.local = EXAMPLE.LOCAL
example.local = EXAMPLE.LOCAL
```

使用KRB5CCNAME指定Kerberos票据时，可以在一个本地用户环境或者一个会话中指定该值。如下命令设置KRB5CCNAME，然后运行kinit，并且运行批处理文件来为Greenplum数据库的客户端设置环境变量。

```
set KRB5CCNAME=%USERPROFILE%\krb5cache
kinit
```

```
"c:\Program Files (x86)\Greenplum\greenplum-clients-<version>\greenplum_clients_path.bat"
```

## 运行psql工具

在Windows系统上安装并且配置Kerberos及其票据之后，可以运行Greenplum数据库的命令行客户端psql。

如果得到警告指出控制台代码页与Windows代码页不同，可以运行Windows工具chcp来改变代码页。如下示例展示了该警告及修正方法。

```
psql -h prod1.example.local warehouse
psql (9.4.20)
WARNING: Console code page (850) differs from Windows code
page (1252)
8-bit characters might not work correctly. See psql
reference
page "Notes for Windows users" for details.
Type "help" for help.

warehouse=# \q

chcp 1252
Active code page: 1252

psql -h prod1.example.local warehouse
psql (9.4.20)
Type "help" for help.
```

## 创建一个Kerberos的密钥表文件

在连接到Greenplum数据库系统时，可以创建并使用一个Kerberos的keytab文件来避免在命令行输入密码或者在脚本文件中列出密码。可以用这些工具创建密钥表文件：

- Windows Kerberos工具ktmppass
- Java JRE密钥表工具ktab

如果使用AES256-CTS-HMAC-SHA1-96加密，需要从Oracle下载并安装Java扩展*Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files for JDK/JRE*。该命令创建密钥表文件 `svcPostgresProd1.keytab`。

作为一个AD域管理员运行ktmppass工具。该工具需要一个包含服务主体名称 (SPN) 的用户账户作为AD用户属性的，该参数不是必需的。

可以设置它作为ktab的一个参数并忽略未设置的警告。

Java JRE的ktab工具需要AD域名管理员，不需要SPN。

Note: 输入密码以创建密钥表文件时，屏幕上会显示密码。

该示例运行ktab工具创建密钥表 dev1.keytab。

```
ktab -out dev1.keytab -princ dev1@EXAMPLE.LOCAL -mapUser
dev1 -pass your_password -crypto all -ptype
KRB5_NT_PRINCIPAL
```

尽管有警告信息Unable to set SPN mapping data仍然可用。

该示例运行Javaktab.exe创建密钥表文件（-a选项）并列出密钥表名称和条目（-l -e -t选项）。

```
C:\Users\dev1>"\Program Files\Java\jre1.8.0_77\bin"\ktab -a
dev1
Password for dev1@EXAMPLE.LOCAL:your_password
Done!
Service key for dev1 is saved in C:\Users\dev1\krb5.keytab

C:\Users\dev1>"\Program Files\Java\jre1.8.0_77\bin"\ktab -l
-e -t
Keytab name: C:\Users\dev1\krb5.keytab
KVNO Timestamp Principal

4 13/04/16 19:14 dev1@EXAMPLE.LOCAL (18: AES256 CTS mode
with HMAC SHA1-96)
4 13/04/16 19:14 dev1@EXAMPLE.LOCAL (17: AES128 CTS mode
with HMAC SHA1-96)
4 13/04/16 19:14 dev1@EXAMPLE.LOCAL (16: DES3 CBC mode with
SHA1-KD)
4 13/04/16 19:14 dev1@EXAMPLE.LOCAL (23: RC4 with HMAC)
```

然后，您可以使用该密钥表：

```
kinit -kt dev1.keytab dev1
or
kinit -kt %USERPROFILE%\krb5.keytab dev1
```

## gupload示例YAML文件

该示例以dev1用户身份，运行gupload任务，用户通过AD域登录到Windows桌面。

示例中的test.yaml控制文件，其USER:行已经被移除。使用了Kerberos认证

```

VERSION: 1.0.0.1
DATABASE: warehouse
HOST: prod1.example.local
PORT: 5432

GPLOAD:
 INPUT:
 - SOURCE:
 PORT_RANGE: [18080,18080]
 FILE:
 - /Users/dev1/Downloads/test.csv
 - FORMAT: text
 - DELIMITER: ','
 - QUOTE: '"'
 - ERROR_LIMIT: 25
 - LOG_ERRORS: true
 OUTPUT:
 - TABLE: public.test
 - MODE: INSERT
 PRELOAD:
 - REUSE_TABLES: true
```

如下命令运行kinit，然后用test.yaml文件运行gpload，并成功显示gpload输出信息。

```
kinit -kt %USERPROFILE%\krb5.keytab dev1

gpload.py -f test.yaml
2016-04-10 16:54:12|INFO|gpload session started 2016-04-10
16:54:12
2016-04-10 16:54:12|INFO|started gpfdist -p 18080 -P 18080
-f "/Users/dev1/Downloads/test.csv" -t 30
2016-04-10 16:54:13|INFO|running time: 0.23 seconds
2016-04-10 16:54:13|INFO|rows Inserted = 3
2016-04-10 16:54:13|INFO|rows Updated = 0
2016-04-10 16:54:13|INFO|data formatting errors = 0
2016-04-10 16:54:13|INFO|gpload succeeded
```

## 问题及可能的解决方案

- 这个消息表示Kerberos无法找到缓冲文件：

```
Credentials cache I/O operation failed XXX
(Kerberos error 193)
krb5_cc_default() failed
```

确保Kerberos能找到该文件，设置环境变量 KRB5CCNAME然后运行 kinit。

```
set KRB5CCNAME=%USERPROFILE%\krb5cache
kinit
```

- 该kinit消息表示kinit -k -t 命令无法找到密钥表文件。

```
kinit: Generic preauthentication failure while getting
initial credentials
```

确认该Kerberos密钥表文件的完整路径和文件名是正确的。

## 使用活动目录配置客户端身份认证

可以使用微软活动目录 (AD) 帐户配置微软Windows用户，以便单点登录Greenplum数据库系统。

配置AD用户帐户以支持使用Kerberos身份认证登录。

通过AD单点登录，Windows用户可以将活动目录凭据与Windows客户端应用程序一起使用，以登录Greenplum数据库系统。针对使用ODBC的Windows应用程序，ODBC驱动程序可以使用活动目录凭据连接到Greenplum数据库系统。

Note: Windows上运行的Greenplum数据库客户端，如gpload，接与Greenplum数据库连接，不要使用微软活动目录。有关将Windows上的Greenplum数据库客户端连接到启用Kerberos身份认证的Greenplum数据库系统的信息，请参阅[在Windows上为Greenplum数据库客户端配置Kerberos](#)。

本章节包含以下信息。

- [先决条件](#)
- [设置活动目录](#)
- [为活动目录设置Greenplum数据库](#)
- [单点登录示例](#)
- [活动目录的问题及可能的解决方案](#)

## 先决条件

这些项需要启用AD单点登录到Greenplum数据库系统。

- 必须将Greenplum数据库系统配置为支持Kerberos身份认证。有关使用Kerberos身份认证配置Greenplum数据库的信息，请参阅[为Windows客户端进行Kerberos配置](#)。
- 必须知道Greenplum数据库master主机的完全限定域名（FQDN）。同时，Greenplum数据库master主机名必须具有域部分。如果系统没有域，则必须将系统配置为使用域。  
该Linux的hostname命令显示FQDN信息。

```
hostname --fqdn
```

- 必须确认Greenplum数据库系统与活动目录域具有相同的日期和时间。例如，您可以将Greenplum Database系统NTP时间源设置为AD域控制器，或配置master主机和AD域控制器使用相同的外部时间源。
- 要支持单点登录，需要将一个AD作为AD的托管服务账号。如下是对于Kerberos认证的要求。
  - 需要添加增加服务主体名称（SPN）属性到用户账户信息中，因为Kerberos工具在Kerberos认证时需要这些信息。
  - 此外，由于Greenplum数据库需要无人值守启动，必须在一个Kerberos密钥表文件中提供账号登录细节。

Note: 设置SPN和创建密钥表所需的AD管理权限。

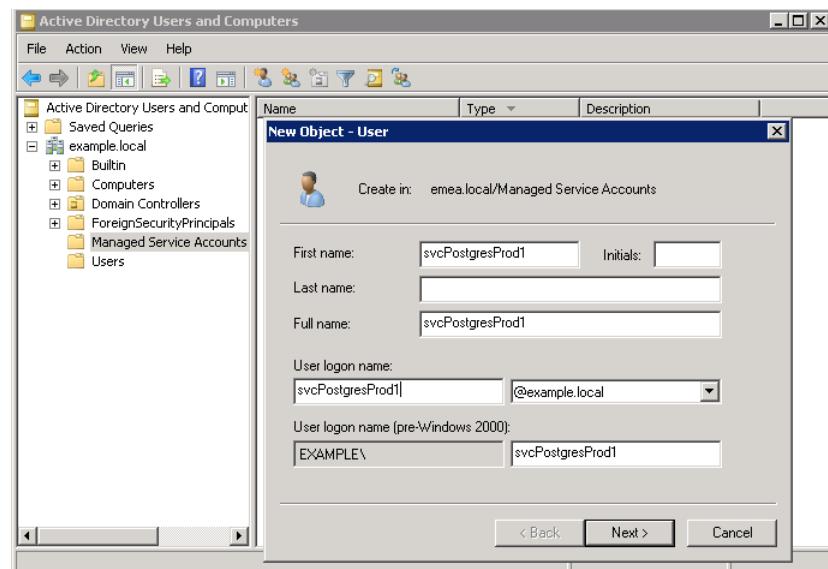
## 设置活动目录

AD命名约定应支持多个Greenplum数据库系统。在此示例中，为Greenplum数据库系统master主机prod1创建了一个新的AD托管服务帐户svcPostresProd1。

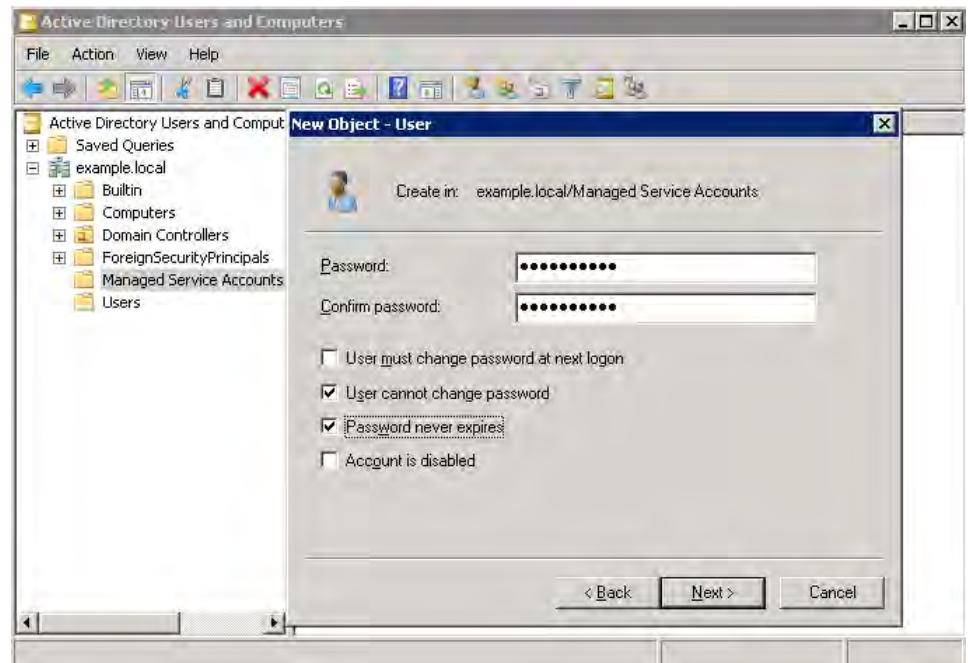
其活动目录的域为example.local。

Greenplum数据库master主机的完全限定域名是prod1.example.local。

添加SPN为postgres/prod1.example.local到这个帐户。其他Greenplum数据库系统的服务帐户都是postgres/*fully qualified hostname*的形式。



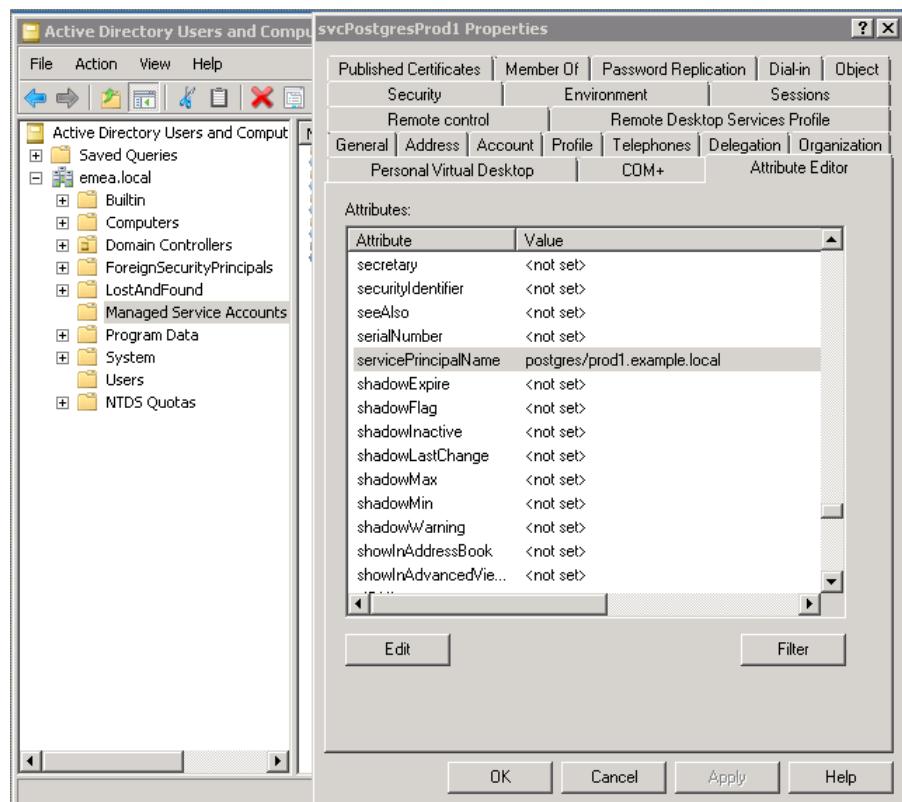
在此示例中，AD的密码被设置为永不过期并且用户不能修改。只有在创建Kerberos密钥表文件时才需要AD账户的密码。不需要把它提供给数据库管理员。



AD管理员必须用Windows的setspn命令行，把服务主体名称属性添加到到账户中。该命令示例为AD用户svcPostgresProd1添加SPN属性postgres/prod1.example.local：

```
setspn -A postgres/prod1.example.local svcPostgresProd1
```

如果设置了高级特性，可以在活动目录用户和计算机视图中，看到该SPN属性。如果有必要，可以在属性编辑器标签页找到并编辑servicePrincipalName。



下一步是创建Kerberos密钥表文件。

如果有安全性需要，可以选择特定的加密方法，如果没有，最好先使其工作，然后删除您不需要的任何加密方法。

作为AD域管理员，您可以使用ktppass命令列出AD域控制器支持的加密类型：

```
ktppass /?
```

作为AD域管理员，可以运行ktppass命令创建一个密钥表文件。该示例命令创建包含如下信息的文件svcPostgresProd1.keytab：

- 服务主体名称 (SPN)：

```
postgres/prod1.example.local@EXAMPLE.LOCAL
```

- AD用户： svcPostgresProd1

- 加密方法： ALL available on AD

- 主体类型： KRB5\_NT\_PRINCIPAL

```
ktppass -out svcPostgresProd1.keytab -princ
postgres/prod1.example.local@EXAMPLE.LOCAL -mapUser
svcPostgresProd1
-pass your_password -crypto all -ptype KRB5_NT_PRINCIPAL
```

Note: AD域名@EXAMPLE.LOCAL被附加到SPN。

拷贝密钥表文件svcPostgresProd1.keytab到Greenplum数据

库master主机。

作为AD域管理员运行ktcpass的替代方案，如果桌面具有Java环境可以运行Java的ktab.exe工具生成一个密钥表文件。不管使用ktcpass或者ktab.exe，密码作为命令行参数，在屏幕上是可见的。

该命令示例创建一个密钥表文件 svcPostgresProd1.keytab。

```
"c:\Program Files\Java\jre1.8.0_77\bin\ktab.exe" -a
svcPostgresprod1 -k svcPostgresProd1.keytab
Password for svcPostgresprod1@EXAMPLE.LOCAL:your_password
Done!
Service key for svcPostgresprod1 is saved in
svcPostgresProd1.keytab
```

Note: 如果使用AES256-CTS-HMAC-SHA1-96加密，需要从Oracle下载并安装Java扩展*Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files for JDK/JRE*。

## 为活动目录设置Greenplum数据库

如下说明假定Kerberos工作站工具krb5-workstation已经安装在Greenplum数据库master主机上。

使用AD域名详细信息和AD域控制器的位置，更新/etc krb5.conf。如下是一个示例配置。

```
[logging]
default = FILE:/var/log/krb5libs.log
kdc = FILE:/var/log/krb5kdc.log
admin_server = FILE:/var/log/kadmind.log

[libdefaults]
default_realm = EXAMPLE.LOCAL
dns_lookup_realm = false
dns_lookup_kdc = false
ticket_lifetime = 24h
renew_lifetime = 7d
forwardable = true

[realms]
EXAMPLE.LOCAL = {
kdc = bocdc.example.local
admin_server = bocdc.example.local
}

[domain_realm]
.example.local = EXAMPLE.LOCAL
```

```
example.com = EXAMPLE.LOCAL
```

拷贝包含AD用户信息的Kerberos密钥表文件到Greenplum数据库master目录。该示例中复制了在[设置活动目录](#)中创建的svcPostgresProd1.keytab。

```
mv svcPostgresProd1.keytab $MASTER_DATA_DIRECTORY
chown gpadmin:gpadmin
$MASTER_DATA_DIRECTORY/svcPostgresProd1.keytab
chmod 600 $MASTER_DATA_DIRECTORY/svcPostgresProd1.keytab
```

在Greenplum数据库pg\_hba.conf最后一行添加如下配置。该内容配置Greenplum数据库使用活动目录授权之前行不匹配的连接请求。

```
host all all 0.0.0.0/0 gss include_realm=0
```

更新Greenplum数据库postgresql.conf文件中密钥表文件的位置详细信息要使用的主体名称。完全限定名称和/etc/krb5.conf中的域名称，构成了完整的服务主体名称。

```
krb_server_keyfile = '/data/master/gpseg-1/svcPostgresProd1.keytab'
krb_srvname = 'postgres'
```

为AD用户创建数据库角色。该示例登录到默认数据库并运行CREATE ROLE命令。用户dev1是[设置活动目录](#)创建密钥表文件的指定用户。

```
psql
create role dev1 with login superuser;
```

重启数据库以使用最新的身份认证信息：

```
gpstop -a
gpstart
```

Note: Greenplum数据库的依赖库可能与Kerberos工作站工具如kinit冲突。在Greenplum数据库master主机，可以不加载\$GPHOME/greenplum\_path.sh脚本使用gpadmin命令行，或者取消设置LD\_LIBRARY\_PATH类似于如下命令：

```
unset LD_LIBRARY_PATH
kinit
source $GPHOME/greenplum_path.sh
```

## 确认Greenplum数据库可以通过Kerberos认证访问

```
kinit dev1
psql -h prod1.example.local -U dev1
```

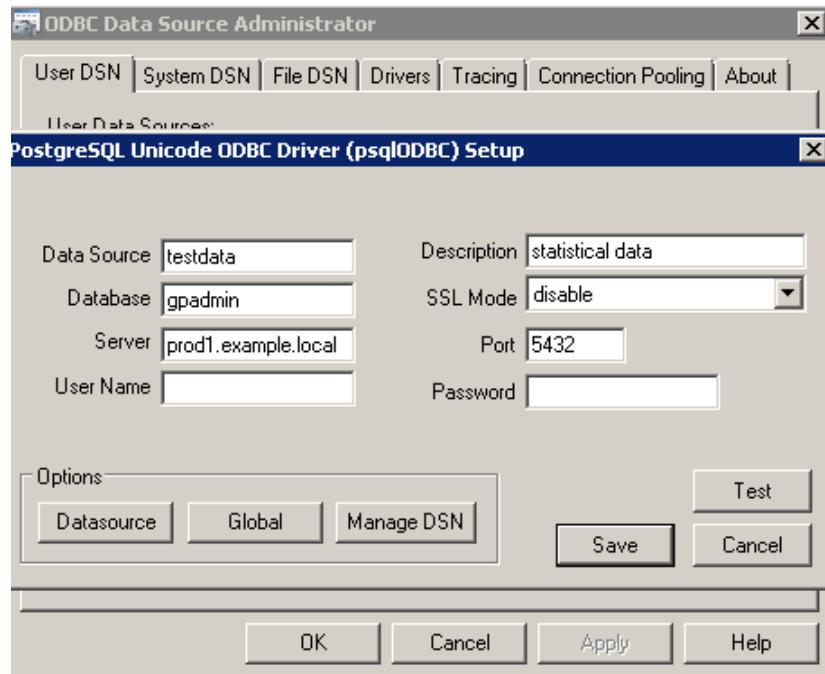
## 单点登录示例

如下使用AD和Kerberos的单点登录实例，假设使用单点登录AD用户dev1登入到Windows桌面。

该示例为Greenplum数据库配置Aginity Workbench。在使用单点登录时，启用了“使用集成的安全性”。



该示例配置一个ODBC源。设置该ODBC源时，请不要输入用户名或密码。该DSN可用作应用程序的ODBC数据源。



可以用R客户端使用DSN为`testdata`。该示例配置R以访问DSN。

```
library("RODBC")
conn <- odbcDriverConnect("testdata")
sql <- "select * from public.data1"
my_data <- sqlQuery(conn,sql)
print(my_data)
```

## 活动目录的问题及可能的解决方案

- Kerberos票据含有的版本号必须匹配AD的版本号。  
要显示密钥表文件中的版本号，使用`klist -ket`命令，例如：

```
klist -ket svcPostgresProd1.keytab
```

要从AD域控制器得到相应的值，可作为一个AD管理员运行这个命令：

```
kvno postgres/prod1.example.local@EXAMPLE.LOCAL
```

- 当Windows ID和Greenplum数据库用户角色ID不匹配时，将会出现如下登录错误。该日志文件项显示了登录错误。用户`dev22`尝试从一个Windows桌面登录，而桌面上登录了一个不同的Windows用户。

```
2016-03-29 14:30:54.041151
PDT, "dev22", "gpadmin", p15370, th2040321824,
"172.28.9.181", "49283", 2016-03-29 14:30:53
```

```
PDT,1917,con32,,seg-1,,,x1917,sx1,
 "FATAL", "28000", "authentication failed for user
 ""dev22"": valid
 until timestamp expired",,,,,,,0,, "auth.c", 628,
```

当用户能被认证但没有对应Greenplum数据库用户角色时，也会发生这个错误。

确保该用户是用的是正确的Windows ID并且为该用户ID配置了Greenplum数据库用户角色。

- 当Kerberos密钥表不含有与正在尝试连接的客户端相匹配的加密类型时，会发生下面这种错误。

```
psql -h 'hostname' postgres
psql: GSSAPI continuation error: Unspecified GSS
failure. Minor code may provide more information
GSSAPI continuation error: Key version is not available
```

解决方案是使用ktutil增加额外的加密类型到该密钥表，或者用来自AD中所有加密系统重新创建postgres的密钥表。

## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

关于Greenplum的查询  
处理

## □ 关于GPORCA

## GPORCA概述

## 启用和禁用GPORCA

## 收集根分区统计信息

使用GPORCA时的考  
虑

## GPORCA特性和增强

## GPORCA改变的行为

## GPORCA的限制

判断被使用的查询优  
化器

## 关于统一多级分区表

## 定义查询

如果一个多级分区 (MLP) 表是一个统一分区表, GPORCA支持在该MLP上的查询。多级分区表是用SUBPARTITION子句创建的分区表。统一分区表必须满足下面这些条件。

- 分区表结构统一。同一层上的每一个分区节点必须具有相同的层次结构。
- 分区键约束必须一致且统一。在每一个子分区层次上, 为每个分支创建的子表上的约束集必须匹配。

可以用几种方式显示分区表的信息, 包括显示来自这些来源的信息:

- pg\_partitions系统视图包含分区表结构的信息。
- pg\_constraint系统catalog包含表约束的信息。
- psql元命令\d+ *tablename*显示一个分区表的叶子子表的表约束。

**Parent topic:** [关于GPORCA](#)

## 示例

这个CREATE TABLE命令创建一个统一分区表。

```
CREATE TABLE mlp (id int, year int, month int, day int,
 region text)
DISTRIBUTED BY (id)
PARTITION BY RANGE (year)
 SUBPARTITION BY LIST (region)
 SUBPARTITION TEMPLATE (
 SUBPARTITION usa VALUES ('usa'),
 SUBPARTITION europe VALUES ('europe'),
 SUBPARTITION asia VALUES ('asia'))
 (START (2006) END (2016) EVERY (5));
```

下面这些是为表mlp创建的子表和分区层次。这个层次由一个包含两个分支的子分区构成。

```
mlp_1_prt_11
 mlp_1_prt_11_2_prt_usa
 mlp_1_prt_11_2_prt_europe
 mlp_1_prt_11_2_prt_asia
```

```
mlp_1_prt_21
 mlp_1_prt_21_2_prt_usa
 mlp_1_prt_21_2_prt_europe
 mlp_1_prt_21_2_prt_asia
```



该表的层次是统一的，每个分区包含三个子表（子分区）的集合。区域子分区的约束也是统一的，分支表mlp\_1\_prt\_11的子表上的约束集和分支表mlp\_1\_prt\_21的子表的约束相同。

作为一种快速检查，这个查询显示这些分区的约束。

```
WITH tbl AS (SELECT oid, partitionlevel AS level,
 partitiontablename AS part
 FROM pg_partitions, pg_class
 WHERE tablename = 'mlp' AND
partitiontablename=relname
 AND partitionlevel=1)
 SELECT tbl.part, consrc
 FROM tbl, pg_constraint
 WHERE tbl.oid = conrelid ORDER BY consrc;
```

Note: 注意：您需要修改查询以获得更复杂的分区表。例如，查询不考虑不同schema中的表名。

consr列显示子分区上的约束。mlp\_1\_prt\_1中子分区的区域约束集合匹配mlp\_1\_prt\_2中子分区的约束。year的约束继承自父分支表。

| part                     | consr                              |
|--------------------------|------------------------------------|
| mlp_1_prt_2_2_prt_asia   | (region = 'asia'::text)            |
| mlp_1_prt_1_2_prt_asia   | (region = 'asia'::text)            |
| mlp_1_prt_2_2_prt_europe | (region = 'europe'::text)          |
| mlp_1_prt_1_2_prt_europe | (region = 'europe'::text)          |
| mlp_1_prt_1_2_prt_usa    | (region = 'usa'::text)             |
| mlp_1_prt_2_2_prt_usa    | (region = 'usa'::text)             |
| mlp_1_prt_1_2_prt_asia   | ((year >= 2006) AND (year < 2011)) |
| mlp_1_prt_1_2_prt_usa    | ((year >= 2006) AND (year < 2011)) |
| mlp_1_prt_1_2_prt_europe | ((year >= 2006) AND (year < 2011)) |
| mlp_1_prt_2_2_prt_usa    | ((year >= 2011) AND (year < 2016)) |
| mlp_1_prt_2_2_prt_asia   | ((year >= 2011) AND (year < 2016)) |
| mlp_1_prt_2_2_prt_europe | ((year >= 2011) AND (year < 2016)) |
| (12 rows)                |                                    |

如果用下面的命令对例子中的分区表增加一个默认分区：

```
ALTER TABLE mlp ADD DEFAULT PARTITION def
```

该分区表仍然会是一个统一分区表。为默认分区创建的分支包含三个

子表并且子表上的约束集匹配现有的子表约束集。

在上面的示例中，如果删除子分区mlp\_1\_prt\_21\_2\_prt\_asia并为区域canada添加另一个子分区，则约束不再一致。

```
ALTER TABLE mlp ALTER PARTITION FOR (RANK(2))
DROP PARTITION asia ;

ALTER TABLE mlp ALTER PARTITION FOR (RANK(2))
ADD PARTITION canada VALUES ('canada');
```

但是，如果将子分区canada添加到原始分区表的mlp\_1\_prt\_21和mlp\_1\_prt\_11，它仍然是统一的分区表。

Note: 只有分区级别的分区集上的约束必须相同。分区的名称可以不同。

## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

关于Greenplum的查询  
处理

## □ 关于GPORCA

## GPORCA概述

## 启用和禁用GPORCA

## 收集根分区统计信息

使用GPORCA时的考  
虑

## GPORCA特性和增强

## GPORCA改变的行为

## GPORCA的限制

判断被使用的查询优  
化器

## 关于统一多级分区表

## 定义查询

默认情况下, Greenplum数据库使用GPORCA来替代传统查询规划器。服务器配置参数可以启用或者禁用GPORCA。

虽然GPORCA默认处于启用状态, 但您可以使用optimizer参数在系统, 数据库, 会话或查询级别配置GPORCA使用情况。如果要更改默认行为, 请参阅以下部分之一:

- [为一个系统启用GPORCA](#)
- [为一个数据库启用GPORCA](#)
- [为一个会话或者查询启用GPORCA](#)

Note: 可以使用服务器配置参数optimizer\_control禁用启用或者禁用GPORCA的能力。有关服务器配置参数的信息请见Greenplum数据库参考指南。

**Parent topic:** [关于GPORCA](#)

## 为一个系统启用GPORCA

为Greenplum数据库系统设置服务器配置参数optimizer。

1. 作为gpadmin (Greenplum数据库管理员) 登入到Greenplum数据库的Master主机。
2. 设置服务器配置参数的值。下面这些Greenplum数据库gpconfig工具命令把这些参数的值设置为on:

```
$ gpconfig -c optimizer -v on --masteronly
```

3. 重启Greenplum数据库。下面这个Greenplum数据库gpstop工具命令重新载入Master和Segment的postgresql.conf文件而不关闭Greenplum数据库。

```
gpstop -u
```

## 为一个数据库启用GPORCA

用ALTER DATABASE命令为单个Greenplum数据库设置服务器配置参数optimizer。例如, 这个命令为数据库test\_db启用GPORCA。

```
> ALTER DATABASE test_db SET OPTIMIZER = ON ;
```

## 为一个会话或者查询启用GPORCA

可以使用SET命令为一个会话设置optimizer服务器配置参数。例如，在使用psql工具连接到Greenplum数据库之后，这个SET命令启用GPORCA：

```
> set optimizer = on ;
```

要为特定查询设置参数，请在运行查询之前包含SET命令。

## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

关于Greenplum的查询  
处理

## □ 关于GPORCA

## GPORCA概述

## 启用和禁用GPORCA

## 收集根分区统计信息

使用GPORCA时的考  
虑

## GPORCA特性和增强

## GPORCA改变的行为

## GPORCA的限制

判断被使用的查询优  
化器

## 关于统一多级分区表

## 定义查询

对于分区表，GPORCA使用表根分区的统计信息来生成查询计划。这些统计信息用于确定联接顺序、拆分和联接聚合节点以及计算查询步骤的成本。相比之下，Postgres规划器使用每个叶分区的统计信息。

如果在分区表上执行查询，应收集根分区的统计信息，并定期更新这些统计信息，以确保gporca能够生成最佳查询计划。如果根分区统计信息不是最新的或不存在，gporca仍然对表的查询执行动态分区消除。但是，查询计划可能不是最佳的。

**Parent topic:** [关于GPORCA](#)

## 运行ANALYZE

默认情况下，在分区表的根分区上运行ANALYZE命令将对表中的叶分区数据进行采样，并存储根分区的统计信息。ANALYZE 收集根分区和叶分区的统计信息，包括HyperLogLog (HLL)统计信息。ANALYZE ROOTPARTITION 只收集根分区上的统计信息。服务器配置参数[optimizer\\_analyze\\_root\\_partition](#)控制是否需要ROOTPARTITION关键字来收集分区表根分区的根统计信息。有关收集分区表统计信息的信息，请参阅[ANALYZE](#)命令。 .

记住，在更新根分区统计信息之前，ANALYZE总是扫描整个表。如果您的表非常大，则此操作可能需要大量时间。ANALYZE ROOTPARTITION还使用一个访问ACCESS SHARE锁，用于在执行期间阻止某些操作，如TRUNCATE和VACUUM操作。出于这些原因，您应该定期或者当叶分区数据发生重大更改时进行ANALYZE操作。

按照以下最佳实践在系统中分区表上运行ANALYZE或ANALYZE ROOTPARTITION：

- 运行ANALYZE <root\_partition> 在一个新的分区表中添加初始数据后。运行ANALYZE <leaf\_partition> 在新的叶分区或数据已更改的叶分区上。默认情况下，如果其他叶分区具有统计信息，则在叶分区上运行命令将更新根分区统计信息。
- 当您在表的EXPLAIN查询计划中发现查询性能衰退，或者在叶分区数据发生重大更改之后，更新根分区统计信息。例如，如果在生成根分区统计信息后的某个时刻添加新的叶子分区，请考虑运行ANALYZE或ANALYZE ROOTPARTITION以使用从子分区插入的新元组更新根分区统计信息。
- 对于非常大的表，只需每周运行ANALYZE或ANALYZE

ROOTPARTITION，或者以长于每天的更新时间间隔运行。

- 避免在没有参数的情况下运行ANALYZE，因为这样做会对所有数据库表（包括分区表）执行命令。对于大型数据库，这些全局ANALYZE操作很难监控，并且很难预测完成所需的时间。
- 如果您的I/O吞吐量能够支持负载，可以考虑并行运行多个ANALYZE <table\_name>或ANALYZE ROOTPARTITION <table\_name>操作来加速统计数据收集的操作。
- 还可以使用Greenplum数据库实用程序analyzedb更新表统计信息。使用analyzedb确保如果没有对叶分区进行任何修改，则不会重新分析以前分析过的表。

## GPORCA和叶分区统计信息

尽管创建和维护根分区统计信息对于分区表的GPORCA查询性能至关重要，但是维护叶分区统计信息也很重要。如果GPORCA无法针对分区表生成查询计划，则使用传统查询优化器，需要叶分区统计信息来为该查询生成最佳计划。

GPORCA本身也将叶分区统计信息用于直接访问叶分区的任何查询，而不是使用带有谓词的根分区来消除分区。例如，如果知道哪些分区包含查询所需的元组，则可以直接查询叶分区表本身；在这种情况下，GPORCA使用叶分区统计信息。

## 禁用自动根分区统计信息收集

如果不打算使用GPORCA对分区表执行查询（设置服务器配置参数`optimizer`为`off`），然后你可以禁用自动收集分区表的根分区的统计信息。服务器配置参数`optimizer_analyze_root_partition`控制是否需要ROOTPARTITION关键字来收集分区表根分区的根统计信息。参数的默认设置为`on`，ANALYZE命令可以在不使用ROOTPARTITION关键字的情况下收集根分区统计信息。通过将参数设置为`off`，可以禁用根分区统计信息的自动收集。当该值为`off`时，必须运行ANALZYE ROOTPARTITION来收集根分区统计信息。

- 以数据库管理员，比如gpadmin的身份登录Greenplum数据库master主机。
- 设置服务器配置参数的值。这些Greenplum数据库gpconfig实用程序命令将参数值设置为`off`：

```
$ gpconfig -c optimizer_analyze_root_partition -v off --
masteronly
```

3. 重启Greenplum数据库。这个实用程序命令gpstop在不关闭Greenplum数据库的情况下重新加载master和segments的postgresql.conf文件。

## Greenplum数据库® 6.0文档

## □ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

关于Greenplum的查询  
处理

□ 关于GPORCA

GPORCA概述

启用和禁用GPORCA

收集根分区统计信息

使用GPORCA时的考  
虑

GPORCA特性和增强

GPORCA改变的行为

GPORCA的限制

判断被使用的查询优  
化器

关于统一多级分区表

定义查询

用GPORCA最优化执行查询需要考虑的查询条件。

确保满足下列条件：

- 表不含有多列分区键。
- 多级分区表是一个统一多级分区表。请见[关于统一多级分区表](#)。
- 在针对只存在于Master的表（例如系统表`pg_attribute`）运行时，服务器配置参数`optimizer_enable_master_only_queries`被设置为on。有关该参数的信息，请见[Greenplum数据库参考指南](#)。  
Note: 启用这一参数会降低catalog短查询的性能。为了避免这一问题，只对会话或者查询设置这一参数。
- 已经在分区表的根分区上收集了统计信息。

如果分区表包含超过20,000个分区，考虑重新设计该表的模式。

这些服务器配置参数影响GPORCA查询处理。

- `optimizer_cte_inlining_bound` 控制对公共表表达式（CTE）查询（含有WHERE子句的查询）执行的内联量。
- `optimizer_force_multistage_agg` 强制GPORCA为标量区分限制聚集选择一种3阶段聚集计划。
- `optimizer_force_three_stage_scalar_dqa` 强制GPORCA在生成了带有多阶段聚集的计划时选择它。
- `optimizer_join_order` 为连接排序设置查询优化级别，通过指定要评估哪些类型的连接排序选项。
- `optimizer_join_order_threshold` 指定GPORCA使用基于动态编程的连接排序算法的最大连接子数。
- `optimizer_nestloop_factor` 控制查询优化时应用到嵌套循环连接的代价因子。
- `optimizer_parallel_union` 控制对于含有UNION或者UNION ALL子句的查询发生的并行量。当该值为on时，GPORCA可以生成一个查询计划，其中UNION或者UNION ALL操作的子操作在Segment实例上并行执行。
- `optimizer_sort_factor` 控制GPORCA在查询优化时应用于排序操作的代价因子。当出现数据倾斜时可以为查询调整代价因子。
- `gp_enable_relsize_collection` 控制GPORCA（和传统查询优化器）处理一个没有统计信息的表的方式。默认情况下，如果统计信息不可用，GPORCA使用默认值估计行数。当该参数为on，GPORCA使用表的估计大小（如果没有统计数据）。  
对于分区表的根分区，此参数将被忽略。如果根分区没有统计信

息，GPORCA总是使用默认值。你可以使用ANALYZE ROOTPARTITION收集根分区的统计信息。请见[ANALYZE](#)。

这些服务器配置参数控制信息的显示和记录。

- optimizer\_print\_missing\_stats控制有关对查询缺失统计信息的列的信息显示（默认是true）
- optimizer\_print\_optimization\_stats控制GPORCA查询优化度量对于查询的记录（默认为off）

有关这些参数的信息请见*Greenplum*数据库参考指南。

GPORCA生成minidump来描述一个给定查询的优化上下文。该文件中的信息不太容易被用于调试或者排错。minidump文件位于Master的数据目录中并且使用下面的命名格式：

Minidump\_date\_time.mdp

有关minidump文件的信息，请见*Greenplum*数据库参考指南中的服务器配置参数 optimizer\_minidump。

当EXPLAIN ANALYZE命令使用GPORCA时，EXPLAIN计划只显示被消除的分区数。被扫描的分区不被显示。要在Segment日志中显示被扫描分区的名字，可设置服务器配置参数gp\_log\_dynamic\_partition\_pruning为on。下面这个SET命令的例子启用该参数。

```
SET gp_log_dynamic_partition_pruning = on;
```

**Parent topic:** [关于GPORCA](#)

## Greenplum数据库® 6.0文档

## □ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

关于Greenplum的查询  
处理

□ 关于GPORCA

GPORCA概述

启用和禁用GPORCA

收集根分区统计信息

使用GPORCA时的考  
虑

GPORCA特性和增强

GPORCA改变的行为

GPORCA的限制

判断被使用的查询优  
化器

关于统一多级分区表

定义查询

GPORCA是Greenplum的下一代查询优化器，它包括了对特定类型的查询和操作的增强：

- [对分区表的查询](#)
- [含有子查询的查询](#)
- [含有公共表表达式的查询](#)
- [GPORCA的DML操作增强](#)

GPORCA还包括下面这些优化增强：

- 改进的连接排序
- 连接-聚集重排序
- 排序顺序优化
- 查询优化中包括的数据倾斜估计

**Parent topic:** [关于GPORCA](#)

## 对分区表的查询

GPORCA包括这些对分区表上查询的增强：

- 改进了分区消除。
- 支持统一多级分区表。有关统一多级分区表的信息，请见[关于统一多级分区表](#)
- 查询计划可以包含分区选择器操作符。
- 不在EXPLAIN计划中枚举分区。

对于涉及静态分区选择的查询，在其中会将分区键与常量进行比较，GPORCA会在EXPLAIN输出的分区选择器操作符下面列出要被扫描的分区数。这个示例分区选择器操作符展示了过滤条件和选中的分区数：

```
Partition Selector for Part_Table (dynamic scan id: 1)
 Filter: a > 10
 Partitions selected: 1 (out of 3)
```

对于涉及动态分区选择的查询，在其中会将分区键与变量进行比较，要扫描的分区数只有在查询执行时才能知道。选中的分区不会显示在EXPLAIN输出中。

- 计划尺寸与分区数无关。
- 由于分区数导致的内存不足错误被减少。



这个CREATE TABLE命令的例子创建一个范围分区表。

```
CREATE TABLE sales(order_id int, item_id int, amount
numeric(15,2),
date date, yr_qtr int)
range partitioned by yr_qtr;
```

GPORCA改进了针对分区表的这些类型的查询：

- 全表扫描。计划中不会枚举分区。

```
SELECT * FROM sales;
```

- 带一个常量过滤谓词的查询。会执行分区消除。

```
SELECT * FROM sales WHERE yr_qtr = 201501;
```

- 范围选择。会执行分区消除。

```
SELECT * FROM sales WHERE yr_qtr BETWEEN 201601 AND
201704 ;
```

- 涉及到分区表的连接。在下面这个例子中，分区过的维度表`date_dim`被连接到事实表`catalog_sales`：

```
SELECT * FROM catalog_sales
WHERE date_id IN (SELECT id FROM date_dim WHERE
month=12);
```

## 含有子查询的查询

GPORCA更有效地处理子查询。子查询是嵌套在外层查询块里面的查询。在下面的查询中，WHERE子句中的SELECT是一个子查询。

```
SELECT * FROM part
WHERE price > (SELECT avg(price) FROM part);
```

GPORCA还能更有效地处理含有相关子查询（CSQ）的查询。相关子查询是使用来自外层查询的值的子查询。在下面的查询中，`price`列被使用在外层查询和子查询中。

```
SELECT * FROM part p1
WHERE price > (SELECT avg(price) FROM part p2
WHERE p2.brand = p1.brand);
```

GPORCA为下列类型的子查询生成更有效的计划：

- SELECT 列表中的CSQ。

```
SELECT *,
 (SELECT min(price) FROM part p2 WHERE p1.brand =
 p2.brand)
 AS foo
 FROM part p1;
```

- 析取 (OR) 过滤条件中的CSQ。

```
SELECT FROM part p1 WHERE p_size > 40 OR
 p_retailprice >
 (SELECT avg(p_retailprice)
 FROM part p2
 WHERE p2.p_brand = p1.p_brand)
```

- 具有越级关联的嵌套CSQ

```
SELECT * FROM part p1 WHERE p1.p_partkey
IN (SELECT p_partkey FROM part p2 WHERE p2.p_retailprice
=
 (SELECT min(p_retailprice)
 FROM part p3
 WHERE p3.p_brand = p1.p_brand)
);
```

Note: 传统查询优化器不支持具有越级关联的嵌套CSQ。

- 有聚集和不等于的CSQ。下面这个例子包含一个有不等于的CSQ。

```
SELECT * FROM part p1 WHERE p1.p_retailprice =
 (SELECT min(p_retailprice) FROM part p2 WHERE
 p2.p_brand <> p1.p_brand);
```

- 必须返回一行的CSQ。

```
SELECT p_partkey,
 (SELECT p_retailprice FROM part p2 WHERE p2.p_brand =
 p1.p_brand)
 FROM part p1;
```

## 含有公共表表达式的查询

GPORCA处理含有WITH子句的查询。WITH子句也被称为公共表表达式。

式 (CTE) , 它会生成只在该查询中存在的临时表。下面这个查询例子包含一个CTE。

```
WITH v AS (SELECT a, sum(b) as s FROM t WHERE c < 10 GROUP BY a)
SELECT * FROM v AS v1, v AS v2
WHERE v1.a <> v2.a AND v1.s < v2.s;
```

作为查询优化的一部分, GPORCA可以把谓词下推到一个CTE中。对于下面的例子查询, GPORCA把等于谓词推到了CTE。

```
WITH v AS (SELECT a, sum(b) as s FROM t GROUP BY a)
SELECT *
FROM v AS v1, v AS v2, v AS v3
WHERE v1.a < v2.a
AND v1.s < v3.s
AND v1.a = 10
AND v2.a = 20
AND v3.a = 30;
```

GPORCA可以处理这些类型的CTE:

- 定义一个或者多个表的CTE。在下面的这个查询中, CTE定义两个表。

```
WITH cte1 AS (SELECT a, sum(b) as s FROM t
 WHERE c < 10 GROUP BY a),
 cte2 AS (SELECT a, s FROM cte1 WHERE s > 1000)
SELECT *
FROM cte1 AS v1, cte2 AS v2, cte2 AS v3
WHERE v1.a < v2.a AND v1.s < v3.s;
```

- 嵌套CTE。

```
WITH v AS (WITH w AS (SELECT a, b FROM foo
 WHERE b < 5)
 SELECT w1.a, w2.b
 FROM w AS w1, w AS w2
 WHERE w1.a = w2.a AND w1.a > 2)
SELECT v1.a, v2.a, v2.b
FROM v AS v1, v AS v2
WHERE v1.a < v2.a;
```

## GPORCA的DML操作增强

GPORCA含有对DML操作 (例如INSERT, UPDATE和DELETE) 的增强。

- 查询计划中的DML节点是一个查询计划操作符。

- 可以作为一个常规节点（目前只有顶层切片）出现在计划中的任何地方
- 可以有消费者
- UPDATE操作使用查询计划操作符Split并且支持这些操作：
  - 表的分布键列上的UPDATE操作。
  - 表的分区键列上的UPDATE操作。

这个计划的例子展示了Split操作符。

```
QUERY PLAN

Update (cost=0.00..5.46 rows=1 width=1)
 -> Redistribute Motion 2:2 (slice1; segments: 2)
 Hash Key: a
 -> Result (cost=0.00..3.23 rows=1 width=48)
 -> Split (cost=0.00..2.13 rows=1
width=40)
 -> Result (cost=0.00..1.05 rows=1
width=40)
 -> Seq Scan on dmltest
```

- 新的查询计划操作符Assert 被用于约束检查。
- 这个计划的例子展示了Assert 操作符。

```
QUERY PLAN

Insert (cost=0.00..4.61 rows=3 width=8)
 -> Assert (cost=0.00..3.37 rows=3 width=24)
 Assert Cond: (dmlsource.a > 2) IS DISTINCT FROM
false
 -> Assert (cost=0.00..2.25 rows=3 width=24)
 Assert Cond: NOT dmlsource.b IS NULL
 -> Result (cost=0.00..1.14 rows=3
width=24)
 -> Seq Scan on dmlsource
```

Greenplum数据库® 6.0文档

- 管理员指南

- Greenplum数据库概念

- 管理一个Greenplum系统

- 管理Greenplum数据库访问

- 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

- 查询数据

关于Greenplum的查询  
处理

- 关于GPORCA

GPORCA概述

启用和禁用GPORCA

收集根分区统计信息

使用GPORCA时的考  
虑

GPORCA特性和增强

## GPORCA改变的行为

GPORCA的限制

判断被使用的查询优  
化器

关于统一多级分区表

定义查询

相比使用传统规划器，启用了GPORCA优化器（默认启用）的Greenplum数据库的行为有些改变。

- 允许在分布键上的UPDATE操作。
- 允许在分区键上的UPDATE操作。
- 支持对统一分区表的查询。
- 对更改为使用外部表作为叶子分区的分区表的查询将回退到Postgres查询优化器。
- 不支持直接在分区表的分区（子表）上的DML操作，INSERT除外。  
对于INSERT命令，可以指定分区表的一个叶子子表来把数据插入到分区表中。如果该数据对于指定的叶子子表不合法，则会返回一个错误。不支持指定一个不是叶子的子表。
- 如果命令CREATE TABLE AS 中没有指定DISTRIBUTED BY子句且没有指定主键或者唯一键，它将会随机分布表数据。
- 不允许不确定的更新。下面的UPDATE命令会返回一个错误。

```
update r set b = r.b + 1 from s where r.a in (select a from s);
```

- 分区表的根表上需要统计信息。ANALYZE命令会在根表和个别分区表（叶子子表）上生成统计信息。详见ANALYZE命令的ROOTPARTITION子句。
- 查询计划中的附加Result节点：
  - 查询计划的Assert 操作符。
  - 查询计划的Partition selector 操作符。
  - 查询计划的Split操作符。
- 在运行EXPLAIN时，GPORCA生成的查询计划与传统查询优化器生成的计划不同。
- 当启用了GPORCA且Greenplum数据库回退到传统查询优化器生成查询计划时，Greenplum数据库会增加日志文件消息Planner produced plan。
- 当一个或者更多个表列的统计信息缺失时，Greenplum数据库会发出一个警告。在用GPORCA执行SQL命令时，如果该命令的性能可以通过在它引用的列或者列组上收集统计信息而提升，Greenplum数据库会发出警告。该警告在命令行并且信息会被加入到Greenplum数据库的日志文件。有关在表列上收集统计信息的内容，请见Greenplum数据库参考指南中的ANALYZE命令。

**Parent topic:** [关于GPORCA](#)

Greenplum数据库® 6.0文档

- 管理员指南

- Greenplum数据库概念

- 管理一个Greenplum系统

- 管理Greenplum数据库访问

- 定义数据库对象

- 分布与倾斜

- 插入, 更新, 和删除数据

- 查询数据

- 关于Greenplum的查询  
处理

- 关于GPORCA

- GPORCA概述

- 启用和禁用GPORCA

- 收集根分区统计信息

- 使用GPORCA时的考  
虑

- GPORCA特性和增强

- GPORCA改变的行为

- GPORCA的限制**

- 判断被使用的查询优  
化器

- 关于统一多级分区表

- 定义查询

在Greenplum数据库中使用默认的GPORCA优化器时有一些限制。GPORCA和传统的查询优化器当前并存于Greenplum数据库中，因为GPORCA不支持所有的Greenplum数据库特性。

这一节描述这些限制。

- [不支持的SQL查询特性](#)
- [性能衰退](#)

**Parent topic:** [关于GPORCA](#)

## 不支持的SQL查询特性

下面这些是GPORCA被启用时（默认启用）不支持的特性：

- 索引表达式（在基于表的一个或者更多列的表达式上定义的索引）
- GIN索引方法。GPORCA只支持B树、位图和Gist索引。GPORCA忽略使用不支持的方法创建的索引。
- 外部参数
- 这些类型的分区表：
  - 非统一分区表。
  - 被修改为用一个外部表作为叶子子分区的分区表。
- SortMergeJoin (SMJ).
- 有序聚集
- 这些分析扩展：
  - CUBE
  - 多分组集
- 这些标量操作符：
  - ROW
  - ROWCOMPARE
  - FIELDSELECT
- 将集合运算符作为输入参数的聚合函数。
- percentile\_\* 窗口函数（Greenplum数据库不支持有序集聚合函  
数）。
- 逆分布函数
- 执行用ON MASTER或ON ALL SEGMENTS属性定义的函数的查



询。

- 在元数据名称（如表名）中包含Unicode字符的查询，以及字符与主机系统区域设置不兼容。
- 表名由关键字ONLY指定的SELECT, UPDATE和DELETE命令。
- 按列排序规则。只有当查询中的所有列都使用时，GPORCA才支持排序规则相同的排序规则。如果查询中的列使用不同的排序规则，Greenplum则使用传统查询计划器。

## 性能衰退

启用GPORCA时，已知下列特性会发生性能衰退：

- 短查询 - 对于GPORCA，短查询可能会因为GPORCA对于判断最优查询执行计划的增强而遇到额外的负担。
- ANALYZE- 对于GPORCA，ANALYZE命令为分区表生成根分区的统计信息。对于传统优化器，这些统计信息不会被生成。
- DML操作 - 对于GPORCA，DML增强包括在分区和分布键上的更新支持，这可能会产生额外的负担。

此外，GPORCA使用这些特性执行SQL语句时，以前版本特性的增强功能可能会导致额外的时间。

## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

## □ 查询数据

关于Greenplum的查询  
处理

## □ 关于GPORCA

GPORCA概述

启用和禁用GPORCA

收集根分区统计信息

使用GPORCA时的考  
虑

GPORCA特性和增强

GPORCA改变的行为

GPORCA的限制

判断被使用的查询优  
化器

关于统一多级分区表

定义查询

当GPORCA被启用（默认启用）时，可以判断Greenplum数据库是在使用GPORCA还是退回到传统查询优化器。

可以检查查询的EXPLAIN查询计划来判断Greenplum数据库使用哪一种查询优化器来执行该查询：

- 当GPORCA生成该查询计划时，配置optimizer=on和GPORCA的版本会被显示在查询计划的末尾。例如：

```
Settings: optimizer=on
Optimizer status:
Pivotal Optimizer (GPORCA) version 1.584
```

当Greenplum数据库退回到使用传统优化器生成该计划时，设置optimizer=on和Postgres query optimizer会被显示在查询计划的末尾。例如：

```
Settings: optimizer=on
Optimizer status:
Postgres query optimizer
```

当服务器配置参数OPTIMIZER是off时，下面这些行会被显示在查询计划的末尾。

```
Settings: optimizer=off
Optimizer status: Postgres query optimizer
```

- 下面这些计划项只出现在GPORCA生成的EXPLAIN计划输出中。传统优化器查询计划中不支持这些项。
  - Assert operator
  - Sequence operator
  - DynamicIndexScan
  - DynamicSeqScan
- 当一个针对分区表的查询由GPORCA生成时，EXPLAIN计划只显示正在被消除的分区数。被扫描的分区不会被显示出来。由传统优化器生成的EXPLAIN计划会列出被扫描的分区。

日志文件包含的消息会指示使用了哪一种查询优化器。如果Greenplum数据库退回到传统优化器，一个带有NOTICE消息的消息会被增加到日志文件来表示不支持的特性。还有，当Greenplum数据库退回到传统优化器时，在该查询的执行日志消息中，标签Planner produced plan:会出现在该查询之前。

Note: 可以通过设置Greenplum数据库的服务器配置参数`client_min_messages`为LOG来配置Greenplum数据库在psql命令行中显示日志消息。有关该参数的信息请见Greenplum数据库参考指南。

**Parent topic:** [关于GPORCA](#)

## 示例

此示例显示启用GPORCA时针对分区表运行的查询的差异。

此CREATE TABLE语句创建具有单级分区的表：

```
CREATE TABLE sales (trans_id int, date date,
 amount decimal(9,2), region text)
 DISTRIBUTED BY (trans_id)
 PARTITION BY RANGE (date)
 (START (date '2016-01-01')
 INCLUSIVE END (date '2017-01-01')
 EXCLUSIVE EVERY (INTERVAL '1 month'),
 DEFAULT PARTITION outlying_dates);
```

GPORCA支持下面这个针对该表的查询并且不会在日志文件中生成错误：

```
select * from sales ;
```

EXPLAIN计划输出仅列出所选分区的数量。

```
-> Partition Selector for sales (dynamic scan id: 1)
(cost=10.00..100.00 rows=50 width=4)
Partitions selected: 13EXPLAIN计划输出只列出选中的分区数。 (out of 13)
```

如果GPORCA不支持一个针对分区表的查询，Greenplum数据库会退回到传统优化器。传统优化器生成的EXPLAIN计划会列出选中的分区。下面这个例子展示了解释计划的一部分，它列出了一些选中的分区。

```
-> Append (cost=0.00..0.00 rows=26 width=53)
-> Seq Scan on sales2_1_prt_7_2_prt_usa sales2 (cost=0.00..0.00 rows=1
width=53)
-> Seq Scan on sales2_1_prt_7_2_prt_asia sales2 (cost=0.00..0.00 rows=1
width=53)
...
...
```

这个例子展示了Greenplum数据库从GPORCA退回到传统查询优化器时的日志输出。

当这个查询被运行时，Greenplum数据库会退回到传统查询优化器。

```
explain select * from pg_class;
```

一个消息会被增加到日志文件。该消息包含下面这个NOTICE信息，它指出了GPORCA没有执行该查询的原因：

```
NOTICE, "Feature not supported: Queries on master-only
tables"
```

## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

## □ 查询数据

关于Greenplum的查询  
处理

## □ 关于GPORCA

定义查询

WITH查询 (公用表表达式)

使用函数和操作符

使用JSON数据

使用XML数据

## □ 使用全文搜索

## 关于全文搜索

在数据库表中搜索文本

控制文本搜索

文本搜索附加功能

本主题概述了Greenplum数据库全文搜索，基本文本搜索表达式，配置和自定义文本搜索。以及Greenplum数据库全文搜索与Pivotal GPText的比较。

本节包含以下子主题：

- [什么是文档](#)
- [基本文本匹配](#)
- [配置](#)
- [对比Greenplum数据库文本搜索与Pivotal GPText](#)

全文搜索（或“文本搜索”）提供识别满足查询的自然语言文档的能力，并且有选择地通过与查询的相关性对它们进行排名。最常见的搜索类型是查找包含给定查询字词的所有文档，并按照它们与查询的相似性的顺序返回它们。

Greenplum数据库提供了一个数据类型 `tsvector` 来存储预处理文档，以及一个数据类型 `tsquery` 来存储已处理的查询（[全文搜索类型](#)）。这些数据类型有许多可用的函数和运算符（[文本搜索函数和操作符](#)），其中最重要的是匹配运算符 `@@`，我们在[基本文本匹配](#)中介绍它们。可以使用索引（[文本搜索的GiST和GIN索引](#)）加速全文搜索。

查询和相似性的概念非常灵活，取决于具体的应用。最简单的搜索将查询视为一组单词和文档中查询单词出现频率的相似性。

Greenplum数据库支持文本数据类型的标准文本匹配运算符 `~`, `~*`, `LIKE` 和 `ILIKE`，但这些运算符缺少搜索文档所需的许多基本属性：

- 没有语言支持，即使是英语也是如此。正则表达式是不够的，因为它们不能容易地处理派生词，例如 `satisfies` 和 `satisfy`。您可能会错过包含 `satisfies` 的文档，尽管您可能希望在搜索 `satisfy` 时找到它们。可以使用 OR 来搜索多个派生形式，但这很繁琐且容易出错（有些词可能有几千种衍生词）。
- 它们不提供搜索结果的排序（排名），这使得它们在找到数千个匹配文档时无效。
- 它们往往很慢，因为没有索引支持，所以他们必须为每次搜索处理所有文档。

全文索引允许对文档进行预处理并保存索引以供以后快速搜索。预处理包括：



- 将文档解析为**token**。识别各种类型的令牌（例如，数字，单词，复杂单词，电子邮件地址）是有用的，以便可以不同地处理它们。原则上，令牌类取决于特定的应用程序，但是对于大多数用途来说，使用预定义的一组类就足够了。Greenplum数据库使用解析器执行此步骤。提供了标准解析器，可以根据特定需求创建自定义解析器。
- 将**token**转换为词位。词位是一个字符串，就像一个**token**，但它已被标准化，以便使相同单词的不同形式相似。例如，规范化几乎总是包括将大写字母折叠成小写字母，并且通常涉及删除后缀（例如英语中的s或es）。这允许搜索找到相同单词的变体形式，而无需繁琐地输入所有可能的变体。此外，该步骤通常消除了停止词，这些词是如此常见以至于它们对搜索无用。（简而言之，**token**是文档文本的原始片段，而词位是被认为对索引和搜索有用的词。）Greenplum数据库使用词典来执行此步骤。提供各种标准词典，并且可以为特定需求创建自定义词典。
- 存储为搜索而优化的预处理文档。例如，每个文档可以表示为标准化词位的排序数组。与词位一起，通常希望存储位置信息以用于邻近度排序，使得包含更“密集”的查询词区域的文档被分配比具有分散查询词的更高的等级。

字典允许对**token**如何标准化进行细粒度控制。使用适当的词典，您可以：

- 定义不应编入索引的停止词。
- 使用Ispell将同义词映射到单个单词。
- 使用同义词库将短语映射到单个单词。
- 使用Ispell字典将单词的不同变体映射到规范形式。
- 使用Snowball词干分析器规则将单词的不同变体映射到规范形式。

## 什么是文档

文档是在全文搜索系统中搜索的单位；例如，杂志文章或电子邮件消息。文本搜索引擎必须能够解析文档并存储与其父文档关联的词位（关键词）。稍后，这些关联用于搜索包含查询词的文档。

对于Greenplum数据库中的搜索，文档通常是数据库表的行中的文本字段，或者可能是这些字段的组合（串联），可能存储在若干表中或动态获得。换句话说，可以从不同的部分构造文档以进行索引，并且它可以不作为整体存储在任何地方。例如：

```
SELECT title || ' ' || author || ' ' || abstract || ' '
```

```

|| body AS document
FROM messages
WHERE mid = 12;

SELECT m.title || ' ' || m.author || ' ' || m.abstract || ' '
|| d.body AS document
FROM messages m, docs d
WHERE mid = did AND mid = 12;

```

**Note:**

实际上，在这些示例查询中，应该使用`coalesce`来防止单个NULL属性导致整个文档的NULL结果。

另一种可能性是将文档作为简单文本文件存储在文件系统中。在这种情况下，数据库可用于存储全文索引并执行搜索，并且一些唯一标识符可用于从文件系统检索文档。但是，从数据库外部检索文件需要超级用户权限或特殊功能支持，因此这通常不如将所有数据保留在Greenplum数据库中。此外，将所有内容保留在数据库中可以轻松访问文档元数据，以帮助索引和显示。

出于文本搜索的目的，每个文档必须简化为预处理的`tsvector`格式。搜索和排名完全基于文档的`tsvector`表示执行 - 只有在选择文档以显示给用户时才需要检索原始文本。因此，我们经常将`tsvector`称为文档，但当然它只是完整文档的紧凑表示。

## 基本文本匹配

Greenplum数据库中的全文搜索基于匹配运算符`@@`，如果`tsvector`（文档）与`tsquery`（查询）匹配，则返回`true`。首先写入哪种数据类型无关紧要：

```

SELECT 'a fat cat sat on a mat and ate a fat rat'::tsvector
@@ 'cat & rat'::tsquery;
?column?

t

SELECT 'fat & cow'::tsquery @@ 'a fat cat sat on a mat and
ate a fat rat'::tsvector;
?column?

f

```

如上例所示，`tsquery`不仅仅是原始文本，也不仅仅是`tsvector`。`tsquery`包含搜索项，它必须是已经规范化的词位，并且可以使`AND`、`OR`、`NOT`

用 , 和 & 运算符组合多个项。 (有关详细信息, 请参阅。) 有一些函数`to_tsquery`和`plainto_tsquery`有助于将用户编写的文本转换为正确的`tsquery`, 例如通过规范化文本中出现的单词。同样, `to_tsvector`用于解析和规范化文档字符串。所以在实践中, 文本搜索匹配看起来更像是这样的:

```
SELECT to_tsvector('fat cats ate fat rats') @@
to_tsquery('fat & rat');
?column?

t
```

如果写为, 则发现此匹配不会成功

```
SELECT 'fat cats ate fat rats'::tsvector @@ to_tsquery('fat
& rat');
?column?

f
```

因为这里没有`rats`这个词的标准化。`tsvector`的元素是词位, 假设已经标准化, 因此`rats`与`rat`不匹配。

`@@`运算符还支持文本输入, 允许在简单情况下将文本字符串显式转换为`tsvector`或`tsquery`。可用的变体是:

```
tsvector @@ tsquery
tsquery @@ tsvector
text @@ tsquery
text @@ text
```

我们已经看到了前两个。形式`text @@ tsquery`等同于`to_tsvector(x) @@ y`。形式`text @@ text`等同于`to_tsvector(x) @@ plainto_tsquery(y)`。

## 配置

以上都是简单的文本搜索示例。如前所述, 全文搜索功能包括执行更多操作的能力: 跳过索引某些单词 (停止词), 处理同义词, 以及使用复杂的解析, 例如, 基于不仅仅是空格的解析。此功能由文本搜索配置控制。Greenplum数据库附带了许多语言的预定义配置, 您可以轻松创建自己的配置。 (psql的\df命令显示所有可用的配置。)

在安装过程中, 会选择适当的配置, 并在`postgresql.conf`中相应

地设置`default_text_search_config`。如果您对整个集群使用相同的文本搜索配置，则可以使用`postgresql.conf`中的值。要在整个集群中使用不同的配置，但在任何一个数据库中使用相同的配置，请使用`ALTER DATABASE ... SET`。否则，您可以在每个会话中设置`default_text_search_config`。

每个依赖于配置的文本搜索功能都有一个可选的`regconfig`参数，因此可以显式指定要使用的配置。

`default_text_search_config`仅在省略此参数时使用。

为了更容易构建自定义文本搜索配置，可以从更简单的数据库对象构建配置。Greenplum数据库的文本搜索工具提供了四种与配置相关的数据库对象：

- 文本搜索解析器将文档分解为token并对每个token进行分类（例如，作为单词或数字）。
- 文本搜索词典将token转换为标准化形式并拒绝停止词。
- 文本搜索模板提供基本字典的功能。（字典只是为模板指定模板和一组参数。）
- 文本搜索配置选择一个解析器和一组字典，用于规范化解析器生成的token。

文本搜索解析器和模板是从低级C函数构建的；因此，它需要C编程才能开发新的，并且需要超级用户权限才能将其安装到数据库中。

（在Greenplum数据库发行版的`contrib`/区域中有附加解析器和模板的示例。）由于字典和配置只是参数化并将一些底层解析器和模板连接在一起，因此创建新字典或配置不需要特殊权限。本章稍后将介绍创建自定义词典和配置的示例。

## 对比Greenplum数据库文本搜索与Pivotal GPText

Greenplum数据库文本搜索是将PostgreSQL文本搜索移植到Greenplum数据库MPP平台。Pivotal还提供Pivotal GPText，它将Greenplum数据库与Apache Solr文本搜索平台集成在一起。GPText在您的Greenplum数据库集群旁边安装Apache Solr集群，并提供Greenplum数据库功能，您可以使用它来创建Solr索引，查询它们，并在数据库会话中接收结果。

这两个系统都提供强大的企业级文档索引和搜索服务。Greenplum数据库文本搜索可立即供您使用，无需安装和维护其他软件。如果它符合您的应用程序的要求，您应该使用它。

带有Solr的GPText具有Greenplum数据库文本搜索所不具备的许多功能。特别是，GPText更适合高级文本分析应用程序。以下是将GPText用于文本搜索应用程序时可以使用的一些优点和功能。

- Apache Solr集群可以与数据库分开扩展。Solr节点可以部署在Greenplum数据库主机上，也可以部署在网络上的不同主机上。
- 可以将索引和搜索工作负载从Greenplum数据库移出到Solr，以维护数据库查询性能。
- GPText创建分割为分片的Solr索引，每个Greenplum数据库segment一个，因此Greenplum数据库MPP体系结构的优势扩展到了文本搜索工作负载。
- 使用Solr索引和搜索文档非常快，可以通过向集群添加更多Solr节点来扩展。
- 文档内容可以存储在Greenplum数据库表，Solr索引或两者中。
- 通过GPText，Solr可以索引存储为Greenplum数据库表格中文本的文档，以及使用HTTP，FTP，S3或HDFS URL访问的外部存储中的文档。
- Solr自动识别大多数丰富的文档格式，并分别索引文档内容和元数据。
- Solr索引可高度自定义。您可以将文本分析链自定义到字段级别。
- 除了Apache项目提供的大量语言，标记符和过滤器之外，GPText还提供社交媒体标记器，国际文本标记器和通用查询解析器，它们可以理解几种常见的文本搜索语法。
- GPText API支持高级文本分析工具，例如分面，命名实体识别(NER) 和词性(POS)识别。

有关GPText的详细信息，请参阅[GPText文档网站](#)。

**Parent topic:** [使用全文搜索](#)

## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

关于Greenplum的查询  
处理

## □ 关于GPORCA

## 定义查询

## WITH查询 (公用表表达式)

## 使用函数和操作符

## 使用JSON数据

## 使用XML数据

## □ 使用全文搜索

## 关于全文搜索

在数据库表中搜索文本

控制文本搜索

文本搜索附加功能

本主题说明如何使用文本搜索运算符搜索数据库表以及如何创建索引以加快文本搜索。

上一节中的示例说明了使用简单常量字符串进行全文匹配。本节介绍如何搜索表数据, 可选择使用索引。

本节包含以下子主题:

- [搜索表](#)
- [创建索引](#)

## 搜索表

可以在没有索引的情况下进行全文搜索。在其body字段中打印包含单词friend的每一行的title的简单查询是:

```
SELECT title
FROM pgweb
WHERE to_tsvector('english', body) @@ to_tsquery('english',
'friend');
```

这也将找到诸如friends和friendly之类的相关单词, 因为所有这些都被简化为相同的标准化词汇。

上面的查询指定english配置用于解析和规范化字符串。或者, 我们可以省略配置参数:

```
SELECT title
FROM pgweb
WHERE to_tsvector(body) @@ to_tsquery('friend');
```

此查询将使用[default\\_text\\_search\\_config](#) 设置的配置。

更复杂的示例是在title或body中选择包含create和table的十个最新文档:

```
SELECT title
FROM pgweb
WHERE to_tsvector(title || ' ' || body) @@ to_tsquery('create & table')
ORDER BY last_mod_date DESC
```



```
LIMIT 10;
```

为清楚起见，我们省略了在两个字段之一中查找包含NULL的行所需的coalesce函数调用。

虽然这些查询在没有索引的情况下可以正常工作，但大多数应用程序会发现这种方法太慢，除非是偶尔的临时搜索。 文本搜索的实际使用通常需要创建索引。

## 创建索引

我们可以创建一个GIN索引（[文本搜索的GiST和GIN索引](#)）来加速文本搜索：

```
CREATE INDEX pgweb_idx ON pgweb USING
gin(to_tsvector('english', body));
```

请注意，使用了to\_tsvector的双参数版本。 只能在表达式索引中使用指定配置名称的文本搜索功能。 这是因为索引内容必须不受[default\\_text\\_search\\_config](#)的影响。 如果它们受到影响，索引内容可能会不一致，因为不同的条目可能包含使用不同文本搜索配置创建的tsvector，并且无法猜测哪个是哪个。 正确地导出和恢复这样的索引是不可能的。

因为在上面的索引中使用了to\_tsvector的双参数版本，所以只有使用具有相同配置名称的to\_tsvector的双参数版本的查询引用才会使用该索引。 也就是说，`WHERE to_tsvector('english', body) @@ 'a & b'`可以使用索引，但`WHERE to_tsvector(body) @@ 'a & b'`不能。 这可确保索引仅与用于创建索引条目的相同配置一起使用。

可以设置更复杂的表达式索引，其中配置名称由另一列指定，例如：

```
CREATE INDEX pgweb_idx ON pgweb USING
gin(to_tsvector(config_name, body));
```

其中config\_name是pgweb表中的一列。 这允许在同一索引中进行混合配置，同时记录每个索引条目使用的配置。 例如，如果文档集合包含不同语言的文档，这将是有用的。 同样，要使用索引的查询必须用来匹配，例如，`WHERE to_tsvector(config_name, body) @@ 'a & b'`。

索引甚至可以连接列：

```
CREATE INDEX pgweb_idx ON pgweb USING
gin(to_tsvector('english', title || ' ' || body));
```

另一种方法是创建一个单独的tsvector列来保存to\_tsvector的输出。此示例是title和body的连接，使用coalesce确保当另一个为NULL时仍将索引一个字段：

```
ALTER TABLE pgweb ADD COLUMN textsearchable_index_col
tsvector;
UPDATE pgweb SET textsearchable_index_col =
 to_tsvector('english', coalesce(title,'') || ' ' ||
coalesce(body,''));
```

然后我们创建一个GIN索引来加速搜索：

```
CREATE INDEX textsearch_idx ON pgweb USING
gin(textsearchable_index_col);
```

现在我们准备执行快速全文搜索：

```
SELECT title FROM pgweb WHERE textsearchable_index_col @@
to_tsquery('create & table')
ORDER BY last_mod_date DESC LIMIT 10;
```

与表达式索引相比，单列方法的一个优点是，不必在查询中显式指定文本搜索配置以便使用索引。如上例所示，查询可以依赖于default\_text\_search\_config。另一个优点是搜索速度更快，因为没有必要重做to\_tsvector调用来验证索引匹配。（当使用GiST索引而不是GIN索引时，这一点更为重要；请参阅[文本搜索的GiST和GIN索引](#)。）然而，表达式索引方法设置起来更简单，并且由于tsvector不是显式存储，因此需要更少的磁盘空间。

**Parent topic:** [使用全文搜索](#)

## Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 定义数据库对象
    - 分布与倾斜
    - 插入, 更新, 和删除数据
  - 查询数据
    - 关于Greenplum的查询处理
    - 关于GPORCA
    - 定义查询
    - WITH查询 (公用表表达式)
    - 使用函数和操作符
    - 使用JSON数据
    - 使用XML数据
  - 使用全文搜索
    - 关于全文搜索
    - 在数据库表中搜索文本
- 控制文本搜索**
- 文本搜索附加功能

本主题说明如何创建搜索和查询向量, 如何对搜索结果进行排名, 以及如何在文本搜索查询的结果中突出显示搜索词。

要实现全文搜索, 必须有一个函数来从文档创建`tsvector`和从用户查询创建`tsquery`。此外, 我们需要以有用的顺序返回结果, 因此我们需要一个函数来比较文档与查询的相关性。能够很好地显示结果也很重要。Greenplum数据库为所有这些功能提供支持。

本主题包含以下子主题:

- [解析文件](#)
- [解析查询](#)
- [排名搜索结果](#)
- [突出显示结果](#)

## 解析文件

Greenplum数据库提供了`to_tsvector`函数, 用于将文档转换为`tsvector`数据类型。

```
to_tsvector([config regconfig,] document text) returns
tsvector
```

`to_tsvector`将文本文档解析为`token`, 将`token`缩减为词位, 并返回一个`tsvector`, 其中列出了词位及其在文档中的位置。根据指定或默认文本搜索配置处理文档。这是一个简单的例子:

```
SELECT to_tsvector('english', 'a fat cat sat on a mat - it
ate a fat rats');
 to_tsvector

'ate':9 'cat':3 'fat':2,11 'mat':7 'rat':12 'sat':4
```

在上面的例子中, 我们看到生成的`tsvector`不包含单词`a`, `on`或`it`, 单词`rats`变成`rat`, 标点符号`-`被忽略。

`to_tsvector`函数在内部调用解析器, 该解析器将文本分解为`token`并为每个`token`分配类型。对于每个`token`, 查阅字典列表 ([文本搜索词典](#)), 其中列表可以根据`token`类型而变化。识别`token`的第

一个字典发出一个或多个规范化的词位来表示token。例如，`rats`变成`rat`，因为其中一个词典认识到单词`rats`是复数形式的`rat`。有些单词被识别为停止词，这会导致它们被忽略，因为它们过于频繁而无法用于搜索。在我们的例子中，这些是`a`, `on`和`it`。如果列表中没有字典识别出`token`，那么它也会被忽略。在这个标点符号发生的例子中 - 因为实际上没有为其`token`类型（空间符号）分配字典，这意味着空间`token`永远不会被索引。解析器，字典和要索引的`token`类型的选择由所选文本搜索配置（[文本搜索配置示例](#)）确定。可以在同一数据库中具有许多不同的配置，并且可以为各种语言提供预定义的配置。在我们的示例中，我们使用英语的默认配置`english`。

函数`setweight`可用于标记具有给定权重的`tsvector`的条目，其中权重是字母A, B, C或D之一。这通常用于标记来自文档的不同部分的条目，如`title`与`body`。之后，此信息可用于搜索结果的排名。

因为`to_tsvector(NULL)`将返回`NULL`，所以只要字段可能为`null`，建议使用`coalesce`。以下是从结构化文档创建`tsvector`的推荐方法：

```
UPDATE tt SET ti =
setweight(to_tsvector(coalesce(title,'')), 'A')
|| setweight(to_tsvector(coalesce(keyword,'')), 'B')
|| setweight(to_tsvector(coalesce(abstract,'')), 'C')
|| setweight(to_tsvector(coalesce(body,'')), 'D');
```

这里我们使用`setweight`标记完成的`tsvector`中每个词位的来源，然后使用`tsvector`连接运算符`||`合并标记的`tsvector`值。（[文本搜索附加功能](#)提供有关这些操作的详细信息。）

## 解析查询

Greenplum数据库提供了`to_tsquery`和`plainto_tsquery`函数，用于将查询转换为`tsquery`数据类型。`to_tsquery`提供了比`plainto_tsquery`更多功能的访问权限，但对其输入的宽容度较低。

```
to_tsquery([config regconfig,] querytext text) returns
tsquery
```

`to_tsquery`从`querytext`创建一个`tsquery`值，该值必须由布尔运算符`&` (AND), `|` (OR) 和 `!` (NOT) 分隔的单个`token`组成。可以使用括号对这些运算符进行分组。换句话说，`to_tsquery`的输入必须已经遵循`tsquery`输入的一般规则，如[全文搜索类型](#)中所述。不同之处在

tsquery

token

to tsquery

于，虽然基本的输入将置于面值，但使用指定或默认配置将每个token规范化为一个词位，并根据配置丢弃任何停止词的标记。例如：

```
SELECT to_tsquery('english', 'The & Fat & Rats');
 to_tsquery

'fat' & 'rat'
```

与基本的tsquery输入一样，权重可以附加到每个词位，以限制它仅匹配那些权重的tsvector词位。例如：

```
SELECT to_tsquery('english', 'Fat | Rats:AB');
 to_tsquery

'fat' | 'rat':AB
```

此外，\*可以附加到词位以指定前缀匹配：

```
SELECT to_tsquery('supern:*A & star:A*B');
 to_tsquery

'supern':*A & 'star':*AB
```

这样的词位将匹配以给定字符串开头的tsvector中的任何单词。

to\_tsquery也可以接受单引号短语。当配置包括可能触发此类短语的同义词词典时，这是非常有用的。在下面的例子中，同义词库包含规则supernovae stars : sn：

```
SELECT to_tsquery('supernovae stars' & !crab);
 to_tsquery

'sn' & '!crab'
```

如果没有引号，to\_tsquery将为未被AND或OR运算符分隔的标记生成语法错误。

```
plainto_tsquery([config regconfig,] querytext ext)
returns tsquery
```

plainto\_tsquery未格式化的文本querytext转换为tsquery。对于to\_tsvector，文本被解析和规范化，然后在残存的单词之间插入& (AND) 布尔运算符。

示例：

```
SELECT plainto_tsquery('english', 'The Fat Rats');
plainto_tsquery

'fat' & 'rat'
```

请注意，`plainto_tsquery`无法在其输入中识别布尔运算符，权重标签或前缀匹配标签：

```
SELECT plainto_tsquery('english', 'The Fat & Rats:C');
plainto_tsquery

'fat' & 'rat' & 'c'
```

这里，所有输入标点符号都被丢弃为空格符号。

## 排名搜索结果

排名尝试测量相关文档对特定查询的影响，以便在存在多个匹配时，可以首先显示最相关的匹配。Greenplum数据库提供两个预定义的排名功能，其中考虑了词位，接近度和结构信息；也就是说，他们会考虑查询term在文档中出现的频率，term在文档中的接近程度，以及文档中出现的部分的重要程度。然而，相关性的概念是模糊的并且非常特定于应用程序。不同的应用程序可能需要用于排名的附加信息，例如文档修改时间。内置排名功能仅是示例。您可以编写自己的排名功能和/或将其结果与其他因素结合起来，以满足您的特定需求。

目前可用的两个排名功能是：

```
ts_rank([weights float4[],] vector tsvector,
query tsquery [, normalization integer]) returns
float4
```

根据匹配词位的频率对矢量进行排名。

```
ts_rank_cd([weights float4[],] vector tsvector,
query tsquery [, normalization integer]) returns
float4
```

此函数计算给定文档向量和查询的盖度排名，如Clarke, Cormack和Tudhope在1999年的“信息处理和管理”杂志中的“一至三期查询的相关性排名”中所述。除了考虑到匹配词位之间的接近程度，盖度类似`ts_rank`排名。

此函数需要词位位置信息来执行其计算。因此，它忽略了`tsvector`中的任何“stripped”词位。如果输入中没有未提取的词位，则结果为零。（有关`tsvector`中`strip`函数和位置信息的更多信息，请参阅[操纵文档](#)。）

对于这两个函数，可选的权重参数提供了对字实例进行或多或少权衡的能力，具体取决于它们的标注方式。权重数组按以下顺序指定对每个单词类别进行称重的程度：

```
{D-weight, C-weight, B-weight, A-weight}
```

如果未提供权重，则使用以下默认值：

```
{0.1, 0.2, 0.4, 1.0}
```

通常，权重用于标记文档特殊区域中的单词，如标题或初始摘要，因此可以比文档正文中的单词更重要或更不重视它们。

由于较长的文档更有可能包含查询词语，因此考虑文档大小是合理的，例如，具有五个搜索字实例的百字文档可能比具有五个实例的千字文档更相关。两个排名函数都采用整数规范化选项，该选项指定文档的长度是否以及如何影响其排名。整数选项控制多个行为，因此它是一个位掩码：您可以使用|指定一个或多个行为（例如，2|4）。

- 0（默认值）忽略文档长度
- 1 将等级除以(1 + 文档长度的对数)
- 2 将等级除以文件长度
- 4 将等级除以范围之间的平均谐波距离（这仅由`ts_rank_cd`实现）
- 8 将等级除以文档中唯一单词的数量
- 16 将等级除以(1 + 文档中唯一字数的对数)
- 32 将等级除以(等级 + 1)

如果指定了多个标志位，则按列出的顺序应用转换。

重要的是要注意，排名函数不使用任何全局信息，因此不可能像有时期望的那样产生1%或100%的公平标准化。归一化选项32(`rank / (rank+1)`)可用于将所有等级缩放到0到1的范围内，但当然这只是一个表面的变化；它不会影响搜索结果的排序。

以下示例仅选择排名最高的十个匹配项：

```

SELECT title, ts_rank_cd(textsearch, query) AS rank
FROM apod, to_tsquery('neutrino|(dark & matter)') query
WHERE query @@ textsearch
ORDER BY rank DESC
LIMIT 10;

```

| title                                         | rank     |
|-----------------------------------------------|----------|
| Neutrinos in the Sun                          | 3.1      |
| The Sudbury Neutrino Detector                 | 2.4      |
| A MACHO View of Galactic Dark Matter          | 2.01317  |
| Hot Gas and Dark Matter                       | 1.91171  |
| The Virgo Cluster: Hot Plasma and Dark Matter | 1.90953  |
| Rafting for Solar Neutrinos                   | 1.9      |
| NGC 4650A: Strange Galaxy and Dark Matter     | 1.85774  |
| Hot Gas and Dark Matter                       | 1.6123   |
| Ice Fishing for Cosmic Neutrinos              | 1.6      |
| Weak Lensing Distorts the Universe            | 0.818218 |

这是使用标准化排名的相同示例：

```

SELECT title, ts_rank_cd(textsearch, query, 32 /*
rank/(rank+1) */) AS rank
FROM apod, to_tsquery('neutrino|(dark & matter)') query
WHERE query @@ textsearch
ORDER BY rank DESC
LIMIT 10;

```

| title                                         | rank              |
|-----------------------------------------------|-------------------|
| Neutrinos in the Sun                          | 0.756097569485493 |
| The Sudbury Neutrino Detector                 | 0.705882361190954 |
| A MACHO View of Galactic Dark Matter          | 0.668123210574724 |
| Hot Gas and Dark Matter                       | 0.65655958650282  |
| The Virgo Cluster: Hot Plasma and Dark Matter | 0.656301290640973 |
| Rafting for Solar Neutrinos                   | 0.655172410958162 |
| NGC 4650A: Strange Galaxy and Dark Matter     | 0.650072921219637 |
| Hot Gas and Dark Matter                       | 0.617195790024749 |
| Ice Fishing for Cosmic Neutrinos              | 0.615384618911517 |
| Weak Lensing Distorts the Universe            | 0.450010798361481 |

排名代价可能很高，因为它需要查询每个匹配文档的tsvector，这可能是I/O相关的，因此很慢。不幸的是，几乎不可能避免，因为实际的查询经常导致大量的匹配。

# 突出显示结果

要显示搜索结果，最好显示每个文档的一部分以及它与查询的关联方式。通常，搜索引擎会显示带有标记搜索词的文档片段。Greenplum数据库提供了一个实现此功能的函数`ts_headline`。

```
ts_headline([config regconfig,] document text, query
tsquery [, options text]) returns text
```

`ts_headline`接受文档和查询，并返回文档中的摘录，其中突出显示查询中的术语。可以通过`config`指定用于解析文档的配置；如果省略`config`，则使用`default_text_search_config`配置。

如果指定了`options`字符串，则它必须包含一个或多个逗号分隔的`option=value`列表对。可用选项包括：

- `StartSel`, `StopSel`: 用于分隔出现在文档中的查询单词的字符串，以区别于其他摘录的单词。如果这些字符串包含空格或逗号，则必须双引号。
- `MaxWords`, `MinWords`: 这些数字决定了输出的最长和最短标题。
- `ShortWord`: 这个长度或更短的单词将在标题的开头和结尾处删除。默认值为3消除了常见的英文文章。
- `HighlightAll`: 布尔标志；如果为`true`，则整个文档将用作标题，忽略前面的三个参数。
- `MaxFragments`: 要显示的最大文本摘要或片段数。默认值零选择非面向片段的标题生成方法。大于零的值选择基于片段的标题生成。此方法查找具有尽可能多的查询字的文本片段，并在查询字周围拉伸这些片段。结果，查询单词接近每个片段的中间并且每侧都有单词。每个片段最多为`MaxWords`，并且在每个片段的开始和结束处删除长度为`ShortWord`或更短的单词。如果不是在文档中找到所有查询词，则将显示文档中第一个`MinWords`的单个片段。
- `FragmentDelimiter`: 当显示多个片段时，片段将被此字符串分隔。

任何未指定的选项都会收到以下默认值：

```
StartSel=, StopSel=,
MaxWords=35, MinWords=15, ShortWord=3, HighlightAll=FALSE,
MaxFragments=0, FragmentDelimiter=" ... "
```

## 示例:

```
SELECT ts_headline('english',
 'The most common type of search
is to find all documents containing given query terms
and return them in order of their similarity to the
query.',
 to_tsquery('query & similarity'));
 ts_headline

-
containing given query terms
and return them in order of their similarity to the
query.

SELECT ts_headline('english',
 'The most common type of search
is to find all documents containing given query terms
and return them in order of their similarity to the
query.',
 to_tsquery('query & similarity'),
 'StartSel = <, StopSel = >');
 ts_headline

containing given <query> terms
and return them in order of their <similarity> to the
<query>.
```

`ts_headline`使用原始文档，而不是`tsvector`摘要，因此它可能很慢，应谨慎使用。一个典型的错误是当只显示十个文档时，为每个匹配的文档调用`ts_headline`。SQL子查询可以提供帮助；这是一个例子：

```
SELECT id, ts_headline(body, q), rank
FROM (SELECT id, body, q, ts_rank_cd(ti, q) AS rank
 FROM apod, to_tsquery('stars') q
 WHERE ti @@ q
 ORDER BY rank DESC
 LIMIT 10) AS foo;
```

**Parent topic:** [使用全文搜索](#)

## Greenplum数据库® 6.0文档

## □ 管理员指南

- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

关于Greenplum的查询  
处理

## □ 关于GPORCA

## 定义查询

## WITH查询 (公用表表达式)

## 使用函数和操作符

## 使用JSON数据

## 使用XML数据

## □ 使用全文搜索

## 关于全文搜索

## 在数据库表中搜索文本

## 控制文本搜索

## 文本搜索附加功能

Greenplum数据库具有其他附加功能和操作符，您可以使用它们来操作搜索和查询向量，以及重写搜索查询。

本节包含以下子主题：

- [操纵文档](#)
- [操纵查询](#)
- [重写查询](#)
- [收集文档统计信息](#)

## 操纵文档

[解析文件](#)显示了如何将原始文本文档转换为`tsvector`值。

Greenplum数据库还提供了可用于处理已经处于`tsvector`形式的文档的函数和运算符。

`tsvector || tsvector`

`tsvector`连接运算符返回一个向量，该向量合并了作为参数给出的两个向量的词位和位置信息。在连接期间保留位置和重量标签。出现在右侧向量中的位置按左侧向量中提到的最大位置偏移，因此结果几乎等于在两个原始文档字符串的串联上执行`to_tsvector`的结果。（等价不精确，因为从左手参数末尾删除的任何停止词都不会影响结果，而如果使用文本连接，它们会影响右手参数中词位的位置。）

在向量形式中使用连接的一个优点是，您可以使用不同的配置来解析文档的不同部分，而不是在应用`to_tsvector`之前连接文本。此外，因为`setweight`函数以相同的方式标记给定向量的所有词位，所以如果要使用不同的权重标记文档的不同部分，则必须在连接之前解析文本并执行`setweight`。

`setweight(vector tsvector, weight "char") returns tsvector`

`setweight`返回输入向量的副本，其中每个位置都标有给定的权重，A, B, C, 或D。（D是新向量的默认值，因此  在输出中显示。）这些标签在连接向量时保留，允许来自文档的不同部分的单词通过排名函数被不同地加权。

请注意，重量标签适用于位置，而不是词位。如果输入向量已被剥离，则setweight不执行任何操作。

`length(vector tsvector) returns integer`

返回向量中存储的词位数。

`strip(vector tsvector) returns tsvector`

返回一个向量，该向量列出与给定向量相同的词位，但缺少任何位置或权重信息。虽然返回的向量比未被剥离的向量用于相关性排名要小得多，但它通常会小得多。

## 操纵查询

[解析查询](#)显示了如何将原始文本查询转换为tsquery值。

Greenplum数据库还提供了可用于操作已经处于tsquery形式的查询的函数和运算符。

`tsquery && tsquery`

返回两个给定查询的AND组合。

`tsquery || tsquery`

返回两个给定查询的OR组合。

`!! tsquery`

返回给定查询的非 (NOT)。

`numnode(query tsquery) returns integer`

返回tsquery中的节点数（词位加运算符）。此函数可用于确定查询是否有意义（返回`> 0`），或仅包含停止词（返回`0`）。例子：

```
SELECT numnode(plainto_tsquery('the any'));
NOTICE: query contains only stopword(s) or doesn't
contain lexeme(s), ignored
numnode

0

SELECT numnode('foo & bar'::tsquery);
numnode

```

```
querytree(query tsquery) returns text
```

返回可用于搜索索引的tsquery部分。此函数对于检测不可索引的查询很有用，例如那些仅包含停止词或仅包含否定词的查询。例如：

```
SELECT querytree(to_tsquery('!defined'));
querytree

```

## 重写查询

`ts_rewrite`系列函数在给定的`tsquery`中搜索目标子查询的出现，并在每次出现替换子查询。实质上，此操作是子串替换的特定于`tsquery`的版本。目标和替代组合可以被认为是查询重写规则。这种重写规则的集合可以是强大的搜索辅助工具。例如，您可以使用同义词（例如，new york, big apple, nyc, gotham）扩展搜索范围，或者缩小搜索范围以引导用户查看某些热门话题。此功能与同义词词典（[同义词词典](#)）之间在功能上存在一些重叠。但是，您可以在不重建索引的情况下即时修改一组重写规则，而更新同义词库则需要重建索引才能生效。

```
ts_rewrite(query tsquery, target tsquery,
substitute tsquery) returns tsquery
```

这种形式的`ts_rewrite`只应用一个重写规则：`target`在查询中出现的任何地方都被`substitute`替换。例如：

```
SELECT ts_rewrite('a & b':::tsquery, 'a':::tsquery,
'c':::tsquery);
ts_rewrite

'b' & 'c'
```

```
ts_rewrite(query tsquery, select text) returns
tsquery
```

这种形式的`ts_rewrite`接受一个起始查询和一个SQL select命令，它以文本字符串形式给出。select必须产生两列`tsquery`类型。对于select结果的每一行，第一列值（target）的出现被当前查询值中的第二列值（substitute）替换。例如：

```

CREATE TABLE aliases (id int, t tsquery, s tsquery);
INSERT INTO aliases VALUES('a', 'c');

SELECT ts_rewrite('a & b'::tsquery, 'SELECT t,s FROM
aliases');
ts_rewrite

'b' & 'c'

```

请注意，当以这种方式应用多个重写规则时，应用程序的顺序可能很重要；所以在实践中你会希望源查询ORDER BY一些排序键。

让我们考虑一个现实生活中的天文例子。我们将使用表驱动的重写规则扩展查询supernovae：

```

CREATE TABLE aliases (id int, t tsquery primary key, s
tsquery);
INSERT INTO aliases VALUES(1, to_tsquery('supernovae'),
to_tsquery('supernovae|sn'));

SELECT ts_rewrite(to_tsquery('supernovae & crab'), 'SELECT
t, s FROM aliases');
ts_rewrite

'crab' & ('supernova' | 'sn')

```

我们可以通过更新表来更改重写规则：

```

UPDATE aliases
SET s = to_tsquery('supernovae|sn & !nebulae')
WHERE t = to_tsquery('supernovae');

SELECT ts_rewrite(to_tsquery('supernovae & crab'), 'SELECT
t, s FROM aliases');
ts_rewrite

'crab' & ('supernova' | 'sn' & !'nebula')

```

当有许多重写规则时，重写可能会很慢，因为它会检查每个规则是否存在可能的匹配。为了过滤掉明显的非候选规则，我们可以使用tsquery类型的包含运算符。在下面的示例中，我们仅选择可能与原始查询匹配的规则：

```

SELECT ts_rewrite('a & b'::tsquery,
 'SELECT t,s FROM aliases WHERE ''a &
b'''::tsquery @> t');
ts_rewrite

'b' & 'c'

```

# 收集文档统计信息

函数`ts_stat`可用于检查配置和查找停止词候选。

```
ts_stat(sqlquery text, [weights text,]
 OUT word text, OUT ndoc integer,
 OUT nentry integer) returns setof record
```

`sqlquery`是一个包含SQL查询的文本值，该查询必须返回单个`tsvector`列。`ts_stat`执行查询并返回有关`tsvector`数据中包含的每个不同词位（`word`）的统计信息。返回的列是

- `word text` — 词位的值
- `ndoc integer` — 出现这个词的文件数量（`tsvectors`）
- `nentry integer` — 单词出现总数

如果提供权重，则仅计算具有这些权重之一的事件。

例如，要查找文档集合中十个最常用的单词：

```
SELECT * FROM ts_stat('SELECT vector FROM apod')
ORDER BY nentry DESC, ndoc DESC, word
LIMIT 10;
```

相同，但只计算权重为A或B的单词出现次数：

```
SELECT * FROM ts_stat('SELECT vector FROM apod', 'ab')
ORDER BY nentry DESC, ndoc DESC, word
LIMIT 10;
```

**Parent topic:** [使用全文搜索](#)

## Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 定义数据库对象
- 分布与倾斜

## 插入, 更新, 和删除数据

- 查询数据
  - 关于Greenplum的查询处理
  - 关于GPORCA
  - 定义查询
  - WITH查询 (公用表表达式)
  - 使用函数和操作符
  - 使用JSON数据
  - 使用XML数据
- 使用全文搜索
  - 关于全文搜索
  - 在数据库表中搜索文本
  - 控制文本搜索
  - 文本搜索附加功能

本主题描述了Greenplum数据库文本搜索解析器从原始文本生成的token类型。

文本搜索解析器负责将原始文档文本拆分为token并标识每个token的类型，其中可能的类型集由解析器本身定义。请注意，解析器根本不会修改文本 - 它只是识别合理的单词边界。由于范围有限，因此与自定义词典相比，对特定于应用程序的自定义解析器的需求较少。目前，Greenplum数据库只提供了一个内置解析器，它已被发现可用于各种应用程序。

内置解析器名为`pg_catalog.default`。它识别23种token类型，如下表所示。

Table 1. 默认解析器的token类型

| 别名              | 描述             | 示例                                                    |
|-----------------|----------------|-------------------------------------------------------|
| asciword        | 单词，全部为ASCII字母  | elephant                                              |
| word            | 单词，全部为字母       | mañana                                                |
| numword         | 单词，字母和数字       | beta1                                                 |
| asciihword      | 连字符，全部为ASCII   | up-to-date                                            |
| hword           | 连字符，全部为字母      | lógico-matemática                                     |
| numhword        | 连字符，字母和数字      | postgresql-beta1                                      |
| hword_asciipart | 连字符部分，全部为ASCII | postgresql in the context postgresql-beta1            |
| hword_part      | 连字符部分，全部为字母    | lógico or matemática in the context lógico-matemática |
| hword_numpart   | 连字符部分，字母和数字    | beta1 in the context postgresql-beta1                 |
| email           | Email地址        | foo@example.com                                       |
| protocol        | 协议头            | http://                                               |
| url             | URL            | example.com/stuff/index.html                          |

| host     | Host   | example.com                                |
|----------|--------|--------------------------------------------|
| url_path | URL路径  | /stuff/index.html, in the context of a URL |
| file     | 文件或路径名 | /usr/local/foo.txt, if not within a URL    |
| sfloat   | 科学计数法  | -1.234e56                                  |
| float    | 十进制表示法 | -1.234                                     |
| int      | 有符号数   | -1234                                      |
| uint     | 无符号数   | 1234                                       |
| version  | 版本号    | 8.3.0                                      |
| tag      | XML标记  | <a href="dictionaries.html">               |
| entity   | XML实体  | &amp;                                      |
| blank    | 空白字符   | (任何未经其他方式认可的空格或标点符号)                       |

Note:

解析器的“字母”概念由数据库的语言环境设置决定，特别是lc\_ctype。仅包含基本ASCII字母的单词将作为单独的token类型报告，因为区分它们有时很有用。在大多数欧洲语言中，token类型word和asciword应该被视为相似。

电子邮件不支持RFC 5322定义的所有有效电子邮件字符。具体而言，电子邮件用户名支持的唯一非字母数字字符是句号，短划线和下划线。

解析器可以从同一段文本生成重叠token。例如，一个带连字符的单词将作为整个单词和每个组件报告：

```
SELECT alias, description, token FROM ts_debug('foo-bar-beta1');
 alias | description
 | token
-----+-----
 numhword | Hyphenated word, letters and digits
 | foo-bar-beta1
 hword_asciipart | Hyphenated word part, all ASCII
 | foo
 blank | Space symbols
 | -
 hword_asciipart | Hyphenated word part, all ASCII
 | bar
 blank | Space symbols
 | -
 hword_numpart | Hyphenated word part, letters and digits
```

| beta1

这种行为是可取的，因为它允许搜索适用于整个复合词和组件。这是另一个有指导性的例子：

```
SELECT alias, description, token FROM
ts_debug('http://example.com/stuff/index.html');
 alias | description | token
-----+-----+-----+
 protocol | Protocol head | http://
 url | URL | example.com/stuff/index.html
 host | Host | example.com
 url_path | URL path | /stuff/index.html
```

**Parent topic:** [使用全文搜索](#)

Greenplum数据库® 6.0文档

## □ 管理员指南

- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

## □ 查询数据

关于Greenplum的查询  
处理

### □ 关于GPORC

定义查询

WITH查询 (公用表表  
达式)

使用函数和操作符

使用JSON数据

使用XML数据

### □ 使用全文搜索

关于全文搜索

在数据库表中搜索文  
本

控制文本搜索

文本搜索附加功能

由Greenplum数据库全文搜索解析器生成的token通过一系列字典传递以产生标准化术语或“词位”。不同种类的词典可用于以不同方式和不同语言过滤和转换token。

本节包含以下子主题：

- [关于文本搜索词典](#)
- [停止词](#)
- [简单的字典](#)
- [同义词词典](#)
- [同义词词典](#)
- [Ispell字典](#)
- [SnowBall字典](#)

## 关于文本搜索词典

字典用于消除在搜索中不应考虑的单词（停用单词），并对单词进行标准化，以使相同单词的不同派生形式匹配。成功标准化的单词称为词位。除了提高搜索质量之外，停止词的标准化和删除减少了文档的tsvector 表示的大小，从而提高了性能。规范化并不总是具有语言意义，通常依赖于应用语义。

标准化的一些例子：

- 语法 - Ispell词典试图将输入词减少到标准化形式; 词干词典删除单词结尾
- 可以规范化URL位置以使URL等效匹配：
  - <http://www.pgsql.ru/db/mw/index.html>
  - <http://www.pgsql.ru/db/mw/>
  - <http://www.pgsql.ru/db/.../db/mw/index.html>
- 颜色名称可以用它们的十六进制值替换，例如红色，绿色，蓝色，品红色->FF0000,00FF00,0000FF, FF00FF
- 如果索引数字，我们可以删除一些小数位以减少可能数字的范围，因此例如3.14159265359,3.1415926,3.14在归一化后将是相同的，如果小数点后只保留两位数。

字典是一个接受token作为输入并返回的程序：A dictionary is a program that accepts a token as input and returns:

- 如果字典已知输入token（注意一个token可以产生多个词位），则为词位数组
- 设置了TSL\_FILTER标志的单个词位，用新token替换原始token以传递给后续字典（执行此操作的字典称为过滤字典）
- 如果字典知道token，则为空数组，但它是停止词
- 如果字典无法识别输入token，则为NULL

Greenplum数据库为许多语言提供预定义的词典。还有一些预定义模板可用于创建具有自定义参数的新词典。每个预定义的字典模板如下所述。如果没有适合的现有模板，则可以创建新模板；有关示例，请参阅Greenplum数据库分发的contrib/ 区域。

文本搜索配置将解析器与一组字典绑定在一起，以处理解析器的输出token。对于解析器可以返回的每个token类型，配置指定单独的字典列表。当解析器找到该类型的token时，依次查询列表中的每个字典，直到某些字典将其识别为已知字。如果它被识别为停止词，或者没有字典识别该token，则它将被丢弃并且不被索引或搜索。通常，返回non-NULL 输出的第一个字典确定结果，并且不查阅任何剩余的字典；但是过滤字典可以用修改后的单词替换给定单词，然后将其传递给后续词典。

配置字典列表的一般规则是首先放置最窄，最具体的字典，然后是更一般的字典，最后是一个非常通用的字典，如Snowball stemmer或simple，它可以识别所有内容。例如，对于特定于天文学的搜索（astro\_en 配置），可以将token类型asciivord（ASCII字）绑定到天文术语的同义词词典，通用英语词典和Snowball英语词干分析器：

```
ALTER TEXT SEARCH CONFIGURATION astro_en
 ADD MAPPING FOR asciivord WITH astrosyn, english_ispell,
 english_stem;
```

过滤字典可以放在列表中的任何位置，除非它没有用处。过滤词典对于部分规范化单词以简化后续词典的任务非常有用。例如，过滤字典可用于从重音字母中删除重音，如非重音模块所做的那样。

## 停止词

停止词是非常常见的词，几乎出现在每个文档中，并且没有区分值。因此，在全文搜索的上下文中可以忽略它们。例如，每个英文文本都包含像a和the这样的单词，因此将它们存储在索引中是没用的。但是，停止词会影响tsvector 中的位置，从而影响排名：

```
SELECT to_tsvector('english','in the list of stop words');
 to_tsvector

'list':3 'stop':5 'word':6
```

丢失的位置1,2,4是因为停止词。计算带有和不带停止词的文档的等级是完全不同的：

```
SELECT ts_rank_cd (to_tsvector('english','in the list of stop words'),
to_tsquery('list & stop'));
 ts_rank_cd

0.05

SELECT ts_rank_cd (to_tsvector('english','list stop words'),
to_tsquery('list & stop'));
 ts_rank_cd

0.1
```

它取决于特定字典如何处理停止词。例如，ispell 词典首先将单词标准化，然后查看停止词列表，而Snowball 词干表首先检查停止词列表。不同行为的原因是试图降低噪音。

# 简单的字典

简单字典模板通过将输入token转换为小写字母并针对停用字文件进行检查来进行操作。如果在文件中找到它，则返回一个空数组，导致该token被丢弃。如果不是，则将该词的小写形式作为标准化词位返回。或者，可以将字典配置为将非停止词报告为无法识别，允许将它们传递到列表中的下一个词典。

以下是使用简单模板的字典定义示例：

```
CREATE TEXT SEARCH DICTIONARY public.simple_dict (
 TEMPLATE = pg_catalog.simple,
 STOPWORDS = english
);
```

这里，english 是停止词文件的基本名称。该文件的全名为\$SHAREDIR/tsearch\_data/english.stop，其中\$SHAREDIR表示Greenplum数据库安装的共享数据目录，通常为/usr/local/greenplum-db-<version>/share/postgresql（使用pg\_config --sharedir 来确定它，如果您不确定。文件格式只是一个单词列表，每行一个。空行和尾随空格被忽略，大写折叠为小写，但没有对文件内容进行其他处理。often

现在我们可以测试我们的字典：

```
SELECT ts_lexize('public.simple_dict','YeS');
ts_lexize

{yes}

SELECT ts_lexize('public.simple_dict','The');
ts_lexize

{}
```

如果在停止词文件中找不到，我们也可以选择返回NULL而不是小写的词。通过将字典的Accept 参数设置为false来选择此行为。继续这个例子：

```
ALTER TEXT SEARCH DICTIONARY public.simple_dict (Accept = false);

SELECT ts_lexize('public.simple_dict','YeS');
ts_lexize

{yes}

SELECT ts_lexize('public.simple_dict','The');
ts_lexize

{}
```

使用Accept = true 的默认设置，只在字典列表的末尾放置一个简单的字典是有用的，因为它永远不会将任何token传递给后面的字典。相反，Accept = false 仅在至少有一个后续字典时才有用。

## CAUTION:

大多数类型的词典都依赖于配置文件，例如停止词的文件。这些文件必须以UTF-8编码存储。如果它们被读入服务器，它们将被转换为实际的数据库编码（如果不同）。

## CAUTION:

通常，数据库会话将首次在会话中使用时读取字典配置文件一次。如果修改配置

文件并希望强制现有会话获取新内容，请在字典上发出ALTER TEXT SEARCH DICTIONARY命令。这可以是“虚拟”更新，实际上不会更改任何参数值。

## 同义词词典

此词典模板用于创建用同义词替换单词的词典。不支持短语 - 使用同义词库模板（[同义词词典](#)）。同义词词典可用于克服语言问题，例如，防止英语词干分词将“Paris”一词减少为“pari”。在同义词词典中有一个Paris paris线就足够了，并把它放在english\_stem 字典之前。例如：

```
SELECT * FROM ts_debug('english', 'Paris');
 alias | description | token | dictionaries | dictionary |
lexemes
-----+-----+-----+-----+-----+
+-----+
 asciiword | Word, all ASCII | Paris | {english_stem} | english_stem | {pari}

CREATE TEXT SEARCH DICTIONARY my_synonym (
 TEMPLATE = synonym,
 SYNONYMS = my_synonyms
);

ALTER TEXT SEARCH CONFIGURATION english
 ALTER MAPPING FOR asciiword
 WITH my_synonym, english_stem;

SELECT * FROM ts_debug('english', 'Paris');
 alias | description | token | dictionaries |
dictionary | lexemes
-----+-----+-----+-----+-----+
+-----+
 asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} | {my_synonym} | {paris}
```

同义词模板所需的唯一参数是SYNONYMS 它是其配置文件的基本名称 - 上例中的my\_synonyms 。该文件的全名  
为\$SHAREDIR/tsearch\_data/my\_synonyms.syn （其中\$SHAREDIR表示Greenplum数据库安装的共享数据目录）。文件格式只是每个要替换的单词一行，单词后跟其同义词，用空格分隔。空行和尾随空格将被忽略。

同义词模板还有一个可选参数CaseSensitive ，默认为false 。当CaseSensitive 为false 时，同义词文件中的单词将折叠为小写，输入token也是如此。如果是true ，则单词和token不会折叠为小写，而是按原样进行比较。

星号 (\*) 可以放在配置文件中同义词的末尾。这表示同义词是前缀。当在to\_tsvector() 中使用该条目时，将忽略星号，但在to\_tsquery() 中使用该条目时，结果将是具有前缀匹配标记的查询项（请参阅[解析查询](#)）。例如，假设我们在\$SHAREDIR/tsearch\_data/synonym\_sample.syn 中有这些条目：

```
postgres pgsql postgresql pgsql postgre pgsql
gogle googl
indices index*
```

然后我们将得到这些结果：

```
mydb=# CREATE TEXT SEARCH DICTIONARY syn (template=synonym,
```

```

synonyms='synonym_sample');
mydb=# SELECT ts_lexize('syn', 'indices')
ts_lexize

{index}
(1 row)

mydb=# CREATE TEXT SEARCH CONFIGURATION tst (copy=simple);
mydb=# ALTER TEXT SEARCH CONFIGURATION tst ALTER MAPPING FOR asciiword
WITH syn;
mydb=# SELECT to_tsvector('tst','indices');
to_tsvector

'index':1
(1 row)

mydb=# SELECT to_tsquery('tst','indices');
to_tsquery

'index':*
(1 row)

mydb=# SELECT 'indexes are very useful'::tsvector;
tsvector

'are' 'indexes' 'useful' 'very'
(1 row)

mydb=# SELECT 'indexes are very useful'::tsvector @@ to_tsquery('tst','indices');
?column?

t
(1 row)

```

## 同义词词典

同义词词典（有时缩写为TZ）是一组词，包括有关词和短语关系的信息，即更广泛的术语（BT），更窄的术语（NT），首选术语，非首选术语，相关术语，等等。

基本上，同义词词典将所有非首选术语替换为一个优选术语，并且可选地，也保留用于索引的原始术语。Greenplum数据库的同义词词典的当前实现是同义词词典的扩展，增加了词组支持。同义词词典需要以下格式的配置文件：

```

this is a comment
sample word(s) : indexed word(s)
more sample word(s) : more indexed word(s)
...

```

其中冒号（:）符号充当短语及其替换之间的分隔符。

同义词词典使用子词典（在词典的配置中指定）来在检查词组匹配之前规范化输入文本。只能选择一个子字典。如果子字典无法识别单词，则会报告错误。在这种情况下，您应该删除单词的使用或教它关于它的子字典。您可以在索引单词的开头放置一个星号（\*）以跳过对其应用子字典，但所有样本单词必须为子字典所知。

如果存在多个与输入匹配的短语，则同义词词典选择最长匹配，并且通过使用最后定义来断开关系。

无法指定子字典识别的特定停止词; 改为使用? 标记任何停止词可以出现的位置。  
例如, 假设a和the 根据子字典是停止词:

```
? one ? two : ssws
```

匹配a one the two 和the one a two ; 两者都将被ssws取代。

由于同义词词典具有识别短语的能力, 因此它必须记住其状态并与解析器交互。同义词词典使用这些分配来检查它是否应该处理下一个单词或停止累积。必须仔细配置同义词词典。例如, 如果分配同义词词典仅处理asciword token, 那么像one 7 一样的同义词词典定义将不起作用, 因为token类型uint 未分配给同义词词典。

#### CAUTION:

在索引期间使用同义词, 因此同义词词典参数的任何更改都需要重新索引。对于大多数其他字典类型, 添加或删除停止词等小的更改不会强制重新编制索引。

#### 同义词库配置

要定义新的同义词词典, 请使用thesaurus 模板。例如:

```
CREATE TEXT SEARCH DICTIONARY thesaurus_simple (
 TEMPLATE = thesaurus,
 DictFile = mythesaurus,
 Dictionary = pg_catalog.english_stem
);
```

其中:

- thesaurus\_simple是新的词典名称
- mythesaurus是同义词库配置文件的基本名称。 (它的全名是\$SHAREDIR/tsearch\_data/mythesaurus.ths, 其中\$SHAREDIR表示安装共享数据目录。)
- pg\_catalog.english\_stem是用于词库规范化的子字典 (这里是一个Snowball英语词干分析器)。 请注意, subdictionary将具有自己的配置 (例如, 停止词), 此处未显示。

现在可以将同义词词典thesaurus\_simple 绑定到配置中的所需token类型, 例如:

```
ALTER TEXT SEARCH CONFIGURATION russian
 ALTER MAPPING FOR asciword, asciihword, hword_asciipart
 WITH thesaurus_simple;
```

#### 词库示例

考虑一个简单的天文词库thesaurus\_astro , 其中包含一些天文单词组合:

```
supernovae stars : sn
crab nebulae : crab
```

下面我们创建一个字典并将一些token类型绑定到天文词库和英语词干分析器:

```
CREATE TEXT SEARCH DICTIONARY thesaurus_astro (
 TEMPLATE = thesaurus,
```

```

 DictFile = thesaurus_astro,
 Dictionary = english_stem
);

ALTER TEXT SEARCH CONFIGURATION russian
 ALTER MAPPING FOR asciiword, asciihword, hword_asciipart
 WITH thesaurus_astro, english_stem;

```

现在我们可以看到它是如何工作的。 `ts_lexize` 对于测试同义词库并不是非常有用，因为它将其输入视为单个token。 相反，我们可以使用`plainto_tsquery` 和`to_tsvector`，它们会将输入字符串分成多个token：

```

SELECT plainto_tsquery('supernova star');
plainto_tsquery

'sn'

SELECT to_tsvector('supernova star');
to_tsvector

'sn':1

```

原则上，如果引用参数，可以使用`to_tsquery`：

```

SELECT to_tsquery(''supernova star''');
to_tsquery

'sn'

```

请注意，`supernova star` 与`thesaurus_astro` 中的`supernovae stars` 相匹配，因为我们

们在同义词定义中指定了`english_stem` 提取器。 提取器去掉了e和s。

要索引原始短语以及替换，只需将其包含在定义的右侧部分：

```

supernovae stars : sn supernovae stars

SELECT plainto_tsquery('supernova star');
plainto_tsquery

'sn' & 'supernova' & 'star'

```

## Ispell字典

Ispell字典模板支持形态词典，可以将单词的许多不同语言形式归一化为相同的词位。 例如，英语Ispell字典可以匹配搜索项`bank` 的所有词形组合和变化，例如`banking`，`banked`，`banks`，`banks'` 和`bank's`。

标准Greenplum数据库分发不包含任何Ispell配置文件。 Ispell 提供大量语言的字典。 此外，还支持一些更现代的字典文件格式 - [MySpell](#) (OO < 2.0.1) 和[Hunspell](#) (OO >= 2.0.2)。 [OpenOffice Wiki](#) 上提供了大量字典。

要创建Ispell字典，请使用内置`ispell` 模板并指定几个参数：

```

CREATE TEXT SEARCH DICTIONARY english_ispell (
 TEMPLATE = ispell,
 DictFile = english,
 AffFile = english,

```

```
StopWords = english
);
```

这里，DictFile，AffFile 和StopWords 指定字典，词缀和停止词文件的基本名称。停止词文件具有与上述simple 词典类型相同的格式。此处未指定其他文件的格式，但可从上述网站获取。

Ispell字典通常识别一组有限的单词，因此应该跟着另一个更广泛的字典；例如，一个可以识别一切的Snowball字典。

Ispell词典支持分裂复合词；一个有用的功能。请注意，词缀文件应使用compoundwords controlled语句指定一个特殊标志，该语句标记可以参与复合词形成的词典单词：

```
compoundwords controlled z
```

以下是挪威语的一些示例：

```
SELECT ts_lexize('norwegian_ispell',
'overbuljongterningpakkemesterassistent');
{over,buljong,terning,pakk,mester,assistent}
SELECT ts_lexize('norwegian_ispell', 'sjokoladefabrikk');
{sjokoladefabrikk,sjokolade,fabrikk}
```

Note:

MySpell不支持复合词。 Hunspell对复合词有着复杂的支持。目前，Greenplum数据库仅实现了Hunspell的基本复合词操作。

## SnowBall词典

Snowball词典模板基于Martin Porter的一个项目，他是流行的Porter英语词干算法的发明者。Snowball现在为许多语言提供词干算法（有关更多信息，请参阅[Snowball站点](#)）。每种算法都理解如何将单词的常见变体形式减少到其语言中的基础或词干拼写。Snowball字典需要一个语言参数来识别要使用的词干分析器，并且可以选择指定一个停止词文件名，该名称提供要删除的单词列表。

（Greenplum数据库的标准停止词列表也由Snowball项目提供。）例如，有一个内置的定义等效于

```
CREATE TEXT SEARCH DICTIONARY english_stem (
 TEMPLATE = snowball,
 Language = english,
 StopWords = english
);
```

停止词文件格式与已经说明的相同。

Snowball字典可以识别所有内容，无论它是否能够简化单词，因此它应该放在字典列表的末尾。在任何其他字典之前使用它是没用的，因为token永远不会通过它传递到下一个字典。

**Parent topic:** [使用全文搜索](#)



## Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 定义数据库对象
  - 分布与倾斜
  - 插入, 更新, 和删除数据
  - 查询数据
    - 关于Greenplum的查询处理
    - 关于GPORCA
    - 定义查询
    - WITH查询 (公用表表达式)
    - 使用函数和操作符
    - 使用JSON数据
    - 使用XML数据
    - 使用全文搜索
      - 关于全文搜索
      - 在数据库表中搜索文本
      - 控制文本搜索
      - 文本搜索附加功能

本主题说明如何创建自定义文本搜索配置以处理文档和查询文本。

文本搜索配置指定将文档转换为`tsvector`所需的所有选项：用于将文本分解为`token`的解析器，以及用于将每个`token`转换为词位的字典。每次调用`to_tsvector`或`to_tsquery`都需要文本搜索配置来执行其处理。配置参数`default_text_search_config` 指定默认配置的名称，如果省略显式配置参数，则默认配置是文本搜索功能使用的名称。可以使用`gpconfig`命令行工具在`postgresql.conf`中设置，也可以使用`SET`命令为单个会话设置。

有几种预定义的文本搜索配置可供使用，您可以轻松创建自定义配置。为了便于管理文本搜索对象，可以使用一组SQL命令，并且有几个`psql`命令显示有关文本搜索对象的信息（[psql支持](#)）。

作为一个例子，我们将创建一个配置`pg`，首先复制内置的`english`配置：

```
CREATE TEXT SEARCH CONFIGURATION public.pg (COPY =
pg_catalog.english);
```

我们将使用PostgreSQL特定的同义词列表并将其存储在`$SHAREDIR/tsearch_data/pg_dict.syn`中。文件内容如下：

```
postgres pg
pgsql pg
postgresql pg
```

我们像这样定义同义词字典：

```
CREATE TEXT SEARCH DICTIONARY pg_dict (
 TEMPLATE = synonym,
 SYNONYMS = pg_dict
);
```

接下来我们注册Ispell字典`english_ispell`，它有自己的配置文件：

```
CREATE TEXT SEARCH DICTIONARY english_ispell (
 TEMPLATE = ispell,
 DictFile = english,
 AffFile = english,
 StopWords = english
```

```
) ;
```

现在我们可以在配置pg中设置单词的映射：

```
ALTER TEXT SEARCH CONFIGURATION pg
 ALTER MAPPING FOR asciiword, asciihword,
 hword_asciipart,
 word, hword, hword_part
 WITH pg_dict, english_ispell, english_stem;
```

我们选择不索引或搜索内置配置处理的某些token类型：

```
ALTER TEXT SEARCH CONFIGURATION pg
 DROP MAPPING FOR email, url, url_path, sfloat, float;
```

现在我们可以测试我们的配置：

```
SELECT * FROM ts_debug('public.pg', '
PostgreSQL, the highly scalable, SQL compliant, open source
object-relational
database management system, is now undergoing beta testing
of the next
version of our software.
');
```

下一步是将会话设置为使用在public schema中创建的新配置：

```
=> \dF
 List of text search configurations
 Schema | Name | Description
-----+-----+-----
 public | pg |
SET default_text_search_config = 'public.pg';
SET

SHOW default_text_search_config;
default_text_search_config

public.pg
```

**Parent topic:** 使用全文搜索

## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

关于Greenplum的查询  
处理

## □ 关于GPORCA

## 定义查询

WITH查询 (公用表表  
达式)

## 使用函数和操作符

## 使用JSON数据

## 使用XML数据

## □ 使用全文搜索

## 关于全文搜索

在数据库表中搜索文  
本

## 控制文本搜索

## 文本搜索附加功能

本主题介绍可用于测试和调试搜索配置或配置中指定的单个解析器和词典的Greenplum数据库函数。

自定义文本搜索配置的行为很容易变得混乱。本节中描述的功能对于测试文本搜索对象非常有用。您可以单独测试完整配置或测试解析器和词典。

本节包含以下子主题：

- [配置测试](#)
- [解析器测试](#)
- [字典测试](#)

## 配置测试

函数`ts_debug`允许轻松测试文本搜索配置。

```
ts_debug([config regconfig,] document text,
 OUT alias text,
 OUT description text,
 OUT token text,
 OUT dictionaries regdictionary[],
 OUT dictionary regdictionary,
 OUT lexemes text[])
 returns setof record
```

`ts_debug`显示有关解析器生成并由配置的字典处理的每个文档`token`的信息。它使用`config`指定的配置，如果省略该参数，则使用`default_text_search_config`。

`ts_debug`为解析器在文本中标识的每个`token`返回一行。返回的列是

- `alias text` — `token`类型的简称
- `description text` — `token`类型的描述
- `token text` — `token`文本
- `dictionaries regdictionary[]` — 由此`token`类型的配置选择的词典
- `dictionary regdictionary` — 识别`token`的字典，如果没有，则为NULL
- `lexemes text[]` — 识别`token`的字典产生的词位，如果没有，则

为NULL; 空数组( { } )表示它被识别为停止词

这是一个简单的例子：

```
SELECT * FROM ts_debug('english','a fat cat sat on a mat - it ate a fat rats');
 alias | description | token | dictionaries |
 dictionary | lexemes
-----+-----+-----+-----+-----+
-----+-----+
 asciiword | Word, all ASCII | a | {english_stem} |
 english_stem | {} |
 blank | Space symbols | | {} |
 |
 asciiword | Word, all ASCII | fat | {english_stem} |
 english_stem | {fat} |
 blank | Space symbols | | {} |
 |
 asciiword | Word, all ASCII | cat | {english_stem} |
 english_stem | {cat} |
 blank | Space symbols | | {} |
 |
 asciiword | Word, all ASCII | sat | {english_stem} |
 english_stem | {sat} |
 blank | Space symbols | | {} |
 |
 asciiword | Word, all ASCII | on | {english_stem} |
 english_stem | {} |
 blank | Space symbols | | {} |
 |
 asciiword | Word, all ASCII | a | {english_stem} |
 english_stem | {} |
 blank | Space symbols | | {} |
 |
 asciiword | Word, all ASCII | mat | {english_stem} |
 english_stem | {mat} |
 blank | Space symbols | | {} |
 |
 blank | Space symbols | - | {} |
 |
 asciiword | Word, all ASCII | it | {english_stem} |
 english_stem | {} |
 blank | Space symbols | | {} |
 |
 asciiword | Word, all ASCII | ate | {english_stem} |
 english_stem | {ate} |
 blank | Space symbols | | {} |
 |
 asciiword | Word, all ASCII | a | {english_stem} |
 english_stem | {} |
 blank | Space symbols | | {} |
 |
 asciiword | Word, all ASCII | fat | {english_stem} |
 english_stem | {fat} |
 blank | Space symbols | | {} |
 |
```

```
asciivord | Word, all ASCII | rats | {english_stem} |
english_stem | {rat}
```

为了进行更广泛的演示，我们首先为英语创建一个public.english配置和Ispell字典：

```
CREATE TEXT SEARCH CONFIGURATION public.english (COPY =
pg_catalog.english);

CREATE TEXT SEARCH DICTIONARY english_ispell (
 TEMPLATE = ispell,
 DictFile = english,
 AffFile = english,
 StopWords = english
);

ALTER TEXT SEARCH CONFIGURATION public.english
 ALTER MAPPING FOR asciivord WITH english_ispell,
 english_stem;
```

```
SELECT * FROM ts_debug('public.english', 'The Brightest
supernovaes');
 alias | description | token |
dictionaries | dictionary | lexemes
-----+-----+-----+-----+
-----+-----+-----+-----+
 asciivord | Word, all ASCII | The |
{english_ispell,english_stem} | english_ispell | {} |
 blank | Space symbols | | {} |
 | |
 asciivord | Word, all ASCII | Brightest |
{english_ispell,english_stem} | english_ispell | {bright} |
 blank | Space symbols | | {} |
 | |
 asciivord | Word, all ASCII | supernovaes |
{english_ispell,english_stem} | english_stem | |
 supernova|
```

在此示例中，解析器将单词Brightest识别为ASCII字（别名asciivord）。对于此token类型，字典列表是english\_ispell和english\_stem。这个词被english\_ispell识别，将其缩小为名词bright。english\_ispell词典中不知道supernovaes这个词，所以它被传递到下一个词典，幸运的是，它被识别出来（事实上，english\_stem是一个可以识别所有内容的Snowball词典；这就是为什么它放在词典列表的末尾）。

单词The被english\_ispell字典识别为停止词（[停止词](#)），不会被编入索引。这些空间也被丢弃，因为配置根本不为它们提供字典。

您可以通过显式指定要查看的列来减小输出的宽度：

```
SELECT alias, token, dictionary, lexemes FROM
ts_debug('public.english', 'The Brightest supernovaes');
 alias | token | dictionary | lexemes
-----+-----+-----+-----+
 asciiword | The | english_ispell | {}
 blank | | | |
 asciiword | Brightest | english_ispell | {bright}
 blank | | | |
 asciiword | supernovaes | english_stem | {supernova}
```

## 解析器测试

以下函数允许直接测试文本搜索解析器。

```
ts_parse(parser_name text, document text,
 OUT tokid integer, OUT token text) returns setof
record
ts_parse(parser_oid oid, document text,
 OUT tokid integer, OUT token text) returns setof
record
```

`ts_parse`解析给定文档并返回一系列记录，每个记录用于解析生成的每个token。每条记录都包含一个显示已分配token类型的`tokid`和一个`token`，是token的文本。例如：

```
SELECT * FROM ts_parse('default', '123 - a number');
 tokid | token
-----+-----
 22 | 123
 12 | -
 12 | -
 1 | a
 12 | -
 1 | number
```

```
ts_token_type(parser_name text, OUT tokid integer,
 OUT alias text, OUT description text) returns
setof record
ts_token_type(parser_oid oid, OUT tokid integer,
 OUT alias text, OUT description text) returns
setof record
```

`ts_token_type`返回一个表，该表描述指定解析器可识别的每种类型的token。对于每个token类型，该表给出了解析器用于标记

该token类型的整数tokid，在配置命令中命名token类型的alias以及简短description。例如：

```
SELECT * FROM ts_token_type('default');
 tokid | alias | description
-----+-----+-----+
 1 | asciiword | Word, all ASCII
 2 | word | Word, all letters
 3 | numword | Word, letters and digits
 4 | email | Email address
 5 | url | URL
 6 | host | Host
 7 | sfloat | Scientific notation
 8 | version | Version number
 9 | hword_numpart
and digits
 10 | hword_part | Hyphenated word part, letters
letters
 11 | hword_asciipart | Hyphenated word part, all ASCII
 12 | blank | Space symbols
 13 | tag | XML tag
 14 | protocol | Protocol head
 15 | numhword | Hyphenated word, letters and
digits
 16 | asciihword | Hyphenated word, all ASCII
 17 | hword | Hyphenated word, all letters
 18 | url_path | URL path
 19 | file | File or path name
 20 | float | Decimal notation
 21 | int | Signed integer
 22 | uint | Unsigned integer
 23 | entity | XML entity
```

## 字典测试

`ts_lexize`函数有助于字典测试。

```
ts_lexize(dictreg dictionary, token text) returns text[]
```

如果字典已知输入token，则`ts_lexize`返回一个词位数组；如果字典已知该token但是它是一个停止词，则返回一个空数组；如果它是一个未知单词，则返回NULL。

示例：

```
SELECT ts_lexize('english_stem', 'stars');
ts_lexize
```

```

{star}

SELECT ts_lexize('english_stem', 'a');
ts_lexize

{}
```

Note: `ts_lexize`函数需要单个token，而不是文本。这是一个令人困惑的案例：

```
SELECT ts_lexize('thesaurus_astro', 'supernovae stars') is
null;
?column?

t
```

同义词词典`thesaurus_astro`确实知道短语`supernovae stars`，但`ts_lexize`失败，因为它不解析输入文本但将其视为单个token。使用`plainto_tsquery`或`to_tsvector`来测试同义词词典，例如：

```
SELECT plainto_tsquery('supernovae stars');
plainto_tsquery

'sn'
```

**Parent topic:** [使用全文搜索](#)

## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

## □ 查询数据

关于Greenplum的查询  
处理

## □ 关于GPORCA

定义查询

WITH查询 (公用表表达式)

使用函数和操作符

使用JSON数据

使用XML数据

## □ 使用全文搜索

关于全文搜索

在数据库表中搜索文  
本

控制文本搜索

文本搜索附加功能

本主题描述并比较用于全文搜索的Greenplum数据库索引类型。

有两种索引可用于加速全文搜索。索引对于全文搜索不是强制性的，但是在定期搜索列的情况下，通常需要索引。

`CREATE INDEX name ON table USING gist(column);`创建基于GiST (广义搜索树) 的索引。该列可以是`tsvector`或`tsquery`类型。`CREATE INDEX name ON table USING gin(column);`创建基于GIN (广义倒置索引) 的索引。该列必须是`tsvector`类型。

两种索引类型之间存在显着的性能差异，因此了解它们的特征非常重要。

GiST索引是有损的，这意味着索引可能产生错误匹配，并且有必要检查实际的表行以消除这种错误匹配。（Greenplum数据库在需要时自动执行此操作。）GiST索引是有损的，因为每个文档在索引中由固定长度签名表示。通过将每个字散列为n比特串中的单个比特来生成签名，所有这些比特一起做OR产生n比特文档签名。当两个单词散列到相同的位位置时，将存在错误匹配。如果查询中的所有单词都匹配（真或假），则必须检索表行以查看匹配是否正确。

由于错误匹配导致不必要的表记录提取，因此有损压缩会导致性能下降。由于对表记录的随机访问速度很慢，因此限制了GiST索引的有用性。错误匹配的可能性取决于几个因素，特别是唯一字的数量，因此建议使用字典来减少此数字。

对于标准查询，GIN索引不是有损的，但它们的性能取决于唯一字数量的对数。（但是，GIN索引只存储`tsvector`值的单词（词位），而不存储它们的权重标签。因此，当使用涉及权重的查询时，需要重新检查表行。）

在选择要使用的索引类型，GiST或GIN时，请考虑以下性能差异：

- GIN索引查找速度比GiST快三倍
- GIN索引的构建时间比GiST长大约三倍
- GIN索引的更新速度比GiST索引要慢，但如果禁用快速更新支持，速度会慢10倍（有关详细信息，请参阅PostgreSQL文档中的[GIN快速更新技术](#)）

- GIN索引比GiST索引大两到三倍

根据经验，GIN索引最适合静态数据，因为查找速度更快。对于动态数据，GiST索引的更新速度更快。具体来说，GiST索引非常适合动态数据，如果唯一单词（词位）的数量低于100,000，则快速，而GIN索引将更好地处理100,000+词位，但更新速度较慢。

请注意，通常可以通过增加[maintenance\\_work\\_mem](#) 来提高GIN索引构建时间，而GiST索引构建时间对该参数不敏感。

对大型集合进行分区以及正确使用GiST和GIN索引可以通过在线更新实现非常快速的搜索。可以使用表继承在数据库级别进行分区，也可以使用[dblink](#)在服务器上分发文档和收集搜索结果。后者是可能的，因为排名功能仅使用本地信息。

**Parent topic:** [使用全文搜索](#)

## Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 定义数据库对象
    - 分布与倾斜
  - 插入, 更新, 和删除数据
- 查询数据
  - 关于Greenplum的查询处理
  - 关于GPORCA
  - 定义查询
  - WITH查询 (公用表表达式)
  - 使用函数和操作符
  - 使用JSON数据
  - 使用XML数据
- 使用全文搜索
  - 关于全文搜索
  - 在数据库表中搜索文本
  - 控制文本搜索
  - 文本搜索附加功能

psql命令行实用程序提供了一个元命令, 用于显示有关Greenplum数据库全文搜索配置的信息。

可以使用一组命令在psql中获取有关文本搜索配置对象的信息:

```
\dF{d,p,t}[+] [PATTERN]
```

可选 + 生成更多细节。

可选参数PATTERN可以是文本搜索对象的名称, 指定schema是可选的。如果省略PATTERN, 则将显示有关所有可见对象的信息。PATTERN可以是正则表达式, 可以为schema和对象名称提供单独的模式。以下示例说明了这一点:

```
=> \dF *fulltext*
 List of text search configurations
Schema | Name | Description
-----+-----+-----+
 public | fulltext_cfg |
```

```
=> \dF *.fulltext*
 List of text search configurations
Schema | Name | Description
-----+-----+-----+
 fulltext | fulltext_cfg |
```

可用的命令是:

```
\dF[+] [PATTERN]
```

列出文本搜索配置 (添加 + 以获取更多详细信息)。

```
=> \dF russian
 List of text search configurations
Schema | Name | Description
-----+-----+-----+
 pg_catalog | russian | configuration for russian
language
```

```
=> \dF+ russian
Text search configuration "pg_catalog.russian"
Parser: "pg_catalog.default"
Token | Dictionaries
```



|                 |              |
|-----------------|--------------|
| asciihword      | english_stem |
| asciicword      | english_stem |
| email           | simple       |
| file            | simple       |
| float           | simple       |
| host            | simple       |
| hword           | russian_stem |
| hword_asciipart | english_stem |
| hword_numpart   | simple       |
| hword_part      | russian_stem |
| int             | simple       |
| numhword        | simple       |
| numword         | simple       |
| sfloat          | simple       |
| uint            | simple       |
| url             | simple       |
| url_path        | simple       |
| version         | simple       |
| word            | russian_stem |

\dFd[+] [PATTERN]

列出文本搜索词典（添加 + 以获取更多详细信息）。

| List of text search dictionaries |                 |                                          |
|----------------------------------|-----------------|------------------------------------------|
| Schema                           | Name            | Description                              |
| pg_catalog                       | danish_stem     | snowball stemmer for danish language     |
| pg_catalog                       | dutch_stem      | snowball stemmer for dutch language      |
| pg_catalog                       | english_stem    | snowball stemmer for english language    |
| pg_catalog                       | finnish_stem    | snowball stemmer for finnish language    |
| pg_catalog                       | french_stem     | snowball stemmer for french language     |
| pg_catalog                       | german_stem     | snowball stemmer for german language     |
| pg_catalog                       | hungarian_stem  | snowball stemmer for hungarian language  |
| pg_catalog                       | italian_stem    | snowball stemmer for italian language    |
| pg_catalog                       | norwegian_stem  | snowball stemmer for norwegian language  |
| pg_catalog                       | portuguese_stem | snowball stemmer for portuguese language |
| pg_catalog                       | romanian_stem   | snowball stemmer for romanian language   |
| pg_catalog                       | russian_stem    | snowball stemmer for                     |

```

russian language
 pg_catalog | simple | simple dictionary:
just lower case and check for stopword
 pg_catalog | spanish_stem | snowball stemmer for
spanish language
 pg_catalog | swedish_stem | snowball stemmer for
swedish language
 pg_catalog | turkish_stem | snowball stemmer for
turkish language

```

## \dFp[+] [PATTERN]

列出文本搜索解析器（添加 + 以获取更多详细信息）。

```

=> \dFp
 List of text search parsers
 Schema | Name | Description
-----+-----+-----
 pg_catalog | default | default word parser
=> \dFp+
 Text search parser "pg_catalog.default"
 Method | Function | Description
-----+-----+-----
 Start parse | prsd_start |
 Get next token | prsd_nexttoken |
 End parse | prsd_end |
 Get headline | prsd_headline |
 Get token types | prsd_lextype |

 Token types for parser "pg_catalog.default"
 Token name | Description
-----+-----
 asciihword | Hyphenated word, all ASCII
 asciiword | Word, all ASCII
 blank | Space symbols
 email | Email address
 entity | XML entity
 file | File or path name
 float | Decimal notation
 host | Host
 hword | Hyphenated word, all letters
 hword_asciipart | Hyphenated word part, all ASCII
 hword_numpart | Hyphenated word part, letters and
 digits
 hword_part | Hyphenated word part, all letters
 int | Signed integer
 numhword | Hyphenated word, letters and digits
 numword | Word, letters and digits
 protocol | Protocol head
 sfloat | Scientific notation
 tag | XML tag
 uint | Unsigned integer
 url | URL
 url_path | URL path
 version | Version number

```

```
word | Word, all letters
(23 rows)
```

\dFt [+] [PATTERN]

列出文本搜索模板（添加 + 以获取更多详细信息）。

```
=> \dFt List of text search
templates
Schema | Name |
Description
-----+-----+-----

pg_catalog | ispell | ispell dictionary
pg_catalog | simple | simple dictionary: just
lower case and check for stopword
pg_catalog | snowball | snowball stemmer
pg_catalog | synonym | synonym dictionary: replace
word by its synonym
pg_catalog | thesaurus | thesaurus dictionary: phrase
by phrase substitution
```

**Parent topic:** 使用全文搜索

## Greenplum数据库® 6.0文档

## □ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

## □ 查询数据

关于Greenplum的查询  
处理

□ 关于GPORCA

定义查询

WITH查询 (公用表表  
达式)

使用函数和操作符

使用JSON数据

使用XML数据

## □ 使用全文搜索

关于全文搜索

在数据库表中搜索文  
本

控制文本搜索

文本搜索附加功能

本主题列出了Greenplum数据库全文搜索对象的限制和最大值。

Greenplum数据库的文本搜索功能目前的局限性是：

- Greenplum数据库表的分发键不支持`tsvector`和`tsquery`类型
- 每个词位的长度必须小于2K字节
- `tsvector`的长度 (词位 + 位置) 必须小于1M
- 词位数必须小于 $2^{64}$
- `tsvector`中的位置值必须大于0且不大于16,383
- 每个词位不超过256个位置
- `tsquery`中的节点数 (词位 + 运算符) 必须小于32,768

为了比较, PostgreSQL 8.1文档包含10,441个不同单词, 总共335,420个单词, 最常用的单词“`postgresql`”在655个文档中被提及6,127次。

另一个例子 - PostgreSQL邮件列表存档包含910,989个不同单词, 其中包含57,491,343个词位, 共461,020条消息。

**Parent topic:** [使用全文搜索](#)

## Greenplum数据库® 6.0文档

- 管理员指南
- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象
- 分布与倾斜
- 插入, 更新, 和删除数据
- 查询数据
- 使用外部数据
- 装载和卸载数据
- 性能管理
- 管理性能
- 性能问题的常见原因
- Greenplum数据库内存总览
- 管理资源
- 用资源组进行工作负载管理

## 使用资源队列

## 检修性能问题

## Greenplum database 5管理员指南

## Greenplum database 4管理员指南

## FAQ

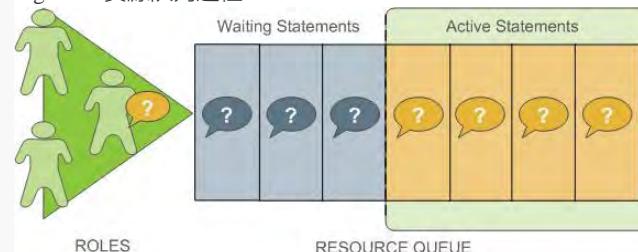
使用Greenplum数据库的工作负载管理根据业务需求对查询按照优先权分配资源，并且在资源不可用时阻止查询开始。

资源队列是用来管理Greenplum数据库系统中并行的程度的工具。你可以使用CREATE RESOURCE QUEUE SQL声明创建资源队列这一数据库对象，你可以用资源队列来管理可能并行执行的活动查询，控制它们的内存使用量，以及队列中的相对排序。资源队列同时也可以防止查询消耗太多资源进而降低系统性能。

每个数据库角色都与一个单个资源队列关联；多个角色可以共享同一个资源队列，使用RESOURCE QUEUE 中的CREATE ROLE或ALTER ROLE来将角色分配到资源队列中。如果没有指定资源队列，则角色将会自动被关联到默认资源队列pg\_default。

当用户提交了一个要执行的查询时，资源队列会根据其限制对查询进行评估，若该查询所需的资源没有超过限制，它将会立即执行；若超过了限制（例如，如果目前所有活动声明槽位都已被占用），该查询则必须等到队列资源有空闲时才能执行。对查询的评估将遵从先进先出的原则。如果启用了查询优先权，则将会预估系统当前的负载并根据查询优先权（参考[优先权如何起作用](#)）进行资源分配。拥有SUPERUSER属性的角色将会不受资源队列限制的限制，无论何时，超级用户的查询都会无视其所在资源队列的限制并立即执行。

Figure 1. 资源队列进程



资源队列会将资源需求相似的查询合并为Class，管理员应该根据组织中不同种类的负载创建资源队列。例如，参考以下不同的服务级别，用户需要根据其所对应的的查询的Class创建资源组：

- ETL查询
- 报告查询
- 执行查询

拥有以下特性的资源组：

#### MEMORY\_LIMIT

队列（每个Segment）中所有查询所使用的的内存的量。例如，对查询设置2GB的MEMORY\_LIMIT，这样每个Segment里的ETL查询最多使用2GB的内存。

#### ACTIVE\_STATEMENTS

队列中槽位的数量；一个队列中最大可并行数，当所有槽位都占用时，新的查询必须等待。默认每个查询使用等量的内存。  
例如，pg\_default资源组的ACTIVE\_STATEMENTS为20。

#### PRIORITY

查询使用的相对CPU使用量，可以设置为以下级别：LOW、MEDIUM、HIGH和MAX。默认级别是MEDIUM，查询优先权的机制会监控系统中所有正在运行的查询的CPU使用量，并根据其优先权级别来调整其CPU使用量。例如，你可以为执行资源队列设置MAX优先权，为其他查询设置MEDIUM优先权，确保执行查询可以获得比较多的CPU资源。

#### MAX\_COST

查询计划消耗限制。

Greenplum数据优化器会为每个查询分配数字型消耗，如果该消耗超过了资源队列的MAX\_COST的值，该查询就会被拒绝。

Note: GPORCA和Postgres查询优化器使用不同的查询消耗模型，因此同一个查询可能会得出不同的消耗，Greenplum数据库资源队列资源管理模式不会采用GPORCA或Postgres优化器得出的消耗，而是使用根据优化器直接返还的消耗

来限制查询。

当基于资源队列的资源管理系统激活时，会使用MEMORY\_LIMIT和ACTIVE\_STATEMENTS限制资源，而不是配置基于消耗的限制。因为即便是使用GPORCA的时候，Greenplum数据库仍然可能会在特定查询中使用Postgres查询优化器从而导致未知的结果。

Greenplum数据库系统默认配置有一个默认资源队列，名为pg\_default，pg\_default资源队列的ACTIVE\_STATEMENTS设置数值为20，没有MEMORY\_LIMIT和MAX\_COST，PRIORITY为中。这意味着所有查询都会被接受并立即执行，没有优先权划分和内存限制；但是并行的查询最多是20个。

一个资源队列所允许的并行查询的数量由是否有设置MEMORY\_LIMIT决定：

- 如果一个资源队列没有设置MEMORY\_LIMIT的话，每个资源所分配到的内存大小就是statement\_mem的服务器配置参数，一个资源队列的可用内存大小是根据statement\_mem和ACTIVE\_STATEMENTS的计算结果。
- 当资源队列有设置MEMORY\_LIMIT时，资源队列中可以并行执行的查询数将会受到该队列的可用内存限制。

提交到系统中的查询将会被分配一定量的内存，且系统会对其生成一个查询计划树。计划树的每个节点都是运算符，例如排序连接和哈希连接。每个运算符都是单独的执行线程，且被分配到了总陈述说明内存中的一部分（至少100KB）。如果计划中含有大量的运算符，运算符所需的最小内存大小可能会超过可用内存，则该查询会因内存不足而被拒绝。运算符会确定能否使用分配到的内存量完成任务，否则就会采取spill to disk的处理方式。这种分配和控制每个运算符使用的内存大小的机制被称为内存配额。

并不是所有提交到资源队列的SQL声明都会受到队列限制的评估，默认只会评估SELECT、SELECT INTO、CREATE TABLE AS SELECT和DECLARE CURSOR声明。如果服务器配置的resource\_select\_only参数为off，那么INSERT、UPDATE和DELETE声明也会受评估。

与此同时，在执行EXPLAIN ANALYZE命令期间跑的SQL声明也会被资源队列排除。

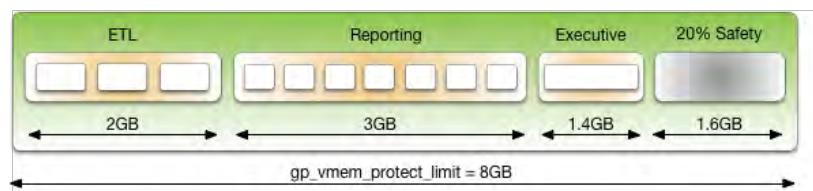
**Parent topic:** [管理资源](#)

## 资源队列示例

默认资源队列pg\_default允许最多20个活动查询平分内存以并行执行。一般来说这是不足以满足生产环境下的资源需求的。为了确保系统能够满足性能期望，用户可以定义查询的组并对其进行分配拥有相应的并行量、内存以及CPU的资源队列。

如下展示了一个示例Greenplum数据库系统（gp\_vmem\_protect\_limit为8GB）的资源队列的配置：

Figure 2. 资源队列配置示例



该示例有三个不同属性和服务等级协议的组，各个有对应的资源队列。其中预留有一部分内存作为冗余。

| 资源队列名 | 活动声明 | 内存限制  | 每个查询的内存 |
|-------|------|-------|---------|
| ETL   | 3    | 2GB   | 667MB   |
| 报告    | 7    | 3GB   | 429MB   |
| 执行    | 1    | 1.4GB | 1.4GB   |

队列所分配的内存总量为6.4GB，也就是说gp\_vmem\_protect\_limit服务器配置参数所定义的总Segment内存的80%。剩余的20%是为了那些可能会需要更多内存的运算符和查询所预留的。

关于命令语法帮助和详细参考信息请见Greenplum数据库参考指南中的CREATE RESOURCE QUEUE和CREATE / ALTER ROLE语句。

## 内存限制如何工作

一个资源队列上的MEMORY\_LIMIT为一个Segment实例设置通过该队列提交的所有活动查询可以消耗的最大内存量。拨给一个查询的内存量是队列内存限制除以活动语句限制（将内存限制与基于语句的队列而不是基于代价的队列）。例如，如果一个队列的内存限制是2000MB而活动语句限制是10，每个通过该队列提交的查询会默认拿到200MB内存。可以以每个查询为基础使用statement\_mem服务器配置参数覆盖默认的内存分配（最高到队列内存限制）。一旦一个查询开始执行，它会在队列中保持分拨给它的内存直至完成，即便在执行中它实际消耗的内存比分配到的内存少也是如此。

用户可以使用statement\_mem服务器配置参数来覆盖当前资源队列设置的内存限制。在会话级别，用户可以增加statement\_mem，最高到资源队列的MEMORY\_LIMIT。这将允许个别查询使用分配给整个队列的所有内存而不影响其他资源队列。

statement\_mem的值会被max\_statement\_mem配置参数（是一个超级用户参数）覆盖。对于一个设置有MEMORY\_LIMIT的资源队列中的查询，statement\_mem的最大值是min(MEMORY\_LIMIT, max\_statement\_mem)。当一个查询被允许进入时，分配给它的内存会被从MEMORY\_LIMIT中减去。如果MEMORY\_LIMIT被耗尽，同一个资源队列中的新查询必须等待。即使ACTIVE\_STATEMENTS还没有达到时也会发生这种事情。注意只有当statement\_mem被用来覆盖资源队列分配的内存时才会发生这种情况。

例如，考虑一个名为adhoc的队列，它有下列设置：

- MEMORY\_LIMIT 为 1.5GB
- ACTIVE\_STATEMENTS 为 3

默认每个被提交到队列的语句会被分配500MB内存。现在考虑下列一系列事件：

1. 用户ADHOC\_1提交查询Q1，并且把STATEMENT\_MEM覆盖为800MB。Q1语句被准许进入系统。
2. 用户ADHOC\_2提交查询Q2，使用默认的500MB。
3. 当Q1和Q2仍在运行时，用户ADHOC3提交查询Q3，使用默认的500MB。

查询Q1和Q2已经用掉了队列的1500MB中的1300MB。因此，Q3在能运行前必须等待Q1或Q2完成。

如果在一个队列上没有设置MEMORY\_LIMIT，查询都被准许进入，直到所有的ACTIVE\_STATEMENTS槽被用尽，并且每个查询可以设置一个任意高的statement\_mem这可能导致资源队列使用无限量的内存。

更多有关在资源队列上配置内存限制以及其他内存利用控制的信息，请见[创建带有内存限制的队列](#)。

## statement\_mem和低内存查询

经实践发现设置了低statement\_mem参数（比如1-3MB的范围内）的低内存需求的查询会有更好的性能表现。在单个查询的基础上使用statement\_mem服务器配置参数来覆盖队列的设置。例如：

```
SET statement_mem='2MB';
```

## 优先权如何起作用

资源队列的**PRIORITY**设置与**MEMORY\_LIMIT**和**ACTIVE\_STATEMENTS**设置不同，后两者决定一个查询是否将被准许进入该队列并且最终被执行。**PRIORITY**设置对于活动查询适用。活动查询会按照其所在资源队列的优先权设置来共享可用的CPU资源。当一个来自高优先权队列的语句进入到活动运行语句分组中时，它可以得到可用CPU中较高的份额，同时也降低了具有较低优先权设置队列中已经在运行的语句得到的份额。

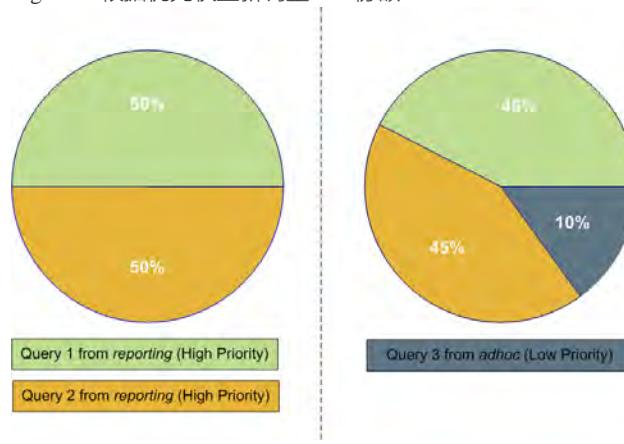
查询的相对尺寸或复杂度不影响CPU的分配。如果一个简单的低代价的查询与一个大型的复杂查询同时运行，并且它们的优先权设置相同，它们将被分配同等份额的可用CPU资源。当一个新的查询变成活动时，CPU份额将会被重新计算，但是优先权相同的查询仍将得到等量的CPU。

例如，管理员创建三个资源队列：**adhoc**用于业务分析师提交的正在进行的查询，**reporting**用于计划的报表任务，而**executive**用于行政用户角色递交的查询。由于分析师可能临时提交查询，而这些查询的资源需求是不可预测的，管理员想要确保计划的报表任务不会受到这类查询的严重影响。还有，管理员想要确保行政角色递交的查询会被分配相当份额的CPU。相应地，资源队列的优先权被设置如下：

- **adhoc** — 低优先权
- **reporting** — 高优先权
- **executive** — 最大优先权

在运行时，活动语句的CPU份额由这些优先权设置决定。如果来自报表队列的查询1和2同时运行，它们有相等份额的CPU。当一个临时查询变成活动时，它会索取一个较小份额的CPU。报表查询所使用的准确份额会被调整，但仍然保持相等，因为它们的优先权设置相等：

Figure 3. 根据优先权重新调整CPU份额

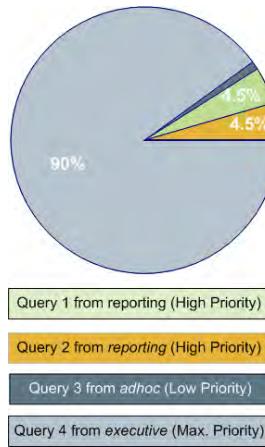


Note:

这些图中显示的百分数都是近似值。高、低和中优先权队列的CPU使用并不总是准确地用这些比例计算出来。

当一个行政查询进入到运行语句分组中时，CPU使用会被调整以说明其最大优先权设置。它可能是一个堪比分析师和报表查询的简单查询，但直到它完成前，它都将要求最大份额的CPU。

Figure 4. 为最大优先权查询重新调整CPU份额



更多有关设置优先权的命令的信息，请见[设置优先级](#).

## 启用负载管理的步骤

在Greenplum数据库中启用并且使用负载管理涉及下列高层任务：

1. 配置负载管理，见[配置负载管理](#).
2. 创建资源队列并且在其上设置限制。见[创建资源队列](#) and [修改资源队列](#).
3. 把队列分派给一个或者更多用户角色。见[指派角色 \(用户\) 到资源队列](#).
4. 使用负载管理系统视图来监控并且管理资源队列。见[检查资源队列状态](#).

## 配置负载管理

在安装Greenplum数据库时默认会启用资源调度，资源调度对所有角色都是必需的。默认的资源队列`pg_default`的活动语句限制是20，没有内存限制和中等的优先权设置。为各种类型的负载创建资源队列。

## 配置负载管理

1. 下列参数用于资源队列的一般配置：
  - `max_resource_queues` - 设置资源队列的最大数量。
  - `max_resource_portals_per_transaction` - S设置每个事务允许同时打开的游标的最大数量。注意一个打开的游标将在资源队列中占有一个活动查询槽。
  - `resource_select_only` - 如果被设置为`on`，那么`SELECT, SELECT INTO, CREATE TABLE AS``SELECT`和`DECLARE CURSOR`命令会被评估。如果被设置为`off`，`INSERT, UPDATE`和`DELETE`也将被评估。
  - `resource_cleanup_gangs_on_wait` - 在资源队列中取得一个槽之前清除空转的Segment工作者进程。
  - `stats_queue_level` - 在资源队列使用上启用统计信息收集，然后可以通过查询`pg_stat_resqueues`系统视图来查看收集到的信息。
2. 下列参数与内存利用有关：
  - `gp_resqueue_memory_policy` - 启用Greenplum数据库的内存管理特性。在Greenplum数据库4.2和其后的版本中，分布算法`eager_free`会利用并非所有操作符都会同时执行这一事实。查询计划被划分成阶段并且Greenplum数据库会饥渴地在上一阶段执行结束时释放分配给上一阶段的内存，然后将释放出来的内存饥渴地分配给新的阶段。  
当被设置为`none`时，内存管理与4.1之前Greenplum数据库发行版相同。当被设置为`auto`时，查询内存使用由`statement_mem`和资源队列内存限制所控制。
  - `statement_mem`和`max_statement_mem` - 被用来在运行时给一个特定查询分

- 配内存（覆盖资源队列指派的默认分配）。`max_statement_mem`被数据库超级用户设置以防止常规数据库用户过度分配。
- `gp_vmem_protect_limit` - 设置所有查询处理能消耗的上界并且不应超过Segment主机的物理内存量。当一台Segment主机在查询执行时达到这一限制，导致超过限制的查询将被取消。
  - `gp_vmem_idle_resource_timeout`和`gp_vmem_protect_segworker_cache_limit` - 被用来释放Segment主机上由闲置数据库进程持有的内存。管理员可能想要在有大量并发的系统上调整这些设置。
  - `shared_buffers` - 设置Greenplum服务器实例用作共享内存缓冲区的内存量。这个设置必须至少为128千字节并且至少为16千字节乘以`max_connections`。该值不能超过操作系统共享内存最大分配请求尺寸，该尺寸由Linux上的`shmmmax`控制。推荐的OS内存设置请见 *Greenplum* 数据库安装指南。

3. 下列参数与查询优先有关。注意下列参数都是本地参数，意味着它们必须在Master和所有Segment的`postgresql.conf`文件中设置：
  - `gp_resqueue_priority` - 查询优先特性默认被启用。
  - `gp_resqueue_priority_sweeper_interval` - 设置所有活动语句重新计算CPU使用的时间间隔。这个参数的默认值应该足够用于通常的数据库操作。
  - `gp_resqueue_priority_cpucores_per_segment` - 指定每个Segment实例分配的CPU核数。Master和Segment的默认值是4。对于Greenplum Data Computing Appliance Version 2，Segment的默认值是4而Master默认值是25。每台主机会在其自己的`postgresql.conf`文件中检查这个参数的值。这个参数也影响Master节点，在Master节点上它应该被设置为一个反映CPU核数更高比率的值。例如，在每台主机有10个CPU核以及4个Segment的集群上，用户可以为`gp_resqueue_priority_cpucores_per_segment`指定这些值：为Master和后备Master指定10。通常，在Master主机上只有Master实例。为Segment主机上的每个Segment实例指定2.5。如果参数值未被设置正确，要么是CPU可能不会被完全利用，要么是查询优先可能无法按照预期工作。例如，如果Greenplum数据库集群在Segment主机上每一个CPU核低于一个Segment实例，确保要相应地调整这个值。实际的CPU核利用取决于Greenplum数据库并行化查询的能力以及执行查询要求的资源。
  - 注意：操作系统任何可用的CPU核都会被包括在CPU核数中。例如，虚拟CPU核会被包括在CPU核数中。
4. 如果用户希望查看或者更改任何负载管理参数值，可以使用`gpconfig`工具。
5. 例如，要查看一个特定参数的设置：

```
$ gpconfig --show gp_vmem_protect_limit
```

6. 例如，要在所有的Segment实例上设置一个值并且在Master上设置一个不同的值：

```
$ gpconfig -c gp_resqueue_priority_cpucores_per_segment -v 2 -m 8
```

7. 重启Greenplum数据库让配置更改生效：

```
$ gpstop -r
```

## 创建资源队列

创建一个资源队列涉及到给它一个名称、设置一个活动查询限制并且可选地在该资源队列上设置一个查询优先权。使用`CREATE RESOURCE QUEUE`命令来创建新的资源队列。

## 创建带有活动查询限制的队列

带有ACTIVE\_STATEMENTS设置的资源队列会限制指派给该队列的角色所执行的查询数量。例如，要创建一个名为`adhoc`且活动查询限制为3的资源队列：

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

这意味着对于所有被分配到`adhoc`资源队列的角色，在任意给定时刻只能有三个活动查询被运行在这个系统上。如果这个队列已经有三个查询在运行并且一个角色在该队列中提交第四个查询，则第四个查询只有等到一个槽被释放出来后才能运行。

## 创建带有内存限制的队列

带有MEMORY\_LIMIT设置的资源队列控制所有通过该队列提交的查询的总内存。总内存不应超过每个Segment可用的物理内存。以每个Segment为基础，设置MEMORY\_LIMIT为90%的可用内存。例如，如果一台主机有48GB物理内存和6个Segment实例，那么每个Segment实例可用的内存是8GB。可以为单个队列按照 $0.90 \times 8 = 7.2$  GB来计算推荐的MEMORY\_LIMIT。如果在系统上创建有多个队列，它们的总内存限制加起来也必须为7.2 GB。

在与ACTIVE\_STATEMENTS联合使用时，每个查询被分配的默认内存量为：MEMORY\_LIMIT / ACTIVE\_STATEMENTS。在与MAX\_COST联合使用时，每个查询被分配的默认内存量为：MEMORY\_LIMIT \* (query\_cost / MAX\_COST)。将MEMORY\_LIMIT与ACTIVE\_STATEMENTS而不是与MAX\_COST一起使用。

例如，要创建一个活动查询限制为10且总内存限制为2000MB的资源队列（每个查询将在执行时被分配200MB的Segment主机内存）：

```
=# CREATE RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=20,
MEMORY_LIMIT='2000MB');
```

可以使用statement\_mem服务器配置参数针对每个查询覆盖默认的内存分配，前提是没有任何超过MEMORY\_LIMIT或者max\_statement\_mem。例如，要对一个特定查询分配更多内存：

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

作为一种一般性的指导方针，对于所有资源队列的MEMORY\_LIMIT不应超过一台Segment主机的物理内存量。如果负载在多个队列之间交错安排，超额分配一些内存可能是OK的，但要记住，如果执行期间Segment主机的内存限制（gp\_vmem\_protect\_limit）被超过，查询可能会被取消。

## 设置优先级

为了控制一个资源队列对可用CPU资源的消耗，管理员可以指派一个合适的优先级。当高并发导致对CPU资源的竞争时，与较高优先权资源队列相关的查询和语句将会比较低优先权的查询和语句得到更大份额的可用CPU。

优先权设置使用命令CREATE RESOURCE QUEUE和ALTER RESOURCE QUEUE的WITH参数创建或修改。例如，要为`adhoc`和`reporting`队列指定优先权设置，管理员会使用下列命令：

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

要以最大优先权创建`executive`队列，管理员可以使用下列命令：

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

当查询优先特性被启用时，如果没有显式地指派，默认会为资源队列给出一个MEDIUM优先权。更多有关运行时如何评估优先权设置的信息，请见[优先权如何起作用](#).

**Important:** 为了在活动查询负载上实施资源队列优先级，用户必须通过设置相关服务器配置参数来启用查询优先特性。请见[配置负载管理](#).

## 指派角色（用户）到资源队列

一旦创建了一个资源队列，用户必须把角色（用户）指派到它们合适的资源队列。如果没有显式地把角色指派给资源队列，它们将进入默认资源队列pg\_default。默认资源队列的活动语句限制是20，没有代价限制，优先权设置是中等。

使用ALTER ROLE或者CREATE ROLE命令来指派角色到资源队列。例如：

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

任一给定时间，一个角色只能被指派给一个资源队列，因此用户可以使用ALTER ROLE命令来初始指派角色的资源队列或者更改角色的资源队列。

资源队列必须按逐用户的方式指派。如果有一个角色层次（例如，组级别角色），那么把资源队列指派给组将不会传播到组中的用户。

超级用户总是被免除资源队列限制。超级用户查询将总是被运行，而不管它们被指派的队列上的限制。

## 从资源队列移除角色

所有用户都必须被指派到资源队列。如果没有被显式指派到一个特定队列，用户将会进入到默认的资源队列pg\_default。如果用户想要从一个资源队列移除一个角色并且把它们放在默认队列中，可以将该角色的队列指派改成none。例如：

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

## 修改资源队列

在资源队列被创建后，用户可以使用ALTER RESOURCE QUEUE命令更改或者重置队列限制。用户可以使用DROP RESOURCE QUEUE命令移除一个资源队列。要更改指派到资源队列的用户，请见[指派角色（用户）到资源队列](#).

## 修改资源队列

ALTER RESOURCE QUEUE命令更改资源队列的限制。要更改一个资源队列的限制，可以为该队列指定想要的新值。例如：

```
=# ALTER RESOURCE QUEUE adhoc WITH (ACTIVE_STATEMENTS=5);
=# ALTER RESOURCE QUEUE exec WITH (PRIORITY=MAX);
```

要重置活动语句或者内存限制为没有限制，输入值-1。要重置最大查询代价为无限制，输入值-1.0。例如：

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

用户可以使用ALTER RESOURCE QUEUE命令更改一个资源队列相关查询的优先权。例如，要把一个队列设置为中等优先级：

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

## 删除资源队列

DROP RESOURCE QUEUE命令可以删除资源队列。要删除一个资源队列，该队列不能有指派给它的角色，也不能有任何语句在其中等待。关于清空一个资源队列的指导请见[从资源队列移除角色](#)和[从资源队列清除等待语句](#)。要删除一个资源队列：

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

## 检查资源队列状态

检查资源队列状态涉及下列任务：

- [查看队列中的语句和资源队列状态](#)
- [查看资源队列统计信息](#)
- [查看指派到资源队列的角色](#)
- [查看资源队列的等待查询](#)
- [从资源队列清除等待语句](#)
- [查看活动语句的优先权](#)
- [重置活动语句的优先权](#)

## 查看队列中的语句和资源队列状态

The `gp_toolkit.gp_resqueue_status`视图允许管理员查看一个负载管理资源队列的状态和活动。对于一个特定资源队列，它展示有多少查询在等待运行以及系统中当前有多少查询是活动的。要查看系统中创建的资源队列、它们的限制属性和当前状态：

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

## 查看资源队列统计信息

如果想要持续跟踪资源队列的统计信息和性能，可以为资源队列启用统计收集。这可以通过在Master的`postgresql.conf`文件中设置下列服务器配置参数实现：

```
stats_queue_level = on
```

一旦统计收集被启用，用户可以使用`pg_stat_resqueues`系统视图来查看在资源队列使用上收集的统计信息。注意启用这一特性确实会引发一点点性能开销，因为每个通过资源队列提交的查询都必须被跟踪。可以先在资源队列上启用统计收集用于初始的诊断和管理规划，然后再连续使用中禁用该特性。

更多有关Greenplum数据库中收集统计信息的内容，可参考PostgreSQL文档中的统计收

集器部分。

## 查看指派到资源队列的角色

要查看指派给资源队列的角色，执行下列  
在pg\_roles和gp\_toolkit(gp\_resqueue\_status)系统目录表上的查询：

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

用户可能想用这个查询创建一个视图来简化未来的查询。例如：

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

然后就可以只查询该视图：

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

## 查看资源队列的等待查询

当资源队列的一个槽被使用时，它被记录在pg\_locks系统目录表中。在其中用户可以看到所有资源队列的所有当前活跃的以及在等待的查询。要检查被放入队列中的语句（甚至不在等待的语句），用户还可以使用gp\_toolkit(gp\_locks\_on\_resqueue)视图。例如：

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

如果这个查询不返回结果，那就意味着当前没有语句在资源队列中等待。

## 从资源队列清除等待语句

在某些情况下，用户可能想要从资源队列中清除等待的语句。例如，用户可能想移除在队列中等待但还未被执行的语句。用户可能还想停止已经被启动但是执行时间太久的查询，或者是在事务中闲置并且占据其他用户所需的资源队列槽的查询。要做到这一点，用户必须标识出想要清除的语句，确定它的进程ID (pid)，然后使用pg\_cancel\_backend和进程ID来结束该进程（如下所示）。可以选择对被结束的进程发送一个消息来告诉用户原因，该消息作为第二个参数传入。

例如，要查看在所有资源队列中当前活动的或者在等待的语句，运行下列查询：

```
=# 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
 AND pg_stat_activity.usename=pg_roles.rolname;
```

如果这个查询不返回结果，那就意味着当前没有语句在资源队列中等待。例如，下面的资源队列在结果中有两个语句：

| rolname | rsqname | pid | granted | current_query | datname |
|---------|---------|-----|---------|---------------|---------|
|---------|---------|-----|---------|---------------|---------|

```
 sammy | webuser | 31861 | t | <IDLE> in transaction | namesdb
 daria | webuser | 31905 | f | SELECT * FROM topten; | namesdb
```

使用这一输出来标识出想要从资源队列中清除的语句的进程ID (pid)。要清除语句，用户可以在Master主机上打开一个终端窗口（作为gpadmin数据库超级用户或者root）并且取消对应的进程。例如：

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

Note:

不要使用任何的操作系统KILL命令。

## 查看活动语句的优先权

*gp\_toolkit*管理方案有一个名为*gp\_resq\_priority\_statement*的视图，它列出了当前正在被执行的语句并且提供优先权、会话ID和其他信息。

T这个视图只有通过*gp\_toolkit*管理方案才可用。更多信息请见Greenplum数据库参考指南。

## 重置活动语句的优先权

超级用户可以使用内建函数*gp\_adjust\_priority(session\_id,statement\_count,priority)*调整当前正在被执行的语句的优先权。使用这个函数，超级用户可以提升或者降低任意查询的优先权。例如：

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

为了获得这个函数所需的会话ID和语句计数参数，超级用户可以使用*gp\_toolkit*管理方案的视图*gp\_resq\_priority\_statement*。从该视图用户可以把这些值用作该函数的参数。

- *rqpSession*列的值用作*session\_id*参数
- *rqpCommand*列的值用作*statement\_count*参数
- *rqpPriority*列的值是当前优先权。可以指定字符串值MAX, HIGH, MEDIUM或者LOW作为*priority*。

Note: *gp\_adjust\_priority()*函数只影响指定的语句。同一资源队列中后续的语句还是使用该队列正常指派的优先权执行。

## Greenplum数据库® 6.0文档

## □ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 定义外部表

**file://协议**

gpfdist://协议

gpfdists:// 协议

pxf:// 协议

s3:// 协议

使用自定义协议

处理外部表数据中的  
错误创建和使用外  
部Web表□ Examples for  
Creating External  
Tables

file:// 协议被用在一个指定操作系统文件的URI中。

该URI包括主机名、端口和该文件的路径。每个文件都必须位于一个Segment主机上由Greenplum超级用户 (gpadmin )可访问的位置。

该URI中使用的主机名必须匹配gp\_segment\_configuration 系统目录表中注册的一个Segment主机名。

如例子中所示, LOCATION子句可以有多个URI:

```
CREATE EXTERNAL TABLE ext_expenses (
 name text, date date, amount float4, category text,
 desc1 text)
LOCATION ('file://host1:5432/data/expense/*.csv',
 'file://host2:5432/data/expense/*.csv',
 'file://host3:5432/data/expense/*.csv')
FORMAT 'CSV' (HEADER);
```

用户在LOCATION子句中指定的URI的数量就是将并行工作访问外部表的Segment实例的数量。对于每一个URI, Greenplum为该文件指派一个指定主机上的主Segment。为了装载数据时的最大并行化, 可将数据与主Segment数量相同的大小均等的文件。这会保证所有的Segment都参与到装载中。每个Segment主机上的外部文件不能超过该主机上主Segment实例的数量。例如, 如果用户的阵列的每个Segment主机上有四个主Segment实例, 用户可以在每个Segment主机上放置四个外部文件。基于file:// 协议的表只能是可读表。

系统视图pg\_max\_external\_files 显示每个外部表允许多少个外部表文件。这个视图列出了使用file://协议协议时每个Segment主机上可用的文件槽。该视图只适用于file://协议。例如:

```
SELECT * FROM pg_max_external_files;
```

**Parent topic:** 定义外部表


## Greenplum数据库® 6.0文档

## □ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 定义外部表

file://协议

gpfdist://协议

gpfdists:// 协议

pxf:// 协议

s3:// 协议

使用自定义协议

处理外部表数据中的  
错误创建和使用外  
部Web表□ Examples for  
Creating External  
Tables

您可以使用Greenplum平台扩展框架 (PXF) pxf:// 协议访问驻留在外部Hadoop系统 (HDFS, Hive, HBase), 对象存储系统 (Azure, Google云端存储, Minio, S3) 和SQL数据库上的数据。

PXF协议打包为Greenplum数据库扩展。 pxf协议支持从外部数据存储读取数据。 您还可以使用pxf协议编写文本, 二进制和parquet-format的数据。

使用pxf协议查询外部数据存储时, 要指定访问的目录, 文件或表。 PXF从数据存储请求数据, 并将相关部分的数据并行传递给为查询提供服务的每个Greenplum数据库segment实例。

使用pxf协议读取或写入外部数据之前, 必须先显式初始化并启动PXF。 您还必须在每个允许用户创建外部表以访问外部数据的数据中启用PXF, 并将pxf协议的权限授予这些Greenplum数据库用户。

有关配置和使用PXF和pxf协议的更详细的信息, 请参考 [使用PXF访问外部数据](#)。

**Parent topic:** [定义外部表](#)



## Greenplum数据库® 6.0文档

- 管理员指南
- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象
  - 分布与倾斜
  - 插入, 更新, 和删除数据
  - 查询数据
  - 使用外部数据
  - 定义外部表

file://协议

gpfdist://协议

gpfdists:// 协议

pxf:// 协议

s3:// 协议

使用自定义协议

处理外部表数据中的  
错误创建和使用外  
部Web表

- Examples for  
Creating External  
Tables

一种自定义协议允许用户连接到一个不能用file://, gpfdist://, 或者 pxf:// 协议的Greenplum数据库。

创建一种自定义协议要求用户用指定的接口实现一组C函数，在Greenplum数据库中声明函数，然后使用CREATE TRUSTED PROTOCOL命令在数据库中启用该协议。

参考[自定义数据访问协议实例样例](#)

**Parent topic:** [定义外部表](#)



## Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 定义数据库对象
    - 分布与倾斜
    - 插入, 更新, 和删除数据
  - 查询数据
  - 使用外部数据
  - 定义外部表

file://协议

gpfdist://协议

gpfdists:// 协议

pxf:// 协议

s3:// 协议

使用自定义协议

处理外部表数据中的  
错误创建和使用外  
部Web表

- Examples for  
Creating External  
Tables

默认情况下, 如果外部表数据中包含有一个错误, 命令就会失败并且不会有数据被载入到目标数据库表中。

定义带有单行错误处理的外部表能够装载正确格式化的行并且隔离外部表数据中的错误。参考 [处理加载错误](#)。

**gpfdist** 文件服务器使用HTTP协议。使用 LIMIT 的外部表查询会在检索到所需的行后结束连接, 导致一个HTTP套接字错误。如果用户对使用gpfdist:// 或 http:// 协议的外部表使用带 LIMIT 的查询, 请忽略这些错误 - 数据已经被按照预期返回给数据库。

**Parent topic:** [定义外部表](#)



# External Tables

Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 定义数据库对象
  - 分布与倾斜
  - 插入, 更新, 和删除数据
  - 查询数据
  - 使用外部数据
  - 定义外部表
- file:// 协议
- gpfdist:// 协议
- gpfdists:// 协议
- pxf:// 协议
- s3:// 协议
- 使用自定义协议
- 处理外部表数据中的错误
- 创建和使用外部Web表

Examples for  
Creating External  
Tables

These examples show how to define external data with different protocols. Each `CREATE EXTERNAL TABLE` command can contain only one protocol.

Note: When using IPv6, always enclose the numeric IP addresses in square brackets.

Start `gpfdist` before you create external tables with the `gpfdist` protocol. The following code starts the `gpfdist` file server program in the background on port 8081 serving files from directory `/var/data/staging`. The logs are saved in `/home/gpadmin/log`.

```
gpfdist -p 8081 -d /var/data/staging -l /home/gpadmin/log &
```

- [Example 1—Single gpfdist instance on single-NIC machine](#)
- [Example 2—Multiple gpfdist instances](#)
- [Example 3—Multiple gpfdists instances](#)
- [Example 4—Single gpfdist instance with error logging](#)
- [Example 5—TEXT Format on a Hadoop Distributed File Server](#)
- [Example 6—Multiple files in CSV format with header rows](#)
- [Example 7—Readable External Web Table with Script](#)
- [Example 8—Writable External Table with gpfdist](#)
- [Example 9—Writable External Web Table with Script](#)
- [Example 10—Readable and Writable External Tables with XML Transformations](#)

**Parent topic:** [定义外部表](#)



## Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 定义数据库对象
    - 分布与倾斜
    - 插入, 更新, 和删除数据
  - 查询数据
  - 使用外部数据
    - 定义外部表
      - 使用PXF访问外部数据
    - 使用外部表访问外部数据

## 写一个外部数据的包装器

使用Greenplum的并行文件服务器 (gpfdist)

- 装载和卸载数据
- 性能管理

Greenplum database 5管理员指南

Greenplum database 4管理员指南

本章概述了如何写一个新的外部数据包装器。

外部表上的所有操作都是通过其外部数据包装器 (FDW) 处理的, FDW是一个包含了Greenplum数据库服务器核心调用的一组函数组成的库。外部数据包装器负责从远程数据存储中获取数据并将其递交给Greenplum数据库执行器。如果外部数据支持更新, 则包装器也必须能处理。

当你准备自己写一个包装器的时候, 包含在Greenplum数据库在github上的开源仓库里的外部数据包装器是一个很好的参考。你需要查阅一下contrib/目录下的 [file\\_fdw](#) 和 [postgres\\_fdw](#) 模块。[CREATE FOREIGN DATA WRAPPER](#) 的参考页面也提供了一些有用的信息。

**Note:** 注意: SQL标准里, 定义了一个关于写外部数据包装器的接口。Greenplum数据库没有实现这个API, 因为适配该API到Greenplum数据库里的成本太高, 并且这个API还没有得到广泛的应用。

该文章包含如下章节:

- [需要准备的东西](#)
- [已知问题和限制](#)
- [头文件](#)
- [外部数据包装器函数](#)
- [外部数据包装器回调函数](#)
- [外部数据包装器帮助函数](#)
- [Greenplum数据库注意事项](#)
- [使用PGXS建立一个外部数据包装器扩展](#)
- [部署注意事项](#)

**Parent topic:** [使用外部表访问外部数据](#)

## 需要准备的东西

当开发Greenplum数据库外部数据包装器API时:

- 必须在一个与Greenplum数据库宿主机相同硬件和软件环境的机器上开发你的代码。
- 代码必须用version-1的接口, 使用像C一样的编译式语言。想更多了解关于C语言调用规则和动态加载, 请参考 [C语言函数文档](#)。
- 目标文件中的符号名称不能有冲突, 也不能和Greenplum数据库里冲突。如果出现了这种错误, 必须重命名函数和变量。
- [复习](#) 中关于外部表的介绍。

[访问外部数据表](#)

## 已知问题和限制

Greenplum 6 Beta版的外部数据包装器实现包括如下已知问题和限制：

- Greenplum 6 Beta版不会安装任何外部数据包装器。
- Greenplum数据库仅对外部表扫描使用mpp\_execute选项值。当插入或更新一个外部表的时候，Greenplum不遵循mpp\_execute设置；所有的写操作都从master进行初始化。

## 头文件

开发外部数据包装器过程中可能使用到的头文件在greenplum-db/src/include/目录里（针对Greenplum数据库github开源仓库做开发时），或安装在\$GPHOME/include/postgresql/server/目录里（针对Greenplum安装版做开发时）：

- foreign/fdwapi.h - FDW API 相关结构体和回调函数
- foreign/foreign.h - FDW帮助相关的结构体和函数
- catalog/pg\_foreign\_table.h - 外部表定义
- catalog/pg\_foreign\_server.h - 外部服务定义

您的FDW代码也可能依赖于访问远程数据存储所需的头文件和库。

## 外部数据包装器函数

开发者必须实现一个SQL可调用的处理函数，和一个可选的验证函数。这两个函数必须使用像C一样的编译语言，都用version-1接口。

处理函数会返回一个被Greenplum数据库优化器，执行器和各个维护命令调用的函数指针结构体。处理函数必须在Greenplum数据库中注册为不带参数并且返回特殊的伪类型fdw\_handler。例如：

```
CREATE FUNCTION NEW_fdw_handler()
RETURNS fdw_handler
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;
```

写一个外部数据包装器的大部分工作就是实现这些回调函数。对于在SQL级别不可见或不可调用的FDW API回调函数和普通的C函数在[Foreign Data Wrapper Callback Functions](#)

里有详细描述。

验证器函数负责验证CREATE和ALTER命令中为其外部数据包装器提供的选项，以及使用包装器的外部服务器，用户映射和外部表。验证器函数必须注册为带两个参数，一个包含要验证的选项的文本数组，以及一个表示与选项关联的对象类型的OID。例如：

```
CREATE FUNCTION NEW_fdw_validator(text[], oid)
RETURNS void
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;
```

OID参数反映了对象存储在系统catalog中的类型，其中包括 ForeignDataWrapperRelationId, ForeignServerRelationId, UserMappingRelationId, 或 ForeignTableRelationId. 如果外部数据包装器未提供验证器函数，则Greenplum数据库不会在对象创建时或对对象更改时检查选项有效性。

## 外部数据包装器回调函数

外部数据包装器API定义了Greenplum数据库在扫描和更新外部表时调用的回调函数。API还包括了对外部表做explain和analyze操作的回调函数。

外部数据包装器处理函数会返回一个由palloc申请的包含下面描述的回调函数指针的FdwRoutine结构体。FdwRoutine结构体存储在头文件foreign/fdwapi.h中，下面是定义：

```
/*
 * FdwRoutine is the struct returned by a foreign-data
wrapper's handler
 * function. It provides pointers to the callback functions
needed by the
 * planner and executor.
 *
 * More function pointers are likely to be added in the future.
Therefore
 * it's recommended that the handler initialize the struct with
 * makeNode(FdwRoutine) so that all fields are set to NULL.
This will
 * ensure that no fields are accidentally left undefined.
*/
typedef struct FdwRoutine
{
 NodeTag type;

 /* Functions for scanning foreign tables */
 GetForeignRelSize_function GetForeignRelSize;
 GetForeignPaths_function GetForeignPaths;
 GetForeignPlan_function GetForeignPlan;
 BeginForeignScan_function BeginForeignScan;
 IterateForeignScan_function IterateForeignScan;
```

```

ReScanForeignScan_function ReScanForeignScan;
EndForeignScan_function EndForeignScan;

/*
 * Remaining functions are optional. Set the pointer
to NULL for any that
 * are not provided.
 */

/* Functions for updating foreign tables */
AddForeignUpdateTargets_function
AddForeignUpdateTargets;
PlanForeignModify_function PlanForeignModify;
BeginForeignModify_function BeginForeignModify;
ExecForeignInsert_function ExecForeignInsert;
ExecForeignUpdate_function ExecForeignUpdate;
ExecForeignDelete_function ExecForeignDelete;
EndForeignModify_function EndForeignModify;
IsForeignRelUpdatable_function IsForeignRelUpdatable;

/* Support functions for EXPLAIN */
ExplainForeignScan_function ExplainForeignScan;
ExplainForeignModify_function ExplainForeignModify;

/* Support functions for ANALYZE */
AnalyzeForeignTable_function AnalyzeForeignTable;
} FdwRoutine;

```

必须在外部数据包装器中实现和扫描相关的函数，其他回调函数是可选的。

扫描相关的函数包括：

| 回调签名                                                                                             | 描述                                                                                                                         |
|--------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <pre> void GetForeignRelSize (PlannerInfo *root, RelOptInfo *baserel, Oid foreigntableid) </pre> | 获取一个外部表大小的估算值。在对一个外部表做查询计划的开始时调用。                                                                                          |
| <pre> void GetForeignPaths (PlannerInfo *root, RelOptInfo *baserel, Oid foreigntableid) </pre>   | 为一个外部表的扫描操作创建可能的访问路径。在查询计划过程中调用。<br>Note: 注意：一个与Greenplum数据库相兼容的FDW必须在它的GetForeignPaths()回调函数中调用create_foreignscan_path()。 |

|  |                                                                                                                                                                   |                                             |
|--|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------|
|  | <pre>ForeignScan * GetForeignPlan (PlannerInfo  *root, RelOptInfo  *baserel, Oid foreigntableid, ForeignPath  *best_path, List *tlist, List  *scan_clauses)</pre> | 从已选择的外部访问路径中创建一个ForeignScan计划节点。在查询计划结束时调用。 |
|  | <pre>void BeginForeignScan (ForeignScanState  *node, int eflags)</pre>                                                                                            | 开始执行一个外部扫描。在执行器启动时调用。                       |
|  | <pre>TupleTableSlot * IterateForeignScan (ForeignScanState  *node)</pre>                                                                                          | 从外部源获取一行，并在一个元组表槽中返回；如果没有可用的行则返回NULL。       |
|  | <pre>void ReScanForeignScan (ForeignScanState  *node)</pre>                                                                                                       | 从头开始重新扫描。                                   |
|  | <pre>void EndForeignScan (ForeignScanState  *node)</pre>                                                                                                          | 结束扫描并释放资源。                                  |

关于FDW回调函数的输入输出更详细的信息，请参考PostgreSQL文档[Foreign Data Wrapper Callback Routines](#)。

# 外部数据包装器帮助函数

FDW API从Greenplum数据库核心服务输出了几个帮助函数，所以外部数据包装器的作者可以轻松的访问FDW相关对象的属性，比如当用户创建或修改外部数据包装器、服务或外部表时提供的选项。要使用这些帮助函数，必须在源文件里包含`foreign.h`头文件：

```
#include "foreign/foreign.h"
```

FDW API包括了下面表格列出的这些帮助函数。想了解更多关于这些函数的信息请参考PostgreSQL文档 [Foreign Data Wrapper Helper Functions](#)。

| 帮着签名                                                                                            | 描述                                                     |
|-------------------------------------------------------------------------------------------------|--------------------------------------------------------|
| <pre>ForeignDataWrapper * GetForeignDataWrapper(Oid fdwid);</pre>                               | 通过给定的OID，返回外部数据包装器 <code>ForeignDataWrapper</code> 对象。 |
| <pre>ForeignDataWrapper * GetForeignDataWrapperByName(const char *name, bool missing_ok);</pre> | 通过给定的名称，返回外部数据包装器 <code>ForeignDataWrapper</code> 对象。  |
| <pre>ForeignServer * GetForeignServer(Oid serverid);</pre>                                      | 通过给定的OID，返回外部服务的 <code>ForeignServer</code> 对象。        |
| <pre>ForeignServer * GetForeignServerByName(const char *name, bool missing_ok);</pre>           | 通过给定的名称，返回外部服务的 <code>ForeignServer</code> 对象。         |
| <pre>UserMapping *  GetUserMapping(Oid userid, Oid serverid);</pre>                             | 在给定的服务上为给定的用户返回用户映射 <code>UserMapping</code> 对象。       |
| <pre>ForeignTable * GetForeignTable(Oid relid);</pre>                                           | 根据OID返回对应外部表的 <code>ForeignTable</code> 对象。            |
| <pre>List * GetForeignColumnOptions(Oid relid, AttrNumber attnum);</pre>                        | 根据OID对应的外部表和属性编号，返回列的FDW选项。                            |

# Greenplum数据库注意事项

一个Greenplum数据库用户可以在创建和修改一个外部表、外部服务或外部数据包装器时指定mpp\_execute选项。一个Greenplum数据库兼容的外部数据包装器会在扫描时检查mpp\_execute属性值并根据它决定向哪发送数据，包括：向master（默认值），任何节点（master或任何一个节点），或所有节点。

**Note:** 注意：不管mpp\_execute如何设置，使用外部数据包装器的写/更新操作总是在Greenplum数据库master节点上执行。

下面的扫描代码片段探测与外部表关联的mpp\_execute值。

```
ForeignTable *table = GetForeignTable(foreignTableId);
if (table->exec_location == FTEXECLOCATION_ALL_SEGMENTS)
{
 ...
}
else if (table->exec_location == FTEXECLOCATION_ANY)
{
 ...
}
else if (table->exec_location == FTEXECLOCATION_MASTER)
{
 ...
}
```

如果外部表创建和外部服务的时候没有设置mpp\_execute，外部数据包装器会探测并决定。如果没有任何一个外部数据相关的对象设置mpp\_execute，则使用默认的master。

如果外部数据包装器支持mpp\_execute设置成'all'，它会实现一个Greenplum节点与数据匹配的策略。为了不重复的从远处取数据，每个节点上的FDW必须能决定哪部分数据归它负责。一个FDW会使用节点标识和节点数量去帮助做这个决定。下面这段代码演示了一个外部数据包装器是怎样获取节点编号和节点数量的：

```
int segmentNumber = GpIdentity.segIndex;
int totalNumberOfSegments = getgpsegmentCount();
```

## 使用PGXS建立一个外部数据包装器扩展

将你用FDW API写的外部数据包装器函数编译成Greenplum数据库按需加载的一个或多个共享库。

你可以使用PostgreSQL扩展基础库(PGXS)为Greenplum数据库安装构建外部数据包装器的源代码。该框架自动化了简单模块的常用构建规则。如果你需要构建一个更复杂的用例，你必须写自己的构建系统。

使用PGXS基础库为你的FDW生成一个共享库，创建一个设置好PGXS相关变量的简单的Makefile。

Note: 注意：请参考[Extension Building Infrastructure](#) 文档中关于PGXS支持的Makefile变量。

例如：下面Makefile根据base\_fdw\_1.c和base\_fdw\_2.c这两个C的源文件在当前工作目录生成了一个名字叫base\_fdw.so的共享库：

```
MODULE_big = base_fdw
OBJS = base_fdw_1.o base_fdw_2.o

PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)

PG_CPPFLAGS = -I$(shell $(PG_CONFIG) --includedir)
SHLIB_LINK = -L$(shell $(PG_CONFIG) --libdir)
include $(PGXS)
```

下面是Makefile中用到的指令的描述：

- MODULE\_big - 标识Makefile生成的共享库的基本名称。
- PG\_CPPFLAGS - 将Greenplum数据库安装中的include/目录加到到编译器头文件的查找路径里。
- SHLIB\_LINK 将Greenplum数据库安装中的库目录(\$GPHOME/lib/)加到链接查找路径里。
- PG\_CONFIG和PGXS变量设置和include语句是必需的，通常位于Makefile的最后三行。

要将外部数据包装器打包为Greenplum数据库扩展，可以创建脚本 (newfdw - version.sql) 和控制 (newfdw.control) 文件，这些文件注册FDW句柄和验证器函数，创建外部数据包装器并识别FDW共享库文件的特征。

Note: 注意：PostgreSQL文档[Packaging Related Objects into an Extension](#) 描述了如何打包一个扩展。

示例名为base\_fdw--1.0.sql的外部数据包装器扩展脚本样例：

```
CREATE FUNCTION base_fdw_handler()
RETURNS fdw_handler
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;

CREATE FUNCTION base_fdw_validator(text[], oid)
RETURNS void
```

```

 AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;

CREATE FOREIGN DATA WRAPPER base_fdw
HANDLER base_fdw_handler
VALIDATOR base_fdw_validator;

```

示例名为base\_fdw.control的FDW控制文件：

```

base_fdw FDW extension
comment = 'base foreign-data wrapper implementation; does not
do much'
default_version = '1.0'
module_pathname = '$libdir/base_fdw'
relocatable = true

```

把下面这些指令加到Makefile里，可以指定FDW扩展控制文件的基本名称（EXTENSION）和SQL脚本（DATA）：

```

EXTENSION = base_fdw
DATA = base_fdw--1.0.sql

```

在包含有这些命令的Makefile文件目录里运行make install会将共享库、FDW SQL和控制文件拷贝到Greenplum数据库安装目录中（\$GPHOME）特定的或默认的位置。

## 部署注意事项

您必须以适合在Greenplum数据库群集中部署的形式打包FDW共享库和扩展文件。当你创建和部署安装包时，考虑一下因素：

- FDW共享库必须安装到集群中master节点和每个segment节点的相同位置。在.control文件里指定这个路径。这个路径通常是\$GPHOME/lib/postgresql/。
- FDW的.sql和.control文件必须安装到集群中 master节点和每个segment节点的\$GPHOME/share/postgresql/extension/路径。
- gpadmin用户必须拥有访问整个FDW共享库和扩展文件的权限。

## Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 定义数据库对象
    - 分布与倾斜
    - 插入, 更新, 和删除数据
  - 查询数据
  - 使用外部数据
  - 装载和卸载数据
    - 使用外部表装载数据
    - 装载和写入非HDFS自定义数据
    - 使用一种自定义格式
    - 使用一种自定协议
  - 处理装载错误
- 用gupload装载数据
- 使用PXF访问外部数据
- 使用gpfldist和gupload转换外部数据
- 用COPY装载数据
  - 在单行错误隔离模式中运行COPY

用户在CREATE EXTERNAL TABLE的FORMAT子句中指定一种自定义数据格式。

```
FORMAT 'CUSTOM' (formatter=format_function,
key1=val1,...keyn=valn)
```

这里 'CUSTOM' 关键词表示数据是一种自定义格式, 而formatter指定用来格式化该数据的函数, 其后跟着给formatter函数的用逗号分隔的参数。

Greenplum数据库为格式化固定宽度的数据提供了函数, 但是用户必须编写用于可变宽度数据的formatter函数。步骤如下。

1. 编写并且编译输入和输出函数为一个共享库。
2. 用CREATE FUNCTION在Greenplum数据库中指定该共享库函数。
3. 使用CREATE EXTERNAL TABLE的FORMAT子句的formatter参数来调用该函数。
  - [导入和导出固定宽度的数据](#)
  - [例子：读取宽度固定的数据](#)

**Parent topic:** [装载和写入非HDFS自定义数据](#)



## Greenplum数据库® 6.0文档

## □ 管理员指南

- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象
- 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

- 使用外部数据
- 装载和卸载数据

## 使用外部表装载数据

## □ 装载和写入非HDFS自定义数据

## □ 使用一种自定义格式

## 使用一种自定协议

## □ 处理装载错误

## 用gupload装载数据

## 使用PXF访问外部数据

## 使用gpfdist和gupload转换外部数据

## 用COPY装载数据

## 在单行错误隔离模式中运行COPY

Greenplum提供了诸如gpfdist、http以及file等协议来通过网络访问数据，或者用户可以创作一个自定义协议。用户可以为自定义协议使用标准的数据格式TEXT和CSV，或者使用一种自定义数据格式。

只要可用的内建协议无法满足一种特定需要，用户就可以创建一种自定义的协议。例如，如果用户需要并行地把Greenplum数据库直接连接到另一个系统，并且在两者之间发送数据流而无需在磁盘上物化系统数据或者使用gpfdist这样的中间进程。用户必须是超级用户才能创建和注册自定义协议。

1. 用C语言和预定义的API创作发送、接受以及（可选的）验证器函数。这些函数会被编译并且注册在Greenplum数据库中。自定义协议的例子可见[自定义数据访问协议实例](#)
2. 在编写并且编译读写函数到一个共享对象 (.so) 中之后，声明一个数据库函数指向该.so文件和函数名。

下面的例子使用了编译好的导入和导出代码。

```
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;
```

可选函数的格式是：

```
CREATE OR REPLACE FUNCTION myvalidate() RETURNS void
AS '$libdir/gpextprotocol.so', 'myprot_validate'
LANGUAGE C STABLE;
```

3. 创建一个访问这些函数的协议。Validatorfunc是可选的。

```
CREATE TRUSTED PROTOCOL myprot(
writefunc='mywrite',
readfunc='myread',
validatorfunc='myvalidate');
```

4. 必要时，把访问权限授予给任何其他用户。

```
GRANT ALL ON PROTOCOL myprot TO otheruser;
```

5. 在可读或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';
```

用SQL命令CREATE TRUSTED PROTOCOL声明自定义协议，然后使用GRANT命令把访问权限授予给用户。例如：

- 允许一个用户使用一种受信的协议创建一个可读的外部表

```
GRANT SELECT ON PROTOCOL <protocol name> TO <user name>;
```

- 允许一个用户使用一种受信的协议创建一个可写的外部表

```
GRANT INSERT ON PROTOCOL <protocol name> TO <user name>;
```

- 允许一个用户使用一种受信的协议创建一个可读写的外部表

```
GRANT ALL ON PROTOCOL <protocol name> TO <user name>;
```

**Parent topic:** [装载和写入非HDFS自定义数据](#)

## Greenplum数据库® 6.0文档

 管理员指南 Greenplum数据库概念 管理一个Greenplum系统

## 关于Greenplum数据库

## 发布版本号

## 启动和停止Greenplum数据库

 访问数据库 配置Greenplum数据库系统

## 关于Greenplum数据库的Master参数和本地参数

 设置配置参数

## 设置本地配置参数

 设置Master配置参数

## 查看服务器配置参数设置

## 配置参数种类

## 启用压缩

 启用高可用和数据持久化特征 备份和恢复数据库 扩容Greenplum系统

要在多个Segment中改变一个本地配置参数，在每一个目标Segment的`postgresql.conf`文件中更新该参数，包括主要的和镜像的Segment。使用`gpconfig`工具可以在所有的Greenplum `postgresql.conf` 文件中设置一个参数。例如：

```
$ gpconfig -c gp_vmem_protect_limit -v 4096
```

重启Greenplum数据库让配置改变生效：

```
$ gpstop -r
```

**Parent topic:** [设置配置参数](#)



## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号启动和停  
止Greenplum数据库

## □ 访问数据库

□ 配置Greenplum数据库  
系统关于Greenplum数据  
库的Master参数和本  
地参数

## □ 设置配置参数

## 设置本地配置参数

□ 设置Master配置参  
数查看服务器配置参数  
设置

## 配置参数种类

## 启用压缩

□ 启用高可用和数据持久  
化特征

## □ 备份和恢复数据库

## □ 扩容Greenplum系统

要设置Master配置参数，请在Greenplum数据库的Master实例上设置它。如果它也是一个*session*参数，用户可以为一个特定数据库、角色或者会话设置该桉树。如果一个参数在多个级别上都被设置，最细粒度级别的设置会优先。例如，会话覆盖角色，角色覆盖数据库，而数据库覆盖系统。

- [设置系统级别参数](#)
- [设置数据库级别参数](#)
- [设置角色级别参数](#)
- [设置会话级别参数](#)

**Parent topic:** [设置配置参数](#)

## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号启动和停  
止Greenplum数据库

## □ 访问数据库

□ 配置Greenplum数据库  
系统关于Greenplum数据  
库的Master参数和本  
地参数

## □ 设置配置参数

## 设置本地配置参数

□ 设置Master配置参  
数设置系统级别  
参数设置数据库级  
别参数设置角色级别  
参数设置会话级别  
参数

## 查看服务器配置参数

Master的`postgresql.conf`文件中的Master参数设置是系统范围默  
认的。要设置一个Master参数：

1. 编辑`$MASTER_DATA_DIRECTORY/postgresql.conf`文件。
2. 找到要设置的参数，取消它的注释（移除前面的#字符），并且输入想要的值。
3. 保存并且关闭该文件。
4. 对于不需要重新启动服务器的`session`参数，按如下上  
传`postgresql.conf`的改变：

```
$ gpstop -u
```

5. 对于要求服务器重启的参数更改，按如下重启Greenplum数据库：

```
$ gpstop -r
```

关于服务器配置参数的细节，请见 *Greenplum*数据库参考指南.

**Parent topic:** [设置Master配置参数](#)







## Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号

启动和停  
止Greenplum数据库

- 访问数据库
- 配置Greenplum数据库  
系统

关于Greenplum数据  
库的Master参数和本  
地参数

- 设置配置参数

设置本地配置参数

- 设置Master配置参  
数

设置系统级别  
参数

设置数据库级  
别参数

设置角色级别  
参数

设置会话级别  
参数

查看服务器配置参数

使用ALTER DATABASE在数据库级别设置参数。例如：

```
=# ALTER DATABASE mydatabase SET search_path TO myschema;
```

当用户在数据库级别设置一个会话参数时，每一个连接到该数据库的会话都使用该参数设置。数据库级别的设置覆盖系统级别的设置。

**Parent topic:** [设置Master配置参数](#)







## Greenplum数据库® 6.0文档

- [管理员指南](#)
- [Greenplum数据库概念](#)
- [管理一个Greenplum系统](#)

关于Greenplum数据库  
发布版本号

启动和停  
止Greenplum数据库

- [访问数据库](#)
- [配置Greenplum数据库  
系统](#)

关于Greenplum数据  
库的Master参数和本  
地参数

- [设置配置参数](#)

设置本地配置参数

- [设置Master配置参  
数](#)

设置系统级别  
参数

设置数据库级  
别参数

设置角色级别  
参数

设置会话级别  
参数

查看服务器配置参数

使用ALTER ROLE在角色级别设置参数。例如：

```
=# ALTER ROLE bob SET search_path TO bobschema;
```

当用户在角色级别设置一个会话参数时，每一个由该角色启动的会话都使用该参数设置。角色级别的设置覆盖数据库级别的设置。

**Parent topic:** [设置Master配置参数](#)







## Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  
- 关于Greenplum数据库发布版本号
  
- 启动和停止Greenplum数据库
  - 访问数据库
  - 配置Greenplum数据库系统

关于Greenplum数据库的Master参数和本地参数

- 设置配置参数

设置本地配置参数

- 设置Master配置参数

设置系统级别参数

设置数据库级别参数

设置角色级别参数

设置会话级别参数

查看服务器配置参数

任何会话参数都可以在一个活动数据库会话中用SET命令设置。例如：

```
=# SET statement_mem TO '200MB';
```

该参数设置对于这个会话的剩余时间都有效，直到发出一个RESET命令。例如：

```
=# RESET statement_mem;
```

会话级别的设置覆盖角色级别的设置。

**Parent topic:** [设置Master配置参数](#)







## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## 关于Greenplum数据库

## 发布版本号

## 启动和停止Greenplum数据库

## □ 访问数据库

## □ 配置Greenplum数据库系统

## 启用压缩

## □ 启用高可用和数据持久化特征

## □ Greenplum数据库高可用性概述

## Segment镜像概述

## Master镜像概述

## □ 在Greenplum数据库中启用镜像

## □ 检测故障的Segment

## □ 恢复故障Segment

## □ 恢复故障的Master

## □ 备份和恢复数据库

## □ 扩容Greenplum系统

当Greenplum数据库高可用性被启用时，有两种类型的Segment：*primarySegment* 和*mirrorSegment*。每个主Segment都有一个对应的镜像Segment。主Segment从 Master接收请求来对该Segment的数据库做更改并且接着把那些更改复制到对应的镜像。如果主Segment变成不可用，镜像segment会切换为主segment，不可用的主segment会切换为镜像segment。故障出现时正在进行的事务会回滚并且必须重启数据库。接下来管理员必须恢复镜像segment，并允许镜像segment与当前主segment进行同步，最后需要交换主segment和镜像segment的角色，让他们处于自己最佳的角色状态。

如果segment镜像未启用，当出现segment实例故障时，Greenplum数据库系统会关闭。管理员必须手工恢复所有失败的segment实例然后才能重新启动数据库。

如果现有系统已经启用了segment镜像，当主segment实例正在生成一个快照时，主segment实例继续为用户提供服务。当主segment快照完成并向镜像segment实例部署时，主segment的变化也会被记录。当快照完全部署到镜像segment后，镜像segment会同步这些变化并采用WAL流复制的方式与主segment保持一致。Greenplum数据库的WAL复制使用 `walsender` 和 `walreceiver` 复制进程。`walsender` 进程是主segment的进程。`walreceiver` 进程是镜像segment 的进程。

当数据库变化出现时，日志捕获到该变化然后将变化流向镜像segment，以保证镜像与其对应的主segment一致。在WAL 复制期间，数据库变化在被应用之前先写入日志中，以确保任何正在处理的数据的完整性。

当Greenplum数据库检测到主segment故障时，WAL复制进程停止，镜像segment自动接管成为活动主segment。如果主segment活动时，镜像segment故障或变得不可访问，主segment会追踪数据库变化并记录在日志中，当镜像恢复后 应用到镜像节点。有关segment故障检测和故障处理的详细信息，请见 [检测故障的Segment](#)。

Greenplum数据库系统表包含镜像和复制的详细信息。

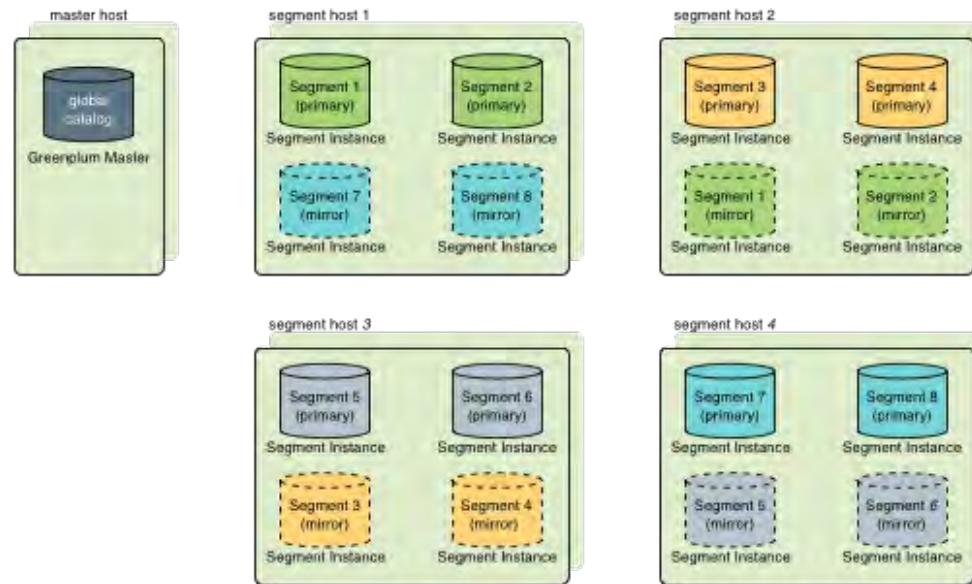
- 系统表 `gp_segment_configuration` 包含主segment、镜像segment、主master和standby master实例的当前配置和状态信息。
- 系统视图 `gp_stat_replication` 包含Greenplum数据库master和segment镜像功能使用的 `walsender` 进程状态统计信息。

# 关于Segment镜像配置

镜像segment实例可以根据配置的不同在集群主机中按不同的方式分布。作为最佳实践，主segment和它对应的镜像放在不同的主机上。每台主机必须有相同的主和镜像segment。当你使用Greenplum工具[gpinitsystem](#)或[gpaddmirrors](#)创建segment镜像时，可以设定segment镜像方式：以分组的方式镜像（默认）或以打散的方式镜像。采用[gpaddmirrors](#)命令时，可以先创建[gpaddmirrors](#)配置文件，然后将文件放在命令行读取执行。

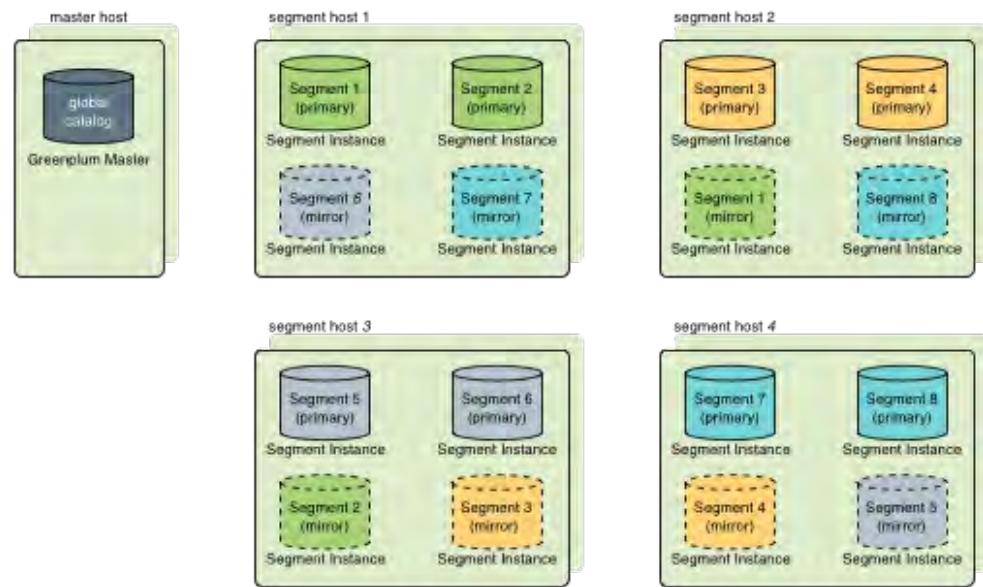
以分组的方式镜像是默认的镜像方式。该方式每台主机的主segment对应的镜像都整体放在另一台主机上，如果有一台主机故障，另外一台接管该主机服务的镜像所在的机器的活动segment数量便会翻倍。图表[Figure 1](#) 显示了以分组的方式配置segment镜像。

Figure 1. 以分组的方式在Greenplum数据库中分布镜像



以打散的方式镜像 将每一个主机的镜像分散到多台主机上，保证每一个机器上至多只有一个 镜像提升为主segment，这种方式可以防止单台主机故障后，另外的主机压力骤增。以打散方式镜像 分布要求集群主机数量多于每台主机上segment的数量。图表[Figure 2](#) 显示了如何以打散的方式配置segment镜像。

Figure 2. 以打散的方式在Greenplum数据库中分布镜像



**Parent topic:** [Greenplum数据库高可用性概述](#)

## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号

启动和停  
止Greenplum数据库

## □ 访问数据库

□ 配置Greenplum数据库  
系统

启用压缩

□ 启用高可用和数据持久  
化特征□ Greenplum数据库高  
可用性概述

Segment镜像概述

## Master镜像概述

□ 在Greenplum数据库  
中启用镜像

## □ 检测故障的Segment

## □ 恢复故障Segment

## □ 恢复故障的Master

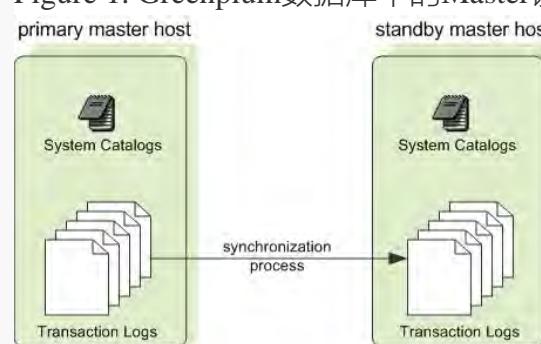
## □ 备份和恢复数据库

## □ 扩容Greenplum系统

可以在单独的主机或者同一台主机上部署Master实例的一个备份或者镜像。当主Master变得无法使用时，备份 Master或者后备Master会作为为一个温备提供服务。可在主Master在线时从中创建一个后备Master。

在取一个主Master实例的事务快照时，主Master会继续为用户提供服务。在取事务快照以及在后备Master上部署 事务快照时，对主Master的更改也会被记录。在该快照被部署在后备Master上之后，那些更新会被部署以同步后备 Master和主Master。当镜像Master快照部署完成后，变化会应用，二者之间会通过基于WAL日志的流复制的方式 保持同步。Greenplum WAL复制通  
过walsender和walreceiver 复制进程保持同步。walsender是  
主master的进程。walreceiver 是standby master的进程。

Figure 1. Greenplum数据库中的Master镜像



由于Master不保存用户数据，只有系统目录表被在主Master和后  
备Master之间同步。当这些表被更新时，复制日志会捕获变化并流  
向standby master以保持与其主master的同步。WAL复制期间，所有  
的数据库更改在应用之前都会写入复制日志，以保证任何正在进行的操  
作的数据完整性。

以下是Greenplum数据库如何处理Master故障

- 如果主master故障，Greenplum数据库系统关闭，master复制进程停  
止。管理员运行 `gpactivatestandby`命令将standby master提升  
为主master。随着standby master提升为主，复制日志会重建  
主master最后成功提交的事务的相关日志。活动standby master接  
下来会完全成为Greenplum数据库的master，以standby master初始化  
时定义的端口号接受外部连接。详情请见[恢复故障的Master](#)。
- 如果在主master可用时，standby master故障或不可访问。  
主master会将数据库变化写入追踪日志，然后等待standby master恢  
复后应用到其上面。

以下这些Greenplum系统表包含镜像和复制相关信息。

- 系统表[`gp\_segment\_configuration`](#)包含主segment、镜

像segment、主master和standby master实例的当前配置 及状态信息。

- 系统视图[gp\\_stat\\_replication](#)包含Greenplum数据库master和segment镜像用到的walsender 进程的复制统计信息。

**Parent topic:** [Greenplum数据库高可用性概述](#)

## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号启动和停  
止Greenplum数据库

## □ 访问数据库

□ 配置Greenplum数据库  
系统

## 启用压缩

□ 启用高可用和数据持久  
化特征□ Greenplum数据库高  
可用性概述□ 在Greenplum数据库  
中启用镜像

## 启用Segment镜像

## 启用Master镜像

## □ 检测故障的Segment

## □ 恢复故障Segment

## □ 恢复故障的Master

## □ 备份和恢复数据库

## □ 扩容Greenplum系统

镜像Segment允许数据库查询在主Segment不可用时故障转移到备用Segment。为了配置镜像，Greenplum数据库系统必须具有足够的节点以保证镜像Segment和对应的主Segment不在同一台主机上。默认情况下，镜像会被配置在主Segment所在的主机阵列上。也可以为镜像Segment选择一组完全不同的主机，这样它们就不会分享任何主Segment的机器。

**Important:** 在线数据复制处理期间，Greenplum数据库应该处于一种静止状态，不应运行负载和其他查询。

## 要增加Segment镜像到一个现有系统 (和主Segment相同的主机阵列)

1. 在所有的Segment主机上为镜像数据分配数据存储区域。数据存储区域必须与主Segment的文件系统位置不同。
2. 使用`gpssh-exkeys`确保Segment主机能通过SSH和SCP免密码连接到彼此。
3. 运行`gpaddmirrors`工具在Greenplum数据库系统中启用镜像。例如，在主Segment端口号基础上加10000来计算得到镜像Segment的端口号：

```
$ gpaddmirrors -p 10000
```

其中-p指定要加在主 Segment端口号上的数字。使用默认的组镜像配置来增加镜像。

## 要增加Segment镜像到一个现有系统 (和主Segment不同的主机阵列)

1. 确保在所有主机上都安装有Greenplum数据库软件。详细的安装指导请见Greenplum数据库安装指南。
2. 在所有的Segment主机上为镜像数据分配数据存储区：
3. 使用`gpssh-exkeys`确保Segment主机能通过SSH和SCP免密码连

接到彼此。

4. 创建一个配置文件，其中列出要在其上创建镜像的主机名称、端口号和数据目录。要创建一个示例配置文件作为起点，可运行：

```
$ gpaddmirrors -o filename
```

镜像配置文件的格式为：

```
mirrorrow_id=contentID:address:port:data_dir
```

其中`row_id`是文件中的行号, `contentID`是segment实例的内容ID, `address`是segment主机的主机名或IP地址, `port`是用于通信的端口号, `data_dir`是segment实例的数据目录。

例如这是一个配置文件，其中有两个Segment主机，每个主机上有两个Segment：

```
mirror0=2:sdw1-1:41000:/data/mirror1/gp2
mirror1=3:sdw1-2:41001:/data/mirror2/gp3
mirror2=0:sdw2-1:41000:/data/mirror1/gp0
mirror3=1:sdw2-2:41001:/data/mirror2/gp1
```

5. 运行`gpaddmirrors`工具在Greenplum数据库系统中启用镜像：

```
$ gpaddmirrors -i mirror_config_file
```

其中`-i`指定所创建的镜像配置文件。

**Parent topic:** [在Greenplum数据库中启用镜像](#)

## Greenplum数据库® 6.0文档

- 管理员指南
- Greenplum数据库概念
- 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号启动和停  
止Greenplum数据库

- 访问数据库
- 配置Greenplum数据库  
系统
- 启用压缩
- 启用高可用和数据持久  
化特征

- Greenplum数据库高  
可用性概述
- 在Greenplum数据库  
中启用镜像

## 启用Segment镜像

## 启用Master镜像

- 检测故障的Segment
- 恢复故障Segment
- 恢复故障的Master
- 备份和恢复数据库
- 扩容Greenplum系统

可以用`gpinitstandby`来配置一个带有后备Master的新Greenplum数据库系统，或者以后用`gpinitstandby`来启用后备Master。这个主题假定现有系统初始化时没有后备Master，现在要向其中加入一个后备Master。

有关工具`gpinitsystem`和`gpinitstandby`的详细信息，请见Greenplum数据库工具指南。

## 要向一个现有系统增加一个standby master

1. 确保standby Master主机已经被安装且配置好Greenplum数据库：  
`gpadmin`系统用户已创建、Greenplum数据库二进制文件已安装、环境变量已设置、SSH密钥已交换并且数据目录和表空间已创建。
2. 在当前活动的`primarymaster`主机上运行`gpinitstandby`工具向Greenplum数据库系统增加一个standby master主机。例如：

```
$ gpinitstandby -s smdw
```

这里`-s`指定后备Master主机的名称。

要把操作切换到standby master上，请见[恢复故障的Master](#)。

## 检查Master镜像进程状态（可选）

可以通过运行`gpstate`工具并带有`-f`选项来显示standby master 主机的详细信息。

```
$ gpstate -f
```

standby master的状态应该是`passive`，WAL sender状态应该是`streaming`。

有关`gpstate`工具的详细信息，请见Greenplum数据库工具指南。

**Parent topic:** 在Greenplum数据库中启用镜像



## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号启动和停  
止Greenplum数据库

## □ 访问数据库

□ 配置Greenplum数据库  
系统

## 启用压缩

□ 启用高可用和数据持久  
化特征□ Greenplum数据库高  
可用性概述□ 在Greenplum数据库  
中启用镜像

## □ 检测故障的Segment

## 检测故障Segment

## 检查日志文件

## □ 恢复故障Segment

## □ 恢复故障的Master

## □ 备份和恢复数据库

## □ 扩容Greenplum系统

启用镜像的情况下，系统允许存在故障Segment并且不影响数据库服务正常运行。可以通过 `gpstate` 工具查看整个数据库系统状态。`gpstate` 可以展示整个数据库系统中包括主Segment、镜像Segment、Master和备用Master在内的每一个单独系统组件的状态。

## 如何检测故障Segment

- 在Master上，用-e选项运行`gpstate`工具显示有错误情况的Segment：

```
$ gpstate -e
```

如果工具列出Segments with Primary and Mirror Roles Switched，表明segment不在他的原始角色（系统初始化时赋予的角色）。这意味着系统处于一种潜在的不平衡状态，因为一些segment主机可能比正常状态下拥有更多的活动segment，并不处于它们的最佳性能状态。

Segments展示Config status为Down则预示着其对应的镜像segment产生故障下线了。

修复该问题的详细过程，请见[从Segment故障中恢复](#)。

- 为了获取故障segment的详细信息，可以检查`gp_segment_configuration`系统表。例如：

```
$ psql postgres -c "SELECT * FROM gp_segment_configuration WHERE status='d';"
```

- 针对那些故障的segment实例，注意主机、端口号、原来角色和数据目录。这些信息会帮助您在问题定位时提供有用的决策信息。

- 要展示镜像segment实例的信息，运行：

```
$ gpstate -m
```

**Parent topic:** [检测故障的Segment](#)



Greenplum数据库® 6.0文档

- 管理员指南

- Greenplum数据库概念

- 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号

启动和停  
止Greenplum数据库

- 访问数据库

- 配置Greenplum数据库  
系统

启用压缩

- 启用高可用和数据持久  
化特征

- Greenplum数据库高  
可用性概述

- 在Greenplum数据库  
中启用镜像

- 检测故障的Segment

检测故障Segment

**检查日志文件**

- 恢复故障Segment

- 恢复故障的Master

- 备份和恢复数据库

- 扩容Greenplum系统

日志文件可以提供信息来帮助判断一个错误的成因。每个Master和Segment实例都在其数据目录的 `pg_log` 中有它们自己的日志文件。Master的日志文件包含了大部分信息，应该总是首先检查它。

`gplogfilter` 工具可以用来检查Greenplum数据库日志文件。如果要检查segment日志文件，使用`gpssh`在segment主机上执行`gplogfilter`工具。

## 如何检查日志文件

1. 对于WARNING、ERROR、FATAL或PANIC日志级别的消息，使用`gplogfilter`检查Master的日志文件：

```
$ gplogfilter -t
```

2. 对于每个Segment实例上的WARNING、ERROR、FATAL或PANIC日志级别的消息，使用`gpssh`检查。例如：

```
$ gpssh -f seg_hosts_file -e 'source
/usr/local/greenplum-db/greenplum_path.sh ; gplogfilter
-t
/data1/primary/*/*/pg_log/gpdb*.log' > seglog.out
```

**Parent topic:** [检测故障的Segment](#)



## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号启动和停  
止Greenplum数据库

## □ 访问数据库

□ 配置Greenplum数据库  
系统

## 启用压缩

□ 启用高可用和数据持久  
化特征□ Greenplum数据库高  
可用性概述□ 在Greenplum数据库  
中启用镜像

## □ 检测故障的Segment

## □ 恢复故障Segment

□ 从Segment故障中  
恢复

## □ 恢复故障的Master

## □ 备份和恢复数据库

## □ 扩容Greenplum系统

Segment主机故障通常会导致多个Segment故障：在该主机上的所有主Segment或者镜像Segment都被标记为“down”并且不可操作。如果没有启用镜像并且一个Segment宕掉，系统会自动变成不可操作。

segment实例可能会因为很多原因产生故障。例如主机故障，网络故障或磁盘故障。当segment实例故障时，它的状态在系统表中被标记为*down*，它的镜像切换到*change tracking*模式。为了将故障segment实例修复并重新进入可操作状态，您必须首先修复导致故障的问题，然后使用gprecoverseg工具恢复该故障实例。

如果一台segment主机不可恢复，此时您已经失去了一个或多个segment实例，那么您可以尝试从它的镜像segment来恢复该segment实例。详见[当一台Segment主机不可恢复时](#)。您也可以从备份文件重新创建Greenplum数据库。详见[备份和恢复数据库](#)。

## 在启用了镜像的情况下恢复

- 确保可以从Master主机连接到该Segment主机。例如：

```
$ ping failed_seg_host_address
```

- 排查解决妨碍Master主机连接到Segment主机的问题。例如，主机可能需要被重启或者替换。
- 在主机上线并且能连接到它后，从Master主机运行gprecoverseg工具来重新激活故障的Segment实例。例如：

```
$ gprecoverseg
```

- 恢复过程会启动失效的Segment并且确定需要同步的已更改文件。该过程可能会花一些时间，请等待该过程结束。在此过程中，数据库的写活动会被禁止。
- 在gprecoverseg完成后，系统会进入到*Resynchronizing*模式并且开始复制更改过的文件。这个过程在后台运行，而系统处于在线状态并且能够接受数据库请求。
- 当重新同步过程完成时，系统状态是*Synchronized*。运行gpstate工具来验证重新同步进程的状态：

## 监控Greenplum系统

```
$ gpstate -m
```

## 要让所有Segment返回到它们的首选角色

当一个主Segment宕掉后，镜像会激活并且成为主Segment。在运行gprecoverseg之后，当前活动的Segment仍是主Segment而失效的Segment变成镜像Segment。这些Segment实例并没有回到在系统初始化时为它们指定的首选角色。这意味着，如果Segment主机上的活动Segment数量超过了让系统性能最优的数量，系统可能处于一种潜在地非平衡状态。要检查非平衡的Segment并且重新平衡系统，运行：

```
$ gpstate -e
```

所有Segment都必须在线并且被完全同步以重新平衡系统。在重新平衡过程中，数据库会话保持连接，但正在进行的查询会被取消并且回滚。

1. 运行gpstate -m来确保所有镜像都是 *Synchronized*。

```
$ gpstate -m
```

2. 如果有任何镜像处于*Resynchronizing*模式，等它们完成。
3. 运行gprecoverseg工具并带有-r选项，让Segment回到它们的首选角色。

```
$ gprecoverseg -r
```

4. 在重新平衡之后，运行gpstate -e来确认所有的Segment都处于它们的首选角色。

```
$ gpstate -e
```

## 从双重故障中恢复

在双重故障中，主Segment和它的镜像都宕掉。如果在不同的Segment主机上同时发生硬件失效，就有可能发生这种情况。如果

发生双重故障，Greenplum数据库会变得不可用。从一次双重故障中恢复的步骤如下：

### 从双重故障中恢复

1. 处理引起双重故障的问题原因并解决，确保segment主机可操作并且可以从master主机访问。
2. 重启Greenplum数据库。`gpstop`工具带有`-r`参数执行时，会停止并重新启动Greenplum 数据库系统。

```
$ gpstop -r
```

3. 在系统重启后，运行`gprecoverseg`重新激活故障segment实例：

```
$ gprecoverseg
```

4. 在`gprecoverseg`完成后，使用`gpstate`检查所有镜像状态，确保segment实例的状态从*Resynchronizing* 模式变成模式：

```
$ gpstate -m
```

5. 如果仍有Segment处于*change tracking*模式，运行`gprecoverseg`并带有`-F`选项执行一次完整恢复。

Warning: 一个完整的恢复会在从活动segment实例（当前主实例）复制数据前，删除离线segment实例的数据目录。在执行全量恢复前，确保segment失败不会引起数据损坏，任何主机的segment磁盘问题都已经被修复。

```
$ gprecoverseg -F
```

6. 如果需要，将segment实例的角色恢复到它们的最佳角色。详见[要让所有Segment返回到它们的首选角色](#)。

## 在没有启用镜像的情况下恢复

1. 确保能够从Master主机连接到该Segment主机。例如：

```
$ ping failed_seg_host_address
```

2. 排查解决妨碍Master主机连接到Segment主机的问题。例如，主机

可能需要被重新启动。

3. 在主机在线之后，验证能够连接到它并且重启Greenplum数据库。gpstop带有-r选项可以停止并重启系统：

```
$ gpstop -r
```

4. 运行gpstate工具来验证所有的实例都在线：

```
$ gpstate
```

- 当一台**Segment**主机不可恢复时

**Parent topic:** [恢复故障Segment](#)

# 时

Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  
- 关于Greenplum数据库发布版本号
  
- 启动和停止Greenplum数据库
  - 访问数据库
  - 配置Greenplum数据库系统
  
- 启用压缩
  
- 启用高可用和数据持久化特征
  - Greenplum数据库高可用性概述
  - 在Greenplum数据库中启用镜像
  - 检测故障的Segment
  - 恢复故障Segment
  
- 从Segment故障中恢复
  - 当一台Segment主机不可恢复时
  - 恢复故障的Master

如果一台主机是不可操作的（例如由于硬件失效导致），就需要把那些Segment恢复到备用的硬件资源上。如果启用了镜像，可以使用[gprecoverseg](#)工具从segment的镜像把它恢复到另一台主机上。

例如：

```
$ gprecoverseg -i recover_config_file
```

其中*recover\_config\_file*的格式是：

```
<failed_host>:<port>:<data_dir>[<recovery_host>:<port>:<recovery_data_dir>]
```

例如，要恢复到不同于失效主机的另一主机上且该没有额外的文件空间配置，（除默认的*pg\_system*文件空间外）：

```
sdw1-1:50001:/data1/mirror/gpseg16 sdw4-1:50001:/data1/recover1/gpseg16
```

有关创建segment实例恢复文件的详细信息，请见[gprecoverseg](#)。

新恢复的Segment主机必须预装好Greenplum数据库软件并且按照现有Segment主机相同的方式配置。

**Parent topic:** [从Segment故障中恢复](#)

当  
一  
台Segment主机  
不  
可  
恢  
复  
时



## Greenplum数据库® 6.0文档

## □ 管理员指南

- Greenplum数据库概念
- 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号

启动和停  
止Greenplum数据库

- 访问数据库
- 配置Greenplum数据库  
系统

启用压缩

- 启用高可用和数据持久  
化特征
- Greenplum数据库高  
可用性概述
- 在Greenplum数据库  
中启用镜像
- 检测故障的Segment
- 恢复故障Segment
- 恢复故障的Master

在恢复后还  
原Master镜像

- 备份和恢复数据库
- 扩容Greenplum系统

在激活一台后备Master进行恢复后，该后备Master会成为主Master。如果后备Master具有和原始Master主机相同的能力和可靠性，可以继续把该实例当作主Master。

必须初始化一个新的后备Master继续提供Master的镜像，除非在激活前一个后备Master时已经这样做了。在活动的Master主机上运行[gpinitstandby](#)来配置一个新的standby Master，详见[启用Master镜像](#)。

可以在原来的主机上恢复主Master和后备Master。这个过程会交换主Master主机和后备Master主机的角色，只有强烈希望在恢复之前的相同主机上运行Master实例时才执行这样的操作。

**Important:** 恢复primary master和standby master实例到他们原始主机并不是一个在线的操作。执行该操作时，master主机必须被停止。

更多有关Greenplum数据库工具的信息，请见[Greenplum数据库工具指南](#)。

## 在原来的主机上恢复Master和后 备Master（可选）

1. 确认原来的Master主机有可靠的运行条件，确保以前的失效原因已  
被修复。
2. 在原来的Master主机上，移动或者移除数据目录gpseg-1。这个  
例子把该目录移动到backup\_gpseg-1：

```
$ mv /data/master/gpseg-1 /data/master/backup_gpseg-1
```

一旦后备被成功地配置，就可以移除备份目录。

3. 在原来的Master主机上初始化一个后备Master。例如，从当前的Master主机 (smdw) 运行这个命令：

```
$ gpinitstandby -s mdw
```

4. 在初始化完成后，检查后备Master (mdw) 的状态，运行  
[gpstate](#)工具带有-f选项来检查standby master状态：

## 监控Greenplum系统

```
$ gpstate -f
```

standby master 状态应该是passive， WAL sender状态应该是streaming。

5. 在后备Master上停止Greenplum数据库的Master实例。例如：

```
$ gpstop -m
```

6. 从原始的Master主机mdw运行gpactivatestandby工具，该主机当前是一个后备Master。例如：

```
$ gpactivatestandby -d $MASTER_DATA_DIRECTORY
```

其中-d选项指定正在激活的主机的数据目录。

7. 在该工具完成后，运行gpstate工具带有-b选项来检查状态：

```
$ gpstate -b
```

验证原来的主Master状态为*Active*。当没有配置一个后备Master时，该命令显示No master standby configured并且该消息表示没有配置一个后备Master实例。

8. 在后备Master主机上，移动或者移除数据目录gpseg-1。这个例子移动该目录：

```
$ mv /data/master/gpseg-1 /data/master/backup_gpseg-1
```

一旦standby master配置成功后便可以移除该备份目录。

9. 在原来的Master主机运行主Greenplum数据库Master之后，可以在原来的后备Master 主机上初始化一个后备Master。例如：

```
$ gpinitstandby -s smdw
```

命令完成后，可以在primary master 主机上运行gpstate -f命令，来检查standby master的状态。

## 检查主Master镜像进程状态（可选）

可以通过运行gpstate工具带有-f选项来显示standby master主机的详细信息。

```
$ gpstate -f
```

standby master状态应该是passive, WAL sender的状态应该是streaming。

更多有关[gpstate](#)工具的信息详见*Greenplum*数据库工具指南。

**Parent topic:** [恢复故障的Master](#)

# 建增量备份

Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统

## 关于Greenplum数据库 发布版本号

## 启动和停止Greenplum数据库

- 访问数据库
  - 配置Greenplum数据库系统
  - 启用压缩
  - 启用高可用和数据持久化特征
  - 备份和恢复数据库

备份和恢复概述

- 使  
用gpbackup和gprestore并  
行备份

需求和限制

## 备份或还原中包含的对象

执行基本备份和还原操作

## 过滤备份或恢复的内容

配置邮件通知

`gpbackup`和`gprestore`工具支持创建追加优化表的增量备份以及从增量备份还原。只有表被更改时，增量备份才会备份所有指定的堆表和追加优化的表（包括追加优化的，面向列的表）。例如，如果追加优化表的行已更改，则会备份该表。对于分区的追加优化表，仅备份更改的叶子分区。

当自上次备份以来，追加优化表或分区表更改的数据与未更改的数据相比量很小的时候，增量备份是高效的。

仅当在上次全量备份或增量备份后对表执行以下操作之一时，增量备份才会备份追加优化表：

- ALTER TABLE
  - DELETE
  - INSERT
  - TRUNCATE
  - UPDATE
  - DROP然后重建表

要从增量备份还原数据，您需要一个完整的增量备份集。

- 关于增量备份集
  - 使用增量备份

## **Parent topic:** 使用gpbackup和gprestore并行备份

## 关于增量备份集

一个增量备份集包含如下备份

- 一个全量的备份。这是增量备份基于的全量备份。
  - 捕获全量备份后数据库的增量备份集。

例如：创建一个全量备份，然后创建三个天级增量备份。全量备份和全部三个增量备份就是备份集。关于增量备份更多信息  参考[使用增量备份集的例子](#)。

创建或添加到增量备份集时，`gpbackup`可确保使用一组一致的备份。

选项创建集合中的备份，以确保可以在还原操作中使用备份集。关于备份集合一致性的信息，参考[使用增量备份](#)。

创建增量备份时，包含这些选项与其他gpbackup选项一起创建备份：

- --leaf-partition-data - 增量备份集中的所有备份都需要。
  - 创建全量备份时必需，该备份将作为增量备份集的基备份。
  - 创建增量备份时必须。
- --incremental - 创建增量备份时必须。  
不能将--data-only或--metadata-only和--incremental一起使用。
- --from-timestamp - 可选的。该选项可以和--incremental一起使用。指定的时间戳是一个已经存在的备份。可以是一个全量备份或增量备份。创建的备份必须和使用--from-timestamp选项指定的备份兼容。  
如果不指定--from-timestamp，gpbackup会尝试基于gpbackup历史文件找一个兼容的备份。参考[增量备份说明](#)。

**Parent topic:** [使用gpbackup和gprestore创建增量备份](#)

## 使用增量备份

- [使用增量备份集的例子](#)
- [使用gpbackup创建增量备份](#)
- [使用gprestore从增量备份恢复](#)
- [增量备份说明](#)

将增量备份添加到备份集时，gpbackup通过检查以下gpbackup选项来确保全量备份和增量备份是一致的：

- --dbname - 数据库必须相同。
- --backup-dir - 文件夹必须相同。备份集，全量和增量备份路径必须相同。
- --single-data-file - 这个选项在集合中所有的备份，要么全部指定，要么全部非指定。
- --plugin-config - 如果指定该选项，则必须在备份集中所有的备份中都指定。配置必须引用相同的二进制插件文件。
- --include-table-file, --include-schema, 或其他过滤表和schema的选项必须相同。  
当检查schema筛选时，只检查schema名字，不检查schema里包含的

对象。

- `--no-compression` - 如果这个选项被指定，必须在备份集中所有的备份中都要指定。  
如果在全量备份上使用压缩，则必须在增量备份上使用压缩。备份集中的备份允许不同的压缩级别。

如果尝试将增量备份添加到备份集，则如果gpbackup选项不一致，则备份操作将失败。

关于gpbackup和gprestore工具选项信息，参考Greenplum数据库工具指南中的[gpbackup](#) 和[gprestore](#) .

## 使用增量备份集的例子

每个备份都有一个创建备份时的时间戳。例如，如果您在2017年5月14日创建备份，则备份文件名包含20170514`hhmmss`。`hhmmss` 表示时间：小时，分钟和秒。

此示例假定您已创建数据库*mytest* 的两个全量备份和增量备份。要创建全量备份，请使用以下命令：

```
gpbackup --dbname mytest --backup-dir /mybackup --leaf-partition-data
```

使用下面命令创建增量备份：

```
gpbackup --dbname mytest --backup-dir /mybackup --leaf-partition-data --incremental
```

当指定`--backup-dir`选项时，备份被创建在每个数据库主机的`/mybackup`目录里。

在该示例中，全量备份具有时间戳  
键20170514054532和20171114064330。其他备份是增量备份。  
该示例包含两个备份集，第一个具有两个增量备份，第二个具有一个  
增量备份。备份从最早到最近列出。

- 20170514054532 (全量备份)
- 20170714095512
- 20170914081205
- 20171114064330 (全量备份)

20180114051246

要基于最新的增量备份创建新的增量备份，必须包含与增量备份相同的--backup-dir选项以及--leaf-partition-data和--incremental选项。

```
gpbackup --dbname mytest --backup-dir /mybackup --leaf-partition-data --incremental
```

您可以指定--from-timestamp选项以基于现有增量备份或全量备份创建增量备份。根据该示例，此命令将第四个增量备份添加到备份集，其中包括20170914081205作为增量备份，并使用20170514054532作为全量备份。

```
gpbackup --dbname mytest --backup-dir /mybackup --leaf-partition-data --incremental --from-timestamp
20170914081205
```

此命令基于全量备份20171114064330创建增量备份集，并与包含增量备份20180114051246的备份集分开。

```
gpbackup --dbname mytest --backup-dir /mybackup --leaf-partition-data --incremental --from-timestamp
20171114064330
```

要使用增量备份20170914081205还原数据库，需要增量备份20120914081205和20170714095512，以及全量备份20170514054532。这将是gprestore命令。

```
gprestore --backup-dir /backupdir --timestamp
20170914081205
```

## 使用gpbackup创建增量备份

gpbackup输出显示增量备份所基于的备份的时间戳。在此示例中，增量备份基于时间戳为20180802171642的备份。备份20180802171642可以是增量备份或全量备份。

```
$ gpbackup --dbname test --backup-dir /backups --leaf-partition-data --incremental
20180803:15:40:51 gpbackup:gpadmin:mdw:002907-[INFO]:-
Starting backup of database test
20180803:15:40:52 gpbackup:gpadmin:mdw:002907-[INFO]:-
Backup Timestamp = 20180803154051
```

```
20180803:15:40:52 gpbackup:gpadmin:mdw:002907-[INFO] :-
Backup Database = test
20180803:15:40:52 gpbackup:gpadmin:mdw:002907-[INFO] :-
Gathering list of tables for backup
20180803:15:40:52 gpbackup:gpadmin:mdw:002907-[INFO] :-
Acquiring ACCESS SHARE locks on tables
Locks acquired: 5 / 5
[=====
100.00% 0s
20180803:15:40:52 gpbackup:gpadmin:mdw:002907-[INFO] :-
Gathering additional table metadata
20180803:15:40:52 gpbackup:gpadmin:mdw:002907-[INFO] :-
Metadata will be written to /backups/gpseg-
1/backups/20180803/20180803154051/gpbackup_20180803154051_met

20180803:15:40:52 gpbackup:gpadmin:mdw:002907-[INFO] :-
Writing global database metadata
20180803:15:40:52 gpbackup:gpadmin:mdw:002907-[INFO] :-
Global database metadata backup complete
20180803:15:40:52 gpbackup:gpadmin:mdw:002907-[INFO] :-
Writing pre-data metadata
20180803:15:40:52 gpbackup:gpadmin:mdw:002907-[INFO] :-
Pre-
data metadata backup complete
20180803:15:40:52 gpbackup:gpadmin:mdw:002907-[INFO] :-
Writing post-data metadata
20180803:15:40:52 gpbackup:gpadmin:mdw:002907-[INFO] :-
Post-
data metadata backup complete
20180803:15:40:52 gpbackup:gpadmin:mdw:002907-
[INFO] :-
Basing incremental backup off of backup with
timestamp = 20180802171642
20180803:15:40:52 gpbackup:gpadmin:mdw:002907-[INFO] :-
Writing data to file
Tables backed up: 4 / 4
[=====
100.00% 0s
20180803:15:40:52 gpbackup:gpadmin:mdw:002907-[INFO] :-
Data
backup complete
20180803:15:40:53 gpbackup:gpadmin:mdw:002907-[INFO] :-
Found
neither /usr/local/greenplum-
db/.bin/gp_email_contacts.yaml nor
/home/gpadmin/gp_email_contacts.yaml
20180803:15:40:53 gpbackup:gpadmin:mdw:002907-[INFO] :-
Email
containing gpbackup report /backups/gpseg-
1/backups/20180803/20180803154051/gpbackup_20180803154051_rep
will not be sent
20180803:15:40:53 gpbackup:gpadmin:mdw:002907-[INFO] :-
Backup completed successfully
```

## 使用gprestore从增量备份恢复

从增量备份还原时，可以指定`--verbose`选项以在命令行上显示还原操作中使用的备份。例如，以下gprestore命令使用时间戳20180807092740（增量备份）还原备份。输出包括用于还原数

## 数据库数据的备份。

```
$ gprestore --create-db --timestamp 20180807162904 --
verbose
...
20180807:16:31:56 gprestore:gpadmin:mdw:008603-[INFO]:-Pre-
data metadata restore complete
20180807:16:31:56 gprestore:gpadmin:mdw:008603-[DEBUG]:-
Verifying backup file count
20180807:16:31:56 gprestore:gpadmin:mdw:008603-
[DEBUG]:-Restoring data from backup with timestamp:
20180807162654
20180807:16:31:56 gprestore:gpadmin:mdw:008603-[DEBUG]:-
Reading data for table public.tbl_ao from file (table 1 of
1)
20180807:16:31:56 gprestore:gpadmin:mdw:008603-[DEBUG]:-
Checking whether segment agents had errors during restore
20180807:16:31:56 gprestore:gpadmin:mdw:008603-
[DEBUG]:-Restoring data from backup with timestamp:
20180807162819
20180807:16:31:56 gprestore:gpadmin:mdw:008603-[DEBUG]:-
Reading data for table public.test_ao from file (table 1 of
1)
20180807:16:31:56 gprestore:gpadmin:mdw:008603-[DEBUG]:-
Checking whether segment agents had errors during restore
20180807:16:31:56 gprestore:gpadmin:mdw:008603-
[DEBUG]:-Restoring data from backup with timestamp:
20180807162904
20180807:16:31:56 gprestore:gpadmin:mdw:008603-[DEBUG]:-
Reading data for table public.homes2 from file (table 1 of
4)
20180807:16:31:56 gprestore:gpadmin:mdw:008603-[DEBUG]:-
Reading data for table public.test2 from file (table 2 of
4)
20180807:16:31:56 gprestore:gpadmin:mdw:008603-[DEBUG]:-
Reading data for table public.homes2a from file (table 3 of
4)
20180807:16:31:56 gprestore:gpadmin:mdw:008603-[DEBUG]:-
Reading data for table public.test2a from file (table 4 of
4)
20180807:16:31:56 gprestore:gpadmin:mdw:008603-[DEBUG]:-
Checking whether segment agents had errors during restore
20180807:16:31:57 gprestore:gpadmin:mdw:008603-[INFO]:-Data
restore complete
20180807:16:31:57 gprestore:gpadmin:mdw:008603-[INFO]:-
Restoring post-data metadata
20180807:16:31:57 gprestore:gpadmin:mdw:008603-[INFO]:-
Post-data metadata restore complete
...
```

输出显示还原操作使用了三个备份。

从增量备份还原时，gprestore还会在gprestore日志文件中列出还原操作中使用的备份。

在还原操作期间，如果全量备份或其他所需的增量备份不可用，`gprestore`将显示错误。

## 增量备份说明

要创建增量备份或从增量备份集还原数据，您需要完整的备份集。存档增量备份时，必须存档完整的备份集。您必须归档在master和所有segment上创建的所有文件。

每次运行`gpbackup`时，该工具都会将备份信息添加到Greenplum数据库master数据目录中的历史文件`gpbackup_history.yaml`。该文件包括备份选项和其他备份信息。

如果在创建增量备份时未指定`--from-timestamp`选项，则`gpbackup`将使用具有一致选项集的最新备份。该工具检查备份历史记录文件以查找具有一致选项集的备份。如果工具找不到具有一致选项集的备份或历史文件不存在，则`gpbackup`会显示一条消息，指出必须先创建全量备份才能创建增量。

如果在创建增量备份时指定`--from-timestamp`选项，则`gpbackup`可确保正在创建的备份选项与指定备份的选项一致。

对于备份集中的所有备份，`gpbackup`选项`--with-stats`不需要相同。但是，要使用`gprestore`选项`--with-stats`执行还原操作以还原统计信息，您指定的备份必须在创建备份时必须使用`--with-stats`。

您可以从备份集中的任何备份执行还原操作。但是，将不会还原在备份用于还原数据库数据之后的增量备份中捕获的更改。

从增量备份集还原时，`gprestore`会检查备份并从备份集中最新版本的追加优化表中还原每个追加优化表，并从最新备份还原堆表。

增量备份集，全量备份和关联的增量备份必须位于单个设备上。例如，备份集中的备份必须全部位于文件系统上，或者必须全部位于Data Domain系统上。

**Warning:** 对Greenplum数据库segment配置的更改会使增量备份无效。更改`segment`配置（添加或删除`segment`实例）后，必须先创建全量备份，然后才能创建增量备份。

**Parent topic:** [使用`gpbackup`和`gprestore`创建增量备份](#)



# 起使用

Greenplum数据库® 6.0 文档

- 管理员指南
- Greenplum数据库概念
- 管理一个Greenplum系统
  - 关于Greenplum数据库  
发布版本号
  - 启动和停止Greenplum数据库
- 访问数据库
- 配置Greenplum数据库  
系统
  - 启用压缩
- 启用高可用和数据持久化特征
- 备份和恢复数据库
  - 备份和恢复概述
    - 使用gpbackup和gprestore并行备份
    - 需求和限制
    - 备份或还原中包含的对象
    - 执行基本备份和还原操作
    - 过滤备份或恢复的内容
    - 配置邮件通知

您可以将Greenplum数据库gpbackup 和gprestore 工具与Data Domain DD Boost文件系统插件 (BoostFS) 一起使用来访问Data Domain系统。 BoostFS利用DD Boost技术，有助于减少带宽使用，可以缩短备份时间，提供负载平衡和机上加密，并支持Data Domain多租户功能集。

您可以在Greenplum数据库主机系统上安装BoostFS插件，以便将Data Domain系统作为标准文件系统安装点进行访问。通过直接访问BoostFS挂载点，gpbackup 和gprestore 可以利用DD Boost协议的存储和网络效率进行备份和恢复。

有关配置BoostFS的信息，可以从Dell支持站点<https://www.dell.com/support> 下载BoostFS for Linux配置指南（需要登录）。登录到支持站点后，您可以通过搜索“BoostFS for Linux配置指南”找到该指南。您可以通过选择仅将“手册和文档”列为资源来限制搜索结果。

要使用BoostFS进行备份或还原，请使用选项--backup-dir和gpbackup 或gprestore 命令来访问Data Domain系统。

- 安装BoostFS
- 使用BoostFS备份和恢复

**Parent topic:** [使用gpbackup和gprestore并行备份](#)

## 安装BoostFS

从Dell支持站点<https://www.dell.com/support> 下载最新的BoostFS RPM（需要登录）。

登录支持站点后，您可以通过搜索“boostfs”找到RPM。您可以选择仅将“下载和驱动程序”列为资源来限制搜索结果。要列出搜索结果顶部附近的最新RPM，请按降序日期对结果进行排序。

RPM支持RHEL和SuSE。

这些步骤安装BoostFS并创建一个访问Data Domain系统的安装目录。

在所有Greenplum数据库主机上执行这些步骤。您创建的安装目录在所有主机上必须相同。

1. 将BoostFS RPM复制到主机并安装RPM。  
安装后，DDBoostFS软件包文件位于/opt/emc/boostfs下。
2. 使用带有boostfs工具的存储单元设置BoostFS密码箱。根据提示输入Data Domain用户密码。

```
/opt/emc/boostfs/bin/boostfs lockbox set -d <Data_Domain_IP> -s <Storage_Unit> -u <Data_Domain_User>
```

<Storage\_Unit>是Data Domain存储单元ID。<Data\_Domain\_User>是可以访  
Data Domain

向存储单元的用户。

- 在要安装BoostFS的位置创建目录。

```
mkdir <path_to_mount_directory>
```

- 使用boostfs工具安装Data Domain存储单元。 使用mount选项--allow-others=true允许其他用户写入BoostFS挂载的文件系统。

```
/opt/emc/boostfs/bin/boostfs mount <path_to_mount_directory> -d $<Data_Domain_IP> -s <Storage_Unit> -o allow-others=true
```

- 运行此命令确认安装成功。

```
mountpoint <mounted_directory>
```

该命令将目录列为安装点。

```
<mounted_directory> is a mountpoint
```

您现在可以使用--backup-dir 选项运行gpbackup 和gprestore ， 以将数据库备份到Data Domain系统上的<mounted\_directory>， 并从Data Domain系统还原数据。

**Parent topic:** [将gpbackup和gprestore与BoostFS一起使用](#)

## 使用BoostFS备份和恢复

使用BoostFS将数据备份到Data Domain系统时，这些是必需的gpbackup 选项。

- --backup-dir - 指定装入的Data Domain存储单元。
- --no-compression - 禁用压缩。 数据压缩会干扰DD Boost重复数据去重。
- --single-data-file - 在每个segment主机上创建单个数据文件。 单个数据文件可避免BoostFS流限制。

使用gprestore 从具有BoostFS的Data Domain系统还原备份时，必须使用选项--backup-dir指定已装入的Data Domain存储单元。

使用gpbackup 选项--single-data-file 时，不能指定--jobs 选项以使用gprestore 执行并行还原操作。

此示例gpbackup 命令备份test 数据库。 该示例假定目录/boostfs-test 是已装入的Data Domain存储单元。

```
$ gpbackup --dbname test --backup-dir /boostfs-test/ --single-data-file --no-compression
```

这些命令将删除test 数据库并从备份中还原数据库。

```
$ dropdb test
$ gprestore --backup-dir /boostfs-test/ --timestamp 20171103153156 --create-db
```

值20171103153156 是要还原的gpbackup 备份集的时间戳。 有关gpbackup 如何

使用时间戳创建备份，参考[使用gpbackup和gprestore并行备份](#)。关于-timestamp 选项的信息，参考[gprestore](#)。

**Parent topic:** [将gpbackup和gprestore与BoostFS一起使用](#)

## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## 关于Greenplum数据库

## 发布版本号

## 启动和停止Greenplum数据库

## □ 访问数据库

## □ 配置Greenplum数据库系统

## 启用压缩

## □ 启用高可用和数据持久化特征

## □ 备份和恢复数据库

## 备份和恢复概述

## □ 使用gpbackup和gprestore并行备份

## 需求和限制

## 备份或还原中包含的对象

## 执行基本备份和还原操作

## 过滤备份或恢复的内容

## 配置邮件通知

您可以配置Greenplum数据库[gpbackup](#)和[gprestore](#)工具，以使用存储插件在备份或还原操作期间处理备份文件。例如，在备份操作期间，插件会将备份文件发送到远程位置。在还原操作期间，插件从远程位置检索文件。

您还可以使用Greenplum数据库备份/还原存储插件API (Beta) 开发自定义存储插件。参考[备份/恢复存储插件API \(Beta版\)](#)。

**Note:** 只有备份/还原存储插件API才是Beta功能。存储插件是受支持的功能。

- [使用带有gpbackup和gprestore的S3存储插件](#)

**Parent topic:** [使用gpbackup和gprestore并行备份](#)



## Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 关于Greenplum数据库  
发布版本号
  - 启动和停  
止Greenplum数据库
  - 访问数据库
  - 配置Greenplum数据库  
系统
  - 启用压缩
  - 启用高可用和数据持久  
化特征
  - 备份和恢复数据库
  - 备份和恢复概述
  - 使用  
gpbackup和gprestore并  
行备份
    - 需求和限制
    - 备份或还原中包含  
的对象
    - 执行基本备份和还  
原操作
    - 过滤备份或恢复的  
内容
    - 配置邮件通知

# 有gpbackup和gprestore的S3存 储插件

S3存储插件应用程序允许您在运行[gpbackup](#)和[gprestore](#)时使  
用Amazon Simple Storage Service (Amazon S3) 位置来存储和检索备份。  
Amazon S3提供安全，持久，高度可扩展的对象存储。

S3存储插件还可以连接到Amazon S3兼容服务，例如[Dell EMC Elastic Cloud Storage](#)  和[Minio](#) 。

要使用S3存储插件应用程序，请在配置文件中指定插件的位置以及S3登录  
和备份位置。运行gpbackup或gprestore时，使用选项--plugin-  
config指定配置文件。有关配置文件的信息，请参阅[S3存储插件配置文  
件格式](#)。

如果使用gpbackup选项--plugin-config执行备份操作，则还必须在  
使用gprestore还原备份时指定--plugin-config选项。

## S3存储插件配置文件格式

配置文件指定Greenplum数据库S3存储插件可执行文件，连接凭据和S3位  
置的绝对路径。

S3存储插件配置文件使用[YAML 1.1](#)  文档格式并实现自己的模式，以指  
定Greenplum数据库S3存储插件的位置，连接凭据以及S3位置和登录信  
息。

配置文件必须是有效的YAML文档。gpbackup和gprestore工具按顺  
序处理控制文件文档，并使用缩进（空格）来确定文档层次结构以及这些  
部分之间的关系。使用空白区域非常重要。不应仅将白色空间用于格式  
化目的，并且根本不应使用制表符。

这是S3存储插件配置文件的结构。

```
executablepath: <absolute-path-to-gpbackup_s3_plugin>
options:
 region: <aws-region>
 endpoint: <S3-endpoint>
 aws_access_key_id: <aws-user-id>
 aws_secret_access_key: <aws-user-id-key>
 bucket: <s3-bucket>
 folder: <s3-location>
 encryption: [on|off]
```

**executablepath**

需要。插件可执行文件的绝对路径。例如，Pivotal Greenplum数据  
库安装位置是\$GPHOME/bin/gpbackup\_s3\_plugin。该插件必  
须位于每个Greenplum数据库主机上的相同位置。

**options**

需要。开始S3存储插件选项部分。

**region**

AWS S3必需。如果连接到S3兼容服务，则不需要此选项。

**endpoint**

S3兼容服务所必需的。指定此选项可连接到S3兼容服务，例  
如ECS。该插件连接到指定的S3端点（主机名或IP地址）以访  
问S3兼容的数据存储。

如果指定了此选项，则插件将忽略region选项，并且不使  
用AWS来解析端点。如果未指定此选项，插件将使  
用region来确定AWS S3端点。

**aws\_access\_key\_id**

可选的。用于访问存储备份文件的S3存储桶位置的S3 ID。

如果未指定此参数，则使用来自会话环境的S3身份验证信息。  
见[备注](#)。

**aws\_secret\_access\_key**

仅在指定aws\_access\_key\_id时才需要。S3 ID的S3密码，  
用于访问S3存储桶位置。

**bucket**

需要。AWS区域或S3兼容数据存储中的S3存储桶的名称。桶  
必须存在。

**folder**

需要。备份的S3位置。在备份操作期间，如果S3存储桶中不存  
在，则插件会创建S3位置。

**encryption**

可选的。连接到S3位置时启用或禁用安全套接字层（SSL）。  
默认值为on，使用通过SSL保护的连接。将此选项设置  
为off以连接到未配置为使用SSL的S3兼容服务。

除off之外的任何值都被视为on。

## 示例

这是在下一个gpbackup示例命令中使用的示例S3存储插件配置文件。该  
文件的名称是s3-test-config.yaml。

```
executablepath: $GPHOME/bin/gpbackup_s3_plugin
options:
 region: us-west-2
 aws_access_key_id: test-s3-user
```

```
aws_secret_access_key: asdf1234asdf
bucket: gpdb-backup
folder: test/backup3
```

这个gpbackup示例使用S3存储插件备份数据库演示。 S3存储插件配置文件的绝对路径是/home/gpadmin/s3-test。

```
gpbackup --dbname demo --plugin-config /home/gpadmin/s3-test-
config.yaml
```

S3存储插件将备份文件写入AWS区域us-west-2中的此S3位置。

```
gpdb-backup/test/backup3/backups/YYYYMMDD/YYYYMMDDHHMMSS/
```

## 备注

S3存储插件应用程序必须位于每个Greenplum数据库主机上的相同位置。仅在master主机上需要配置文件。

使用S3存储插件执行备份时，插件会将备份文件存储在S3存储桶中的此位置。

```
<folder>/backups/<datestamp>/<timestamp>
```

其中`folder`是您在S3配置文件中指定的位置，而`datestamp`和`timestamp`是备份日期和时间戳。

使用Amazon S3备份和还原数据需要Amazon AWS账户可以访问Amazon S3存储桶。 这些是备份和还原数据所需的Amazon S3存储桶权限。

- 上传/删除上传文件的S3用户ID
- 打开/下载和查看访问文件的S3用户ID

如果未在配置文件中指

定`aws_access_key_id`和`aws_secret_access_key`，则S3插件将使用来自运行备份操作的会话的系统环境的S3身份验证信息。 S3插件使用第一个可用源搜索这些源中的信息。

1. 环境变量`AWS_ACCESS_KEY_ID`和`AWS_SECRET_ACCESS_KEY`。
2. 使用AWS CLI命令设置的身份验证信息是`aws configure`。
3. 如果从EC2实例运行备份，则为Amazon EC2 IAM角色的凭据。

有关Amazon S3的信息，参考[Amazon S3](#)。

有关Amazon S3区域和端点的信息，参考[http://docs.aws.amazon.com/general/latest/gr/rande.html#s3\\_region](http://docs.aws.amazon.com/general/latest/gr/rande.html#s3_region)

。

- 有关S3存储桶和文件夹的信息，请参阅Amazon S3文档<https://aws.amazon.com/documentation/s3/>。

**Parent topic:** [使用gpbackup存储插件](#)

# 件API (Beta版)

Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
    - 关于Greenplum数据库  
发布版本号
    - 启动和停止Greenplum数据库
  - 访问数据库
  - 配置Greenplum数据库  
系统
    - 启用压缩
    - 启用高可用和数据持久化特征
    - 备份和恢复数据库
      - 备份和恢复概述
      - 使用gpbackup和gprestore并行备份
        - 需求和限制
        - 备份或还原中包含的对象
        - 执行基本备份和还原操作
        - 过滤备份或恢复的内容
        - 配置邮件通知

本主题介绍如何使用Greenplum数据库备份/还原存储插件API开发自定义存储插件。

**Note:** 只有备份/还原存储插件API才是Beta功能。存储插件是受支持的功能。

备份/恢复存储插件API提供了一个框架，您可以使用该框架开发自定义备份存储系统并将其与Greenplum数据库 [gpbackup](#) 和 [gprestore](#) 工具集成。

备份/恢复存储插件API定义了插件必须支持的一组接口。API还指定插件的配置文件的格式和内容。

使用备份/还原存储插件API时，您将创建Greenplum数据库管理员部署到Greenplum数据库集群的插件。部署后，该插件可用于某些备份和还原操作。

本主题包含如下子主题：

- [插件配置文件](#)
- [插件API](#)
- [插件命令](#)
- [实现一个备份/恢复存储插件](#)
- [验证备份/还原存储插件](#)
- [打包和部署备份/还原存储插件](#)

**Parent topic:** [使用gpbackup和gprestore并行备份](#)

## 插件配置文件

指定gpbackup和gprestore命令的--plugin-config选项会指示工具使用配置文件中指定的插件进行操作。

插件配置文件提供Greenplum数据库和插件的信息。备份/恢复存储插件API定义插件配置文件的格式和某些关键字。

插件的配置文件是一个如下格式的YAML文件：



```
executablepath: path_to_plugin_executable
options:
 keyword1: value1
 keyword2: value2
 ...
 keywordN: valueN
```

gpbackup和gprestore使用**executablepath**值来确定插件可执行程序的文件系统位置。

插件配置文件还可以包括特定于插件实例的关键字和值。 备份/恢复存储插件可以使用文件中指定的选项块来从用户获取执行其任务所需的信息。 例如，该信息可以包括位置，连接或认证信息。 插件应该在**keyword:value**语法中指定和使用此信息的内容。

Greenplum数据库S3备份/恢复存储插件的示例插件配置文件如下：

```
executablepath: $GPHOME/bin/gpbackup_s3_plugin
options:
 region: us-west-2
 aws_access_key_id: notarealID
 aws_secret_access_key: notarealkey
 bucket: gp_backup_bucket
 folder: greenplum_backups
```

## 插件API

您使用备份/恢复存储插件API时实现的插件是一个可执行程序，它支持gpbackup和gprestore在各自生命周期操作中定义的点调用的特定命令：

- Greenplum数据库备份/恢复存储插件API，提供在初始化、备份和清理/退出时挂载到gpbackup生命周期的钩子。
- API在初始化、恢复期间以及清理/退出时提供到gprestore生命周期的钩子。
- API提供了参数，用于指定插件设置或清除命令的执行范围（master主机，segment主机或segment实例）。 范围可以是这些值之一。
  - master - 在master主机上执行一次插件。
  - segment\_host - 在每个segment主机上执行一次插件。
  - segment - 对运行segment实例的主机上的每个活着的segment实例执行一次插件。

备份开始时，Greenplum数据库master和segment实例基

于Greenplum数据库配置。值segment\_host和segment是作为segment主机提供的，可以托管多个segment实例。与每个segment实例相比，segment主机级别可能需要一些设置或清理。

备份/恢复存储插件API为备份/恢复存储插件可执行程序定义以下调用语法：

```
plugin_executable command config_file args
```

其中：

- *plugin\_executable* - 备份/恢复存储插件可执行程序的绝对路径。此路径由插件的配置YAML文件中配置的executablepath属性值确定。
- *command* - 备份/恢复存储插件API命令的名称，用于标识gpbackup或gprestore生命周期操作的特定入口点。
- *config\_file* - 插件的配置YAML文件的绝对路径。
- *args* - 命令参数；实际参数因指定的*command*而异。

## 插件命令

Greenplum数据库备份/恢复存储插件API定义以下命令：

Table 1. 备份/恢复存储插件API命令

| 命令名称                                      | 描述                                    |
|-------------------------------------------|---------------------------------------|
| <a href="#">plugin_api_version</a>        | 返回插件支持的备份/恢复存储插件API的版本。唯一支持的版本是0.3.0。 |
| <a href="#">setup_plugin_for_backup</a>   | 初始化插件以进行备份操作。                         |
| <a href="#">backup_file</a>               | 将备份文件移动到远程存储系统。                       |
| <a href="#">backup_data</a>               | 将流数据从stdin移动到远程存储系统上的文件。              |
| <a href="#">cleanup_plugin_for_backup</a> | 备份操作后清理。                              |
| <a href="#">setup_plugin_for_restore</a>  | 初始化插件以进行还原操作。                         |
| <a href="#">restore_file</a>              | 将备份文件从远程存储系统移动到本地主机上的指定位置。            |
| <a href="#">restore_data</a>              | 从远程存储系统移动备份文件，将数据流式传输到stdout。         |

[cleanup\\_plugin\\_for\\_restore](#)

还原操作后清理。

备份/恢复存储插件必须支持上面标识的每个命令，即使它是无操作。

## 实现一个备份/恢复存储插件

您可以使用任何编程语言或脚本语言实现可执行的备份/恢复存储插件。

备份/恢复存储插件执行的任务将非常特定于远程存储系统。在设计插件实现时，您需要：

- 检查远程存储系统的连接和数据传输接口。
- 确定远程系统的存储路径细节。
- 确定用户所需的配置信息。
- 为插件配置文件中所需的信息定义关键字和值语法。
- 确定插件是否以及如何修改（压缩等）到达/来自远程存储系统的数据。
- 定义gpbackup文件路径和远程存储系统之间的映射。
- 确定gpbackup选项如何影响插件，以及哪些是必需的和/或不适用的。例如，如果插件执行自己的压缩，则必须使用--no-compression选项调用gpbackup以防止工具压缩数据。

您实现的备份/恢复存储插件必须：

- 支持[插件命令](#)中标识的所有插件命令。每个命令都必须使用命令参考页面上标识的值退出。

插件实现的示例，请参考[github仓库gpbackup-s3-plugin](#)。

## 验证备份/还原存储插件

备份/恢复存储插件API包含一个测试平台，您可以运行该平台以确保插件与gpbackup和gprestore良好集成。

测试平台是您在Greenplum数据库安装中运行的bash脚本。该脚本在Greenplum数据库表中生成一个小（<1MB）数据集，显式测试每个命令，并运行数据的备份和恢复（文件和流）。测试平台调用gpbackup和gprestore，后者又单独调用/测试插件中实现的每个备份/还原存储插件API命令。

测试平台程序调用语法是：

```
plugin_test_bench.sh plugin_executable plugin_config
```

## 过程

针对插件运行备份/恢复存储插件API测试平台：

1. 登录Greenplum数据库master主机并设置您的环境。例如：

```
$ ssh gpadmin@<gpmaster>
gpadmin@gpmaster$. /usr/local/greenplum-db/greenplum_path.sh
```

2. 从gpbackupgithub仓库获取测试平台的副本。例如：

```
$ git clone git@github.com:greenplum-db/gpbackup.git
```

克隆操作在当前工作目录中创建名为gpbackup/的目录。

3. 在gpbackup/master/plugins目录中找到测试平台程序。例如：

```
$ ls gpbackup/master/plugins/plugin_test_bench.sh
```

4. 将插件可执行程序和插件配置YAML文件从开发系统复制到Greenplum数据库master主机。记下复制文件的文件系统位置。
5. 将插件可执行程序从Greenplum数据库master主机复制到每个segment主机上的相同文件系统位置。
6. 如果需要，编辑插件配置YAML文件以指定刚刚复制到Greenplum segment的插件可执行程序的绝对路径。
7. 针对插件运行测试平台程序。例如：

```
$ gpbackup/master/plugins/plugin_test_bench.sh
/path/to/pluginexec /path/to/plugincfg.yaml
```

8. 检查测试台输出。如果所有输出消息都显示RUNNING或PASSED，则您的插件会通过测试平台。例如：

```

Starting gpbackup plugin tests

[RUNNING] plugin_api_version
[PASSED] plugin_api_version
[RUNNING] setup_plugin_for_backup
[RUNNING] backup_file
[RUNNING] setup_plugin_for_restore
[RUNNING] restore_file
[PASSED] setup_plugin_for_backup
[PASSED] backup_file
[PASSED] setup_plugin_for_restore
[PASSED] restore_file
[RUNNING] backup_data
[RUNNING] restore_data
[PASSED] backup_data
[PASSED] restore_data
[RUNNING] cleanup_plugin_for_backup
[PASSED] cleanup_plugin_for_backup
[RUNNING] cleanup_plugin_for_restore
[PASSED] cleanup_plugin_for_restore
[RUNNING] gpbackup with test database
[RUNNING] gprestore with test database
[PASSED] gpbackup and gprestore

Finished gpbackup plugin tests

```

## 打包和部署备份/还原存储插件

在插件通过测试和测试台验证后，您的备份/恢复存储插件已准备好部署到Greenplum数据库安装。 打包备份/还原存储插件时，请考虑以下事项：

- 备份/恢复存储插件必须安装在Greenplum数据库集群中每个主机上的相同位置。 提供插件标识相同的安装说明。
- gpadmin用户必须具有遍历备份/恢复插件可执行程序的文件系统路径的权限。
- 在插件中包含模板配置文件。
- 记录有效的插件配置关键字，确保包含期望值的语法。
- 记录gpbackup所需的选项以及它们如何影响插件处理。

## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号

启动和停  
止Greenplum数据库

## □ 访问数据库

□ 配置Greenplum数据库  
系统

启用压缩

□ 启用高可用和数据持久  
化特征

## □ 备份和恢复数据库

备份和恢复概述

□ 使用  
gpbackup和gprestore并  
行备份

需求和限制

备份或还原中包含  
的对象

执行基本备份和还  
原操作

过滤备份或恢复的  
内容

配置邮件通知

插件命令将流数据从stdin移动到远程存储系统。

## 概要

```
plugin_executable backup_data plugin_config_file
data_filenamekey
```

## 描述

gpbackup在流式备份期间在每个segment主机上调用backup\_data插件命令。

backup\_data实现应该从stdin读取可能很大的数据流，并将数据写入远程存储系统上的单个文件。数据作为每个Greenplum数据segment的单个连续流发送到命令。如果backup\_data以任何方式修改数据（即压缩），则restore\_data必须执行撤销操作。

命名或维护从目标文件到data\_filenamekey的映射。这将是用于还原操作的文件密钥。

## 参数

*plugin\_config\_file*

插件配置YAML文件的绝对路径。

*data\_filenamekey*

流数据的特殊命名备份文件的映射关键字。

## 退出码

backup\_data命令必须在成功时以值0退出，如果发生错误则必须为非零。在非零退出代码的情况下，gpbackup向用户显示err的内容。



Greenplum数据库® 6.0文档

- 管理员指南

- Greenplum数据库概念

- 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号

启动和停  
止Greenplum数据库

- 访问数据库

- 配置Greenplum数据库  
系统

启用压缩

- 启用高可用和数据持久  
化特征

- 备份和恢复数据库

备份和恢复概述

- 使用  
gpbackup和gprestore并  
行备份

需求和限制

备份或还原中包含  
的对象

执行基本备份和还  
原操作

过滤备份或恢复的  
内容

配置邮件通知

插件命令将备份文件移动到远程存储系统。

## 概要

```
plugin_executable backup_file plugin_config_file
file_to_backup
```

## 描述

gpbackup在master服务器和每个segment主机上调用*backup\_file*插件命令，以获取gpbackup写入本地磁盘上备份目录的文件。

*backup\_file*实现应该处理文件并将其复制到远程存储系统。不要删除使用*file\_to\_backup*指定的文件的本地副本。

## 参数

*plugin\_config\_file*

插件配置YAML文件的绝对路径。

*file\_to\_backup*

gpbackup生成的本地备份文件的绝对路径。不要删除使  
用*file\_to\_backup*指定的文件的本地副本。

## 退出码

*backup\_file*命令必须在成功时以值0退出，如果发生错误则不为零。在非零退出代码的情况下，gpbackup向用户显示stderr的内容。



Greenplum数据库® 6.0文档

管理员指南

Greenplum数据库概念

管理一个Greenplum系统

关于Greenplum数据库

发布版本号

启动和停止Greenplum数据库

访问数据库

配置Greenplum数据库系统

启用压缩

启用高可用和数据持久化特征

备份和恢复数据库

备份和恢复概述

使用gpbackup和gprestore并行备份

需求和限制

备份或还原中包含的对象

执行基本备份和还原操作

过滤备份或恢复的内容

配置邮件通知

插件命令用于在备份后清理存储插件。

## 概要

```
plugin_executable cleanup_plugin_for_backup
plugin_config_file local_backup_dir scope
```

```
plugin_executable cleanup_plugin_for_backup
plugin_config_file local_backup_dir scope contentID
```

## 描述

gpbackup在gpbackup操作完成时调用cleanup\_plugin\_for\_backup插件命令，包括成功和失败情况。scope参数指定执行范围。gpbackup将使用每个scope值调用该命令。

cleanup\_plugin\_for\_backup命令应执行备份后清理远程存储系统所需的操作。清理活动可能包括删除备份期间创建的远程目录或临时文件，与备份服务断开连接等。

## 参数

*plugin\_config\_file*

插件配置YAML文件的绝对路径。

*local\_backup\_dir*

Greenplum数据库主机（master机和segment）上的本地目录，gpbackup将备份文件写入其中。

- 当*scope*是master时，*local\_backup\_dir*是Greenplum数据库master服务器的备份目录。
- 当*scope*是segment时，*local\_backup\_dir*是segment实例的备份目录。*contentID*标识segment实例。
- 当*scope*是segment\_host时，*local\_backup\_dir*是主机上的任意备份目录。

**scope**

执行*scope*值表示主机和插件命令的执行次数。*scope*可以是以下值之一：

- master - 在master主机上执行一次插件命令。
- segment\_host - 在每个segment主机上执行一次plugin命令。
- segment - 对运行segment实例的主机上的每个活动segment实例执行一次plugin命令。*contentID*标识segment实例。

在首次启动备份时，Greenplum数据库master和segment实例基于Greenplum数据库配置。

**contentID**

与*scope*对应的Greenplum数据库master或segment实例的*contentID*。仅当*scope*为master或segment时才传递*contentID*。

- 当*scope*为master时，*contentID*为-1。
- 当*scope*是segment时，*contentID*是活动segment实例的内容标识符。

## 退出码

`cleanup_plugin_for_backup`命令必须在成功时以值0退出，如果发生错误则必须为非零。在非零退出代码的情况下，`gpbackup`向用户显示`stderr`的内容。

Greenplum数据库® 6.0文档

## □ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号

启动和停  
止Greenplum数据库

□ 访问数据库

□ 配置Greenplum数据库  
系统

启用压缩

□ 启用高可用和数据持久  
化特征

□ 备份和恢复数据库

备份和恢复概述

□ 使  
用gpbackup和gprestore并  
行备份

需求和限制

备份或还原中包含  
的对象

执行基本备份和还  
原操作

过滤备份或恢复的  
内容

配置邮件通知

用于在还原后清理存储插件的插件命令。

## 概要

```
plugin_executable cleanup_plugin_for_restore
plugin_config_file local_backup_dir scope
```

```
plugin_executable cleanup_plugin_for_restore
plugin_config_file local_backup_dir scope contentID
```

## 描述

gprestore在gprestore操作完成时调  
用cleanup\_plugin\_for\_restore插件命令，包括成功和失败情  
况。 **scope**参数指定执行范围。 gprestore将使用每个**scope**值调用  
该命令。

cleanup\_plugin\_for\_restore实现应执行还原后清理远程存储  
系统所需的操作。 清理活动可能包括删除在还原期间创建的远程目录  
或临时文件，与备份服务断开连接等。

## 参数

*plugin\_config\_file*

插件配置YAML文件的绝对路径。

*local\_backup\_dir*

Greenplum数据库主机（master和segment）上的本地目  
录，gprestore从该目录读取备份文件。

- 当**scope**是master时，*local\_backup\_dir*是Greenplum数据  
库master服务器的备份目录。
- 当**scope**是segment时，*local\_backup\_dir*是segment实例的备  
份目录。*contentID*标识segment实例。
- 当**scope**是segment\_host时，*local\_backup\_dir*是主机上的  
任意备份目录。

**scope**

执行*scope*值表示主机和插件命令的执行次数。*scope*可以是以下值之一：

- master - 在master主机上执行一次插件命令。
- segment\_host - 在每个segment主机上执行一次plugin命令。
- segment - 对运行segment实例的主机上的每个活动segment实例执行一次plugin命令。*contentID*标识segment实例。

在首次启动备份时，Greenplum数据库master和segment实例基于Greenplum数据库配置。

**contentID**

与*scope*对应的Greenplum数据库master或segment实例的*contentID*。仅当*scope*为master或segment时才传递*contentID*。

- 当*scope*为master时，*contentID*为-1。
- 当*scope*是segment时，*contentID*是活动segment实例的内容标识符。

## 退出码

`cleanup_plugin_for_restore`命令必须在成功时以值0退出，如果发生错误则必须为非零。在非零退出代码的情况下，`gprestore`向用户显示stderr的内容。

Greenplum数据库® 6.0文档

插件命令显示支持的Backup Storage Plugin API版本。

## □ 管理员指南

- Greenplum数据库概念
- 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号

启动和停  
止Greenplum数据库

- 访问数据库
- 配置Greenplum数据库  
系统

启用压缩

- 启用高可用和数据持久  
化特征
- 备份和恢复数据库

备份和恢复概述

- 使  
用gpbackup和gprestore并  
行备份

需求和限制

备份或还原中包含  
的对象执行基本备份和还  
原操作过滤备份或恢复的  
内容

配置邮件通知

## 概要

```
plugin_executable plugin_api_version
```

## 描述

gpbackup和gprestore在备份或还原操作之前调  
用plugin\_api\_version插件命令，以确定Backup Storage Plugin  
API版本兼容性。

## 返回值

plugin\_api\_version命令必须返回存储插件支持的Backup Storage  
Plugin API版本号，“0.3.0”。



## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号

启动和停  
止Greenplum数据库

## □ 访问数据库

□ 配置Greenplum数据库  
系统

启用压缩

□ 启用高可用和数据持久  
化特征

## □ 备份和恢复数据库

备份和恢复概述

□ 使用  
gpbackup和gprestore并  
行备份

需求和限制

备份或还原中包含  
的对象

执行基本备份和还  
原操作

过滤备份或恢复的  
内容

配置邮件通知

插件命令将数据从远程存储系统流式传输到stdout。

## 概要

```
plugin_executable restore_data plugin_config_file
data_filenamekey
```

## 描述

当还原流式备份时，`gprestore`会在每个segment主机上调用`restore_data`插件命令。

`restore_data`实现应该从远程存储系统读取一个名为或映射到`data_filenamekey`的潜在大型数据文件，并将内容写入`stdout`。如果`backup_data`命令以任何方式（即压缩）修改了数据，则`restore_data`应执行撤销操作。

## 参数

*plugin\_config\_file*

插件配置YAML文件的绝对路径。

*data\_filenamekey*

远程存储系统上备份文件的映射关键字。

`data_filenamekey`与`backup_data`命令提供的密钥相同。

## 返回值

`restore_data`命令必须在成功时以值0退出，如果发生错误则必须为非零。在非零退出代码的情况下，`gprestore`向用户显示`stderr`的内容。





Greenplum数据库® 6.0文档

## □ 管理员指南

- Greenplum数据库概念
- 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号

启动和停  
止Greenplum数据库

- 访问数据库
- 配置Greenplum数据库  
系统

启用压缩

- 启用高可用和数据持久  
化特征
- 备份和恢复数据库

备份和恢复概述

- 使用  
gpbackup和gprestore并  
行备份

需求和限制

备份或还原中包含  
的对象

执行基本备份和还  
原操作

过滤备份或恢复的  
内容

配置邮件通知

插件命令用于从远程存储系统移动备份文件。

## 概要

```
plugin_executable restore_file plugin_config_file
file_to_restore
```

## 描述

gprestore在master服务器和每个segment主机上调用**restore\_file**插件命令，以获取gprestore将从本地磁盘上的备份目录中读取的文件。

**restore\_file**命令应处理文件并将文件从远程存储系统移动到本地主机上的*file\_to\_restore*。

## 参数

*plugin\_config\_file*

插件配置YAML文件的绝对路径。

*file\_to\_restore*

从远程存储系统移动备份文件的绝对路径。

## 返回值

**restore\_file**命令必须在成功时以值0退出，如果发生错误则必须为非零。在非零退出代码的情况下，gprestore向用户显示stderr的内容。



Greenplum数据库® 6.0文档

- 管理员指南

- Greenplum数据库概念

- 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号

启动和停  
止Greenplum数据库

- 访问数据库

- 配置Greenplum数据库  
系统

启用压缩

- 启用高可用和数据持久  
化特征

- 备份和恢复数据库

备份和恢复概述

- 使用  
gpbackup和gprestore并  
行备份

需求和限制

备份或还原中包含  
的对象

执行基本备份和还  
原操作

过滤备份或恢复的  
内容

配置邮件通知

插件命令用于初始化备份操作的存储插件。

## 概要

```
plugin_executable setup_plugin_for_backup
plugin_config_file local_backup_dir scope
```

```
plugin_executable setup_plugin_for_backup
plugin_config_file local_backup_dir scope contentID
```

## Description

gpbackup在gpbackup初始化阶段调用setup\_plugin\_for\_backup插件命令。*scope*参数指定执行范围。gpbackup将使用每个*scope*值调用该命令。

setup\_plugin\_for\_backup命令应在备份开始之前执行初始化远程存储系统所需的活动。设置活动可能包括创建远程目录，验证与远程存储系统的连接，检查磁盘等。

## 参数

*plugin\_config\_file*

插件配置YAML文件的绝对路径。

*local\_backup\_dir*

Greenplum数据库主机（master和segment）上的本地目  
录，gpbackup将写入备份文件。gpbackup创建此本地目录。

- 当*scope*是master时，*local\_backup\_dir*是Greenplum数据  
库master的备份目录。
- 当*scope*是segment时，*local\_backup\_dir*是segment实例的备  
份目录。*contentID*标识segment实例。
- 当*scope*是segment\_host时，*local\_backup\_dir*是segment\_host上  
任意备份目录。

*scope*

执行范围值表示主机和插件命令的执行次数。 *scope*可以是以下值之一：

- *master* - 在master主机上执行一次插件命令。
- *segment\_host* - 在每个segment主机上执行一次plugin命令。
- *segment* - 对运行segment实例的主机上的每个活动segment实例执行一次plugin命令。*contentID*标识segment实例。

在首次启动备份时，Greenplum数据库master和segment实例基于Greenplum数据库配置。

#### *contentID*

与*scope*对应的Greenplum数据库master或segment实例的*contentID*。仅当*scope*为*master*或*segment*时才传递*contentID*。

- 当*scope*值为*master*，*contentID*值为-1。
- 当*scope*值为*segment*，*contentID*值为活动segment实例的内容标识符。

## 返回值

`setup_plugin_for_backup`命令必须在成功时以值0退出，如果发生错误则必须为非零。在非零退出代码的情况下，`gpbackup`向用户显示*stderr*的内容。

## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

关于Greenplum数据库  
发布版本号

启动和停  
止Greenplum数据库

## □ 访问数据库

□ 配置Greenplum数据库  
系统

启用压缩

□ 启用高可用和数据持久  
化特征

## □ 备份和恢复数据库

备份和恢复概述

□ 使用  
gpbackup和gprestore并  
行备份

需求和限制

备份或还原中包含  
的对象

执行基本备份和还  
原操作

过滤备份或恢复的  
内容

配置邮件通知

插件命令用于初始化还原操作的存储插件。

## 概要

```
plugin_executable setup_plugin_for_restore
plugin_config_file local_backup_dir scope
```

```
plugin_executable setup_plugin_for_restore
plugin_config_file local_backup_dir scope contentID
```

## 描述

`gprestore`在`gprestore`初始化阶段调用`setup_plugin_for_restore`插件命令。`scope`参数指定执行范围。`gprestore`将使用每个`scope`值调用该命令。

`setup_plugin_for_restore`命令应在还原操作开始之前执行初始化远程存储系统所需的活动。设置活动可能包括创建远程目录，验证与远程存储系统的连接等。

## 参数

`plugin_config_file`

插件配置YAML文件的绝对路径。

`local_backup_dir`

Greenplum数据库主机（master和segment）上的本地目录，`gprestore`从该目录读取备份文件。`gprestore`创建此本地目录。

- 当`scope`是`master`时，`local_backup_dir`是Greenplum数据库master的备份目录。
- 当`scope`是`segment`时，`local_backup_dir`是`segment`实例的备份目录。`contentID`标识`segment`实例。
- 当`scope`是`segment_host`时，`local_backup_dir`是主机上的任意备份目录。

**scope**

执行范围值表示主机和插件命令的执行次数。 **scope**可以是以下值之一：

- **master** - 在master主机上执行一次插件命令。
- **segment\_host** - 在每个segment主机上执行一次plugin命令。
- **segment** - 对运行segment实例的主机上的每个活动segment实例执行一次plugin命令。**contentID**标识segment实例。

在首次启动备份时，Greenplum数据库master和segment实例基于Greenplum数据库配置。

**contentID**

与**scope**对应的Greenplum数据库master或segment实例的**contentID**。仅当**scope**为**master**或**segment**时才传递**contentID**。

- 当**scope**值为**master**，**contentID**值为-1。
- 当**scope**值为**segment**，**contentID**值为活动segment实例的内容标识符。

## 返回值

`setup_plugin_for_restore`命令必须在成功时以值0退出，如果发生错误则不为零。在非零退出代码的情况下，`gprestore`向用户显示**stderr**的内容。

# instance on single-NIC machine

Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 定义数据库对象
- 分布与倾斜
- 插入, 更新, 和删除数据
  - 查询数据
  - 使用外部数据
  - 定义外部表
- file://协议
- gpfdist://协议
- gpfdists:// 协议
- pxf:// 协议
- s3:// 协议
- 使用自定义协议
- 处理外部表数据中的错误
- 创建和使用外部Web表
- Examples for Creating External Tables

Creates a readable external table, `ext_expenses`, using the `gpfdist` protocol. The files are formatted with a pipe (|) as the column delimiter.

```
=# CREATE EXTERNAL TABLE ext_expenses (name text,
 date date, amount float4, category text, desc1 text)
 LOCATION ('gpfdist://etlhost-1:8081/*')
 FORMAT 'TEXT' (DELIMITER '|');
```

**Parent topic:** [Examples for Creating External Tables](#)



# instances

Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 定义数据库对象
    - 分布与倾斜
    - 插入, 更新, 和删除数据
  - 查询数据
  - 使用外部数据
  - 定义外部表
    - file://协议
    - gpfdist://协议
    - gpfdists:// 协议
    - pxf:// 协议
    - s3:// 协议
    - 使用自定义协议
  - 处理外部表数据中的错误
  - 创建和使用外部Web表
- Examples for Creating External Tables

Creates a readable external table, `ext_expenses`, using the gpfdist protocol from all files with the `.txt` extension. The column delimiter is a pipe ( | ) and NULL ( ' ') is a space.

```
=# CREATE EXTERNAL TABLE ext_expenses (name text,
 date date, amount float4, category text, desc1 text)
LOCATION ('gpfdist://etlhost-1:8081/*.txt',
 'gpfdist://etlhost-2:8081/*.txt')
FORMAT 'TEXT' (DELIMITER '|' NULL ' ');
```

**Parent topic:** [Examples for Creating External Tables](#)



# instances

Greenplum数据库® 6.0文档

- 管理员指南
- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象
  - 分布与倾斜
  - 插入, 更新, 和删除数据
- 查询数据
- 使用外部数据
  - 定义外部表
    - file://协议
    - gpfdist://协议
    - gpfdists:// 协议
    - pxf:// 协议
    - s3:// 协议
    - 使用自定义协议
    - 处理外部表数据中的错误
    - 创建和使用外部Web表
- Examples for Creating External Tables

Creates a readable external table, `ext_expenses`, from all files with the `.txt` extension using the `gpfdists` protocol. The column delimiter is a pipe (|) and NULL (' ') is a space. For information about the location of security certificates, see [gpfdists:// 协议](#).

1. Run `gpfdist` with the `--ssl` option.

2. Run the following command.

```
=# 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 ' ') ;
```

**Parent topic:** [Examples for Creating External Tables](#)



# instance with error logging

Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 定义数据库对象
  - 分布与倾斜
  - 插入, 更新, 和删除数据
    - 查询数据
    - 使用外部数据
    - 定义外部表
  - file://协议
  - gpfdist://协议
  - gpfdists:// 协议
  - pxf:// 协议
  - s3:// 协议
  - 使用自定义协议
  - 处理外部表数据中的错误
  - 创建和使用外部Web表
- Examples for Creating External Tables

Uses the gpfdist protocol to create a readable external table, `ext_expenses`, from all files with the `.txt` extension. The column delimiter is a pipe ( | ) and NULL ( ' ') is a space.

Access to the external table is single row error isolation mode. Input data formatting errors are captured internally in Greenplum Database with a description of the error. See [在错误日志中查看不正确的行](#) for information about investigating error rows. You can view the errors, fix the issues, and then reload the rejected data. If the error count on a segment is greater than five (the `SEGMENT REJECT LIMIT` value), the entire external table operation fails and no rows are processed.

```
=# 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 SEGMENT REJECT LIMIT 5;
```

To create the readable `ext_expenses` table from CSV-formatted text files:

```
=# 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 SEGMENT REJECT LIMIT 5;
```

**Parent topic:** [Examples for Creating External Tables](#)



# on a Hadoop Distributed File Server

Greenplum数据库® 6.0文档

## □ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 定义外部表

file://协议

gpfdist://协议

gpfdists:// 协议

pxf:// 协议

s3:// 协议

使用自定义协议

处理外部表数据中的  
错误

创建和使用外  
部Web表

□ Examples for  
Creating External  
Tables

Creates a readable external table, `ext_expenses`, using the `pxf` protocol. The column delimiter is a pipe ( | ).

```
=# CREATE EXTERNAL TABLE ext_expenses (name text,
 date date, amount float4, category text, desc1 text)
 LOCATION ('pxf://dir/data/filename.txt?
 PROFILE=hdfs:text')
 FORMAT 'TEXT' (DELIMITER '|');
```

Refer to [Accessing External Data with PXF](#) for information about using the Greenplum Platform Extension Framework (PXF) to access data on a Hadoop Distributed File System.

**Parent topic:** [Examples for Creating External Tables](#)



# CSV format with header rows

Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 定义数据库对象
- 分布与倾斜
- 插入, 更新, 和删除数据
  - 查询数据
  - 使用外部数据
  - 定义外部表
- file://协议
- gpfdist://协议
- gpfdists:// 协议
- pxf:// 协议
- s3:// 协议
- 使用自定义协议
- 处理外部表数据中的错误
- 创建和使用外部Web表
- Examples for Creating External Tables

Creates a readable external table, `ext_expenses`, using the `file` protocol. The files are CSV format and have a header row.

```
=# CREATE EXTERNAL TABLE ext_expenses (name text,
 date date, amount float4, category text, desc1 text)
LOCATION ('file:///filehost/data/international/*',
 'file:///filehost/data/regional/*',
 'file:///filehost/data/supplement/*.csv')
FORMAT 'CSV' (HEADER);
```

**Parent topic:** [Examples for Creating External Tables](#)



Greenplum数据库® 6.0文档

- 管理员指南

- Greenplum数据库概念

- 管理一个Greenplum系统

- 管理Greenplum数据库访问

- 定义数据库对象

- 分布与倾斜

- 插入, 更新, 和删除数据

- 查询数据

- 使用外部数据

- 定义外部表

file://协议

gpfdist://协议

gpfdists:// 协议

pxf:// 协议

s3:// 协议

使用自定义协议

处理外部表数据中的  
错误

创建和使用外  
部Web表

- Examples for  
Creating External  
Tables

# External Web Table with Script

Creates a readable external web table that executes a script once per segment host:

```
=# CREATE EXTERNAL WEB TABLE log_output (linenum int,
 message text)
EXECUTE '/var/load_scripts/get_log_data.sh' ON HOST
FORMAT 'TEXT' (DELIMITER '|');
```

**Parent topic:** [Examples for Creating External Tables](#)



# External Table with gpfdist

Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 定义数据库对象
    - 分布与倾斜
    - 插入, 更新, 和删除数据
    - 查询数据
    - 使用外部数据
    - 定义外部表
      - file://协议
      - gpfdist://协议
      - gpfdists:// 协议
      - pxf:// 协议
      - s3:// 协议
      - 使用自定义协议
    - 处理外部表数据中的错误
    - 创建和使用外部Web表
  - Examples for Creating External Tables

Creates a writable external table, *sales\_out*, that uses gpfdist to write output data to the file *sales.out*. The column delimiter is a pipe (|) and NULL (' ') is a space. The file will be created in the directory specified when you started the gpfdist file server.

```
=# CREATE WRITABLE EXTERNAL TABLE sales_out (LIKE sales)
 LOCATION ('gpfdist://etl1:8081/sales.out')
 FORMAT 'TEXT' (DELIMITER '|' NULL ' ')
 DISTRIBUTED BY (txnid);
```

**Parent topic:** [Examples for Creating External Tables](#)



# External Web Table with Script

Greenplum数据库® 6.0文档

## □ 管理员指南

- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

- 查询数据
- 使用外部数据
- 定义外部表

file://协议

gpfdist://协议

gpfdists:// 协议

pxf:// 协议

s3:// 协议

使用自定义协议

处理外部表数据中的  
错误

创建和使用外  
部Web表

- Examples for  
Creating External  
Tables

Creates a writable external web table, `campaign_out`, that pipes output data received by the segments to an executable script, `to_adreport_etl.sh`:

```
=# CREATE WRITABLE EXTERNAL WEB TABLE campaign_out
 (LIKE campaign)
 EXECUTE '/var/unload_scripts/to_adreport_etl.sh'
 FORMAT 'TEXT' (DELIMITER '|');
```

**Parent topic:** [Examples for Creating External Tables](#)



Greenplum数据库® 6.0文档

- 管理员指南

- Greenplum数据库概念

- 管理一个Greenplum系统

- 管理Greenplum数据库访问

- 定义数据库对象

- 分布与倾斜

- 插入, 更新, 和删除数据

- 查询数据

- 使用外部数据

- 定义外部表

- file://协议

- gpfldist://协议

- gpfdists:// 协议

- pxf:// 协议

- s3:// 协议

- 使用自定义协议

- 处理外部表数据中的  
错误

- 创建和使用外  
部Web表

- Examples for  
Creating External  
Tables

# Writable External Tables with XML Transformations

Greenplum Database can read and write XML data to and from external tables with gpfdist. For information about setting up an XML transform, see [使用gpfdist和gupload转换外部数据](#).

**Parent topic:** [Examples for Creating External Tables](#)



## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

## □ 使用外部数据

## □ 装载和卸载数据

## 使用外部表装载数据

## □ 装载和写入非HDFS自定义数据

## □ 使用一种自定义格式

## 导入和导出固定宽度的数据

例子：读取宽度固定的数据

## 使用一种自定协议

## □ 处理装载错误

用gupload装载数据

使用PXF访问外部数据

用Greenplum数据库函数fixedwidth\_in和fixedwidth\_out为固定宽度的数据指定自定义格式。这些函数已经存在于文件\$GPHOME/share/postgresql/cdb\_external\_extensions.sql中。下面的例子声明一个自定格式，然后调用fixedwidth\_in函数来格式化数据。

```
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');
```

下列选项指定如何导入固定宽度的数据。

- 读取所有数据。

要装载一行固定宽度数据上的所有域，用户必须按照它们的物理顺序装载它们。用户必须指定域长度，但不能指定开始和结束位置。固定宽度参数中的域名称必须匹配CREATE TABLE命令开头的域列表中的顺序。

- 设置空白和空值字符的选项。

T拖尾的空白会被默认修剪掉。要保留拖尾的空白，使用preserve\_blanks=on选项。用户可以用preserve\_blanks=off选项重置拖尾空白选项为默认值。使用null='null\_string\_value'选项来为空值字符指定一个值。

- 如果用户指定preserve\_blanks=on，用户还必须为空值字符定义一个值。
- 如果用户指定preserve\_blanks=off、空值没有定义并且域只包含空白，Greenplum会写一个空值到表中。如果定义了空值，Greenplum会写一个空串到表中。使用line\_delim='line\_ending'参数来指定行结束字符串。下面的例子覆盖了大部分情况。E指定了一个转义字符串常量。

```
line_delim=E'\n'
line_delim=E'\r'
line_delim=E'\r\n'
line_delim='abc'
```

**Parent topic:** [使用一种自定义格式](#)







## Greenplum数据库® 6.0文档

 管理员指南 Greenplum数据库概念 管理一个Greenplum系统 管理Greenplum数据库访问 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

 查询数据 使用外部数据 装载和卸载数据

## 使用外部表装载数据

 装载和写入非HDFS自定义数据 使用一种自定义格式

## 导入和导出固定宽度的数据

**例子：读取宽度固定的数据**

## 使用一种自定协议

 处理装载错误

## 用gupload装载数据

## 使用PXF访问外部数据

下面的例子展示了如何读取宽度固定的数据。

## 例 1 – 装载一个所有域都定义好的表

```
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 – 装载一个打开了PRESERVED\_BLANKS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 – 装载没有行定界符的数据

```
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', line_delim='?@')
```

## 例 4 – 用\r\n行定界符创建一个可写的外部表

使用gpfldist和gpload转

```
CREATE WRITABLE EXTERNAL TABLE students_out (
 name varchar(20), address varchar(30), age int)
LOCATION ('gpfldist://<host>:<portNum>/file/path/students_out.txt')
FORMAT 'CUSTOM' (formatter=fixedwidth_out,
 name=20, address=30, age=4, line_delim=E'\r\n');
```

**Parent topic:** [使用一种自定义格式](#)

# 定义一个带有单行错误隔离的外部表

Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 定义数据库对象
- 分布与倾斜
- 插入, 更新, 和删除数据
- 查询数据
- 使用外部数据
- 装载和卸载数据
  - 使用外部表装载数据
  - 装载和写入非HDFS自定义数据
  - 处理装载错误

定义一个带有单行错误隔离的外部表

捕捉行格式化错误并且声明拒绝极限

在错误日志中查看不正确的行

在表之间移动数据

用gupload装载数据

使用PXF访问外部数据

下面的例子在Greenplum数据库内部记录错误并且设置错误阈值为10个错误。

```
=# CREATE EXTERNAL TABLE ext_expenses (name text,
 date date, amount float4, category text, desc1 text)
LOCATION ('gpfdist://etlhost-1:8081/*',
 'gpfdist://etlhost-2:8082/*')
FORMAT 'TEXT' (DELIMITER '|')
LOG ERRORS SEGMENT REJECT LIMIT 10
ROWS;
```

使用内建的SQL函

数`gp_read_error_log('external_table')`来读取错误日志数据。这个例子命令显示了`ext_expenses`的日志错误：

```
SELECT gp_read_error_log('ext_expenses');
```

有关错误日志格式的信息, 请见[在错误日志中查看不正确的行](#)

内建SQL函数`gp_truncate_error_log('external_table')`删除错误数据。这个例子删除从上前一个外部表例子创建的错误日志数据：

```
SELECT gp_truncate_error_log('ext_expenses');
```

**Parent topic:** [处理装载错误](#)



# 拒绝极限

Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 定义数据库对象
  - 分布与倾斜
  - 插入, 更新, 和删除数据
  - 查询数据
  - 使用外部数据
  - 装载和卸载数据

下面的SQL片段会捕捉Greenplum数据库内部的格式化错误并且声明一个拒绝极限为10行。

```
LOG ERRORS SEGMENT REJECT LIMIT 10 ROWS
```

使用内建的SQL函数`gp_read_error_log()`可以读取错误日志数据。有关查看日志错误的信息，可见[在错误日志中查看不正确的行](#)

**Parent topic:** [处理装载错误](#)

使用外部表装载数据

- 装载和写入非HDFS自定义数据
- 处理装载错误

定义一个带有单行错误隔离的外部表

**捕捉行格式化错误并且声明拒绝极限**

在错误日志中查看不正确的行

在表之间移动数据

用`gupload`装载数据

使用PXF访问外部数据



# 行

Greenplum数据库® 6.0文档

- 管理员指南
  - Greenplum数据库概念
  - 管理一个Greenplum系统
  - 管理Greenplum数据库访问
  - 定义数据库对象
    - 分布与倾斜
    - 插入, 更新, 和删除数据
  - 查询数据
  - 使用外部数据
  - 装载和卸载数据
    - 使用外部表装载数据
      - 装载和写入非HDFS自定义数据
      - 处理装载错误
    - 定义一个带有单行错误隔离的外部表
    - 捕捉行格式化错误并且声明拒绝极限
  - 在错误日志中查看不正确的行**
  - 在表之间移动数据
  - 用gupload装载数据
  - 使用PXF访问外部数据

如果用户使用了单行错误隔离（见[定义一个带有单行错误隔离的外部表](#)或者[在单行错误隔离模式中运行COPY](#)），任何由格式错误的行会被Greenplum数据库内部记录下来。

Greenplum数据库用一种表格式捕捉下列错误信息：

**Table 1. 错误日志格式**

| 列        | 类型          | 描述                                                                                                               |
|----------|-------------|------------------------------------------------------------------------------------------------------------------|
| cmdtime  | timestamptz | 错误发生时的时间戳。                                                                                                       |
| relname  | text        | COPY命令的外部表名称或者目标表名称。                                                                                             |
| filename | text        | 包含该错误的装载文件的名称。                                                                                                   |
| linenum  | int         | 如果使用的是COPY，这里是错误发生在装载文件的行号。对于使用file://协议或者gpfdist://协议以及CSV格式的外部表，文件名和行号会被记录。                                    |
| byt enum | int         | 对于使用gpfdist://协议以及TEXT格式数据的外部表：错误发生在装载文件中的字节偏移。gpfdist按块解析TEXT文件，因此不可能记录行号。<br><br>CSV文件是一次解析一行，因此对于CSV文件可以跟踪行号。 |
| errmsg   | text        | 错误消息文本。                                                                                                          |
| rawdata  | text        | 被拒绝行的裸数据。                                                                                                        |
| rawbytes | bytea       | 在有数据库编码错误（使用的客户端编码不能被转换成一种服务器端编码）的情况下，不可能把编码错误记录为rawdata。相反会存储裸字节，并且将看到任何非七位ASCII字符的十进制码。                        |

Greenplum

SQL

用户可以使用数据库的内建函数`gp_read_error_log()`来显示内部记录的格式化错误。例如，这个命令显示表`ext_expenses`的错误日志信息：

```
SELECT gp_read_error_log('ext_expenses');
```

有关管理被内部记录的格式化错误的信息，请见*Greenplum*数据库参考指南中命令`COPY`或者`CREATE EXTERNAL TABLE`的部分。

**Parent topic:** [处理装载错误](#)

## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

## □ 使用外部数据

## □ 装载和卸载数据

## 使用外部表装载数据

□ 装载和写入非HDFS自  
定义数据

## □ 处理装载错误

定义一个带有单行错  
误隔离的外部表捕捉行格式化错误并  
且声明拒绝极限在错误日志中查看不  
正确的行

## 在表之间移动数据

## 用gupload装载数据

## 使用PXF访问外部数据

用户可以使用CREATE TABLE AS或者INSERT...SELECT把外部表和外部Web表的数据装载到另一个（非外部）数据库表中，并且这些数据会被根据外部表或者外部Web表定义并行装载。

如果一个外部表文件或者外部Web表数据源有错误，根据使用的隔离模式将会发生下列情况之一：

- 没有错误隔离模式的表：任何从该表中读取的操作都会失败。从没有错误隔离模式的外部表或者外部Web表装载是一种全做或者全不做的操作。
- 有错误隔离模式的表：整个文件都将被装载，除了其中有问题的行（服从于配置的REJECT\_LIMIT）。

**Parent topic:** [处理装载错误](#)



## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

## □ 使用外部数据

## □ 装载和卸载数据

## 使用外部表装载数据

□ 装载和写入非HDFS自  
定义数据

## □ 处理装载错误

## 用gupload装载数据

## 使用PXF访问外部数据

使用gpfdist和gupload转  
换外部数据

## 用COPY装载数据

在单行错误隔离模式中  
运行COPY优化数据装载和查询性  
能

使用Greenplum并行文件服务器程序gpfdist 或者Greenplum平台扩展框架PXF (Greenplum与Hadoop的接口) 输出数据到文件的可写外部表。

使用CREATE WRITABLE EXTERNAL TABLE命令定义外部表并且指定输出文件的位置和格式。为配合外部表使用, 设置gpfdist 的指导可见[使用Greenplum的并行文件服务器 \(gpfdist\)](#) 以及设置PXF以与外部表一起使用的指导可见[..../external/pxf-overview.html](#)

- 对于一个使用gpfdist 协议的可写外部表, Greenplum的Segment把它们的数据发送到gpfdist , 后者会把数据写到指定的文件中。gpfdist 必须运行在一个Greenplum的Segment能够通过网络访问的主机上。gpfdist 指向一个输出主机上的位置并且把从Greenplum的Segment接收的数据写到该文件中。要把输出数据划分到多个文件之间, 在用户的可写外部表定义中列出多个gpfdist URI。
- 一个可写外部Web表把数据当作数据流发送给一个应用。例如, 从Greenplum数据库卸载数据并且把它发送到一个连接到另一数据的应用或者把该数据装载到其他地方的ETL工具。可写外部Web表使用EXECUTE子句来指定一个shell命令、脚本或者应用运行在Segment主机上并且接受一个输入数据流。更多有关在可写外部表定义中使用EXECUTE的信息请见[定义基于命令的可写外部Web表](#)

用户可以有选择地为用户的可写外部表声明一个分布策略。默认情况下, 可写外部表使用一种随机分布策略。如果从中导出数据的源表具有一种哈希分布策略, 为可写外部表定义相同的分布键列可以提高卸载性能, 因为这消除了在Interconnect之上移动行的需求。如果用户从一个特定表卸载数据, 用户可以使用LIKE子句从源表复制列定义和分布策略。

- [例 1—Greenplum文件服务器 \(gpfdist\)](#)
- [例 2—Hadoop文件系统\(pxf\)](#)

**Parent topic:** [从Greenplum数据库卸载数据](#)







# (gpfdist)

Greenplum数据库® 6.0文档

- 管理员指南
- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

- 查询数据
- 使用外部数据
- 装载和卸载数据

使用外部表装载数据

- 装载和写入非HDFS自定义数据
- 处理装载错误

用gupload装载数据

使用PXF访问外部数据

使用gpfdist和gupload转换外部数据

用COPY装载数据

在单行错误隔离模式中运行COPY

优化数据装载和查询性能

```
=# 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);
```

**Parent topic:** [定义基于文件的可写外部表](#)







## Greenplum数据库® 6.0文档

## □ 管理员指南

□ Greenplum数据库概念

□ 管理一个Greenplum系统

□ 管理Greenplum数据库访问

□ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

□ 查询数据

□ 使用外部数据

□ 装载和卸载数据

## 使用外部表装载数据

□ 装载和写入非HDFS自  
定义数据

□ 处理装载错误

## 用gupload装载数据

## 使用PXF访问外部数据

使用gpfldist和gupload转  
换外部数据

## 用COPY装载数据

在单行错误隔离模式中  
运行COPY优化数据装载和查询性  
能

```
=# CREATE WRITABLE EXTERNAL TABLE unload_expenses
 (LIKE expenses)
 LOCATION ('pxf://dir/path?PROFILE=hdfs:text')
 FORMAT 'TEXT' (DELIMITER ',')
 DISTRIBUTED BY (exp_id);
```

用户为使用pxf协议创建的可写外部表指定HDFS目录。

**Parent topic:** 定义基于文件的可写外部表





# 外部表

Greenplum数据库® 6.0文档

- 管理员指南
- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

- 查询数据
- 使用外部数据
- 装载和卸载数据
- 使用外部表装载数据

□ 装载和写入非HDFS自  
定义数据

□ 处理装载错误

用gupload装载数据

使用PXF访问外部数据

使用gpfldist和gupload转  
换外部数据

用COPY装载数据

在单行错误隔离模式中  
运行COPY优化数据装载和查询性  
能

□ 从Greenplum数据库卸

用户可以定义可写的外部Web表把输出行发送到一个应用或者脚本。该应用必须接受一个输入流、在所有的Greenplum的Segment主机上位于相同位置并且对gpadmin用户可执行。Greenplum系统中的所有Segment都运行该应用或者脚本，不管Segment有没有输出行需要处理。

使用CREATE WRITABLE EXTERNAL WEB TABLE来定义外部表并且制定在Segment主机上运行的应用或者脚本。命令从数据库内部执行并且不能访问环境变量（例如\$PATH）。在用户的可写外部表定义的EXECUTE子句中设置环境变量。例如：

```
=# CREATE WRITABLE EXTERNAL WEB TABLE output (output text)
 EXECUTE 'export PATH=$PATH:/home/gpadmin
 /programs;
 myprogram.sh'
 FORMAT 'TEXT'
 DISTRIBUTED RANDOMLY;
```

下面的Greenplum数据库变量可以用在一个Web或者可写外部表所执行的OS命令中。把这些变量设置为执行命令的shell中的环境变量。它们可以被用来标识外部表语句在Greenplum数据库的主机和Segment实例阵列之间所作的一个要求集合。

Table 1. 外部表EXECUTE变量

| 变量                | 描述                                      |
|-------------------|-----------------------------------------|
| \$GP_CID          | 执行外部表命令的事务的命令计数。                        |
| \$GP_DATABASE     | 外部表定义所在的数据库。                            |
| \$GP_DATE         | 外部表命令被运行的日期。                            |
| \$GP_MASTER_HOST  | 分派外部表语句的Greenplum的Master主机的主机名。         |
| \$GP_MASTER_PORT  | 分派外部表语句的Greenplum的Master主机的端口号。         |
| \$GP_QUERY_STRING | Greenplum数据库执行的SQL命令(DML或SQL查询)。        |
| \$GP_SEG_DATADIR  | 执行外部表命令的Segment的数据目录的位置。                |
| \$GP_SEG_PG_CONF  | 执行外部表命令的Segment实例的postgresql.conf文件的位置。 |

|                    |                                                               |
|--------------------|---------------------------------------------------------------|
| \$GP_SEG_PORT      | 执行外部表命令的Segment的端口号。                                          |
| \$GP_SEGMENT_COUNT | Greenplum数据库系统中主要Segment实例的总数。                                |
| \$GP_SEGMENT_ID    | 执行外部表命令的Segment实例的ID号<br>(与gp_segment_configuration中的dbid一样)。 |
| \$GP_SESSION_ID    | 与外部表语句相关的数据库会话标识号。                                            |
| \$GP_SN            | 外部表语句执行计划中外部表扫描节点的序号。                                         |
| \$GP_TIME          | 外部表命令被执行的时间。                                                  |
| \$GP_USER          | 执行外部表语句的用户。                                                   |
| \$GP_XID           | 外部表语句的事务ID。                                                   |

- 为Web或者可写外部表禁用EXECUTE

**Parent topic:** [从Greenplum数据库卸载数据](#)

# 用EXECUTE

Greenplum数据库® 6.0文档

- 管理员指南
- Greenplum数据库概念
- 管理一个Greenplum系统
- 管理Greenplum数据库访问
- 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

- 查询数据
- 使用外部数据
- 装载和卸载数据

使用外部表装载数据

- 装载和写入非HDFS自定义数据
- 处理装载错误

用gupload装载数据

使用PXF访问外部数据

使用gpfdist和gupload转换外部数据

用COPY装载数据

在单行错误隔离模式中运行COPY

优化数据装载和查询性能

允许外部表执行OS命令或者脚本会有安全性风险。要在Web以及可写外部表定义中禁用EXECUTE, 在用户的Master的postgresql.conf文件中设置配置参数gp\_external\_enable\_exec server为off:

```
gp_external_enable_exec = off
```

**Parent topic:** [定义基于命令的可写外部Web表](#)







## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

## □ 查询数据

## □ 使用外部数据

## □ 装载和卸载数据

使用外部表装载数据

□ 装载和写入非HDFS自  
定义数据

## □ 处理装载错误

用gupload装载数据

使用PXF访问外部数据

使用gpfdist和gupload转  
换外部数据

用COPY装载数据

在单行错误隔离模式中  
运行COPY优化数据装载和查询性  
能

可写外部表只允许INSERT操作。不是表拥有者或者超级用户的用户必须被授予表上的INSERT权限。例如：

```
GRANT INSERT ON writable_ext_table TO admin;
```

要使用可写外部表卸载数据，从源表选择数据并且将它插入到可写外部表中。结果行被输出到可写外部表。例如：

```
INSERT INTO writable_ext_table SELECT * FROM regular_table;
```

**Parent topic:** [从Greenplum数据库卸载数据](#)







## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

## □ 使用外部数据

## □ 装载和卸载数据

## 使用外部表装载数据

□ 装载和写入非HDFS自  
定义数据

## □ 处理装载错误

## 用gupload装载数据

## 使用PXF访问外部数据

使用gpfdist和gupload转  
换外部数据

## 用COPY装载数据

在单行错误隔离模式中  
运行COPY优化数据装载和查询性  
能

COPY TO在Greenplum的Master主机上用一个Master实例的单进程从一个表复制数据到一个文件（或者标准输入）。使用COPY输出一个表的全部内容，或者使用一个SELECT语句过滤输出。例如：

```
COPY (SELECT * FROM country WHERE country_name LIKE 'A%')
TO '/home/gpadmin/a_list_countries.out';
```

**Parent topic:** [从Greenplum数据库卸载数据](#)







## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

## □ 使用外部数据

## □ 装载和卸载数据

## 使用外部表装载数据

□ 装载和写入非HDFS自  
定义数据

## □ 处理装载错误

## 用gupload装载数据

## 使用PXF访问外部数据

使用gpfldist和gupload转  
换外部数据

## 用COPY装载数据

在单行错误隔离模式中  
运行COPY优化数据装载和查询性  
能

Greenplum数据期望数据中的行被LF字符（换行, 0x0A）、CR（回车, 0x0D）或者后面跟着LF的CR（CR+LF, 0x0D 0x0A）所分隔。在UNIX或者类UNIX操作系统上, LF是标准的新行表示。而Windows或者Mac OS X等操作系统试用CR或者CR+LF。Greenplum数据库支持所有这些新行的表示形式来作为行界定符。更多信息请见[导入和导出固定宽度的数据](#)。

**Parent topic:** [格式化数据文件](#)







## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

## □ 使用外部数据

## □ 装载和卸载数据

## 使用外部表装载数据

□ 装载和写入非HDFS自  
定义数据

## □ 处理装载错误

## 用gupload装载数据

## 使用PXF访问外部数据

使用gpfldist和gupload转  
换外部数据

## 用COPY装载数据

在单行错误隔离模式中  
运行COPY优化数据装载和查询性  
能

默认的列或者域定界符对于文本文件是水平TAB字符 (0x09) , 对于CSV文件是逗号字符 (0x2C) 。在定义数据格式时, 用户可以使用COPY、CREATE EXTERNAL TABLE或者gupload的DELIMITER子句声明一个单字符定界符。定界符字符必须出现在任意两个数据值域之间。不要在行的首尾放置定界符。例如, 如果竖线字符 (|) 是定界符:

```
data value 1|data value 2|data value 3
```

下列命令展示了竖线字符作为列定界符的使用:

```
=# CREATE EXTERNAL TABLE ext_table (name text, date date)
LOCATION ('gpfdist://<hostname>/filename.txt')
FORMAT 'TEXT' (DELIMITER '|');
```

**Parent topic:** 格式化数据文件







## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

## □ 使用外部数据

## □ 装载和卸载数据

## 使用外部表装载数据

□ 装载和写入非HDFS自  
定义数据

## □ 处理装载错误

## 用gupload装载数据

## 使用PXF访问外部数据

使用gpfdist和gupload转  
换外部数据

## 用COPY装载数据

在单行错误隔离模式中  
运行COPY优化数据装载和查询性  
能

NULL标识列或者域中一片未知的数据。在用户的数据文件中，用户可以指定一个字符串来表示空值。在TEXT模式中默认的字符串是\N（反斜线加上N），在CSV模式中是一个没有引用的空值。用户也可以在定义数据格式时用COPY、CREATE EXTERNAL TABLE 或者gupload的NULL子句声明一个不同的字符串。例如，如果用户不想区分空值和空字符串，用户可以使用一个空字符串。在使用Greenplum数据库的装载工具时，任何匹配指定的空值串的数据项都会被认为是一个空值。

**Parent topic:** 格式化数据文件







## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

## □ 使用外部数据

## □ 装载和卸载数据

## 使用外部表装载数据

□ 装载和写入非HDFS自  
定义数据

## □ 处理装载错误

## 用gupload装载数据

## 使用PXF访问外部数据

使用gpfldist和gupload转  
换外部数据

## 用COPY装载数据

在单行错误隔离模式中  
运行COPY优化数据装载和查询性  
能

用户可以在COPY、CREATE EXTERNAL TABLE 或者gupload的ESCAPE子句中声明一个不同的转义字符。如果用户的转义字符出现在用户的数据中，用它来转义自身。

例如，假定用户有一个具有三列的表并且用户想载入下列三个域：

- backslash = \
- vertical bar = |
- exclamation point = !

用户指派的分隔字符是|（竖线字符）并且用户指派的转义字符是\（反斜线）。用户的数据文件中的已格式化行看起来像这样：

```
backslash = \\ | vertical bar = \| | exclamation point = !
```

注意是如何用另一个反斜线字符转义作为数据一部分的反斜线字符和竖线字符的。

用户可以使用转义字符来转义十进制和十六进制序列。在被载入到Greenplum数据库时，被转义的值会被转换成等效的字符。例如，要载入花号字符（&），使用转义字符来转义其等效的十六进制（\0x26）或者十进制（\046）表达。

用户可以这样使用COPY、CREATE EXTERNAL TABLE 或者gupload的ESCAPE子句来禁用TEXT格式文件中的转义：

```
ESCAPE 'OFF'
```

这对于包含很多反斜线字符的输入数据（例如Web日志数据）有用。

**Parent topic:** [转义](#)







## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

分布与倾斜

插入, 更新, 和删除数据

## □ 查询数据

## □ 使用外部数据

## □ 装载和卸载数据

使用外部表装载数据

□ 装载和写入非HDFS自  
定义数据

## □ 处理装载错误

用gupload装载数据

使用PXF访问外部数据

使用gpfldist和gupload转  
换外部数据

用COPY装载数据

在单行错误隔离模式中  
运行COPY优化数据装载和查询性  
能

默认情况下，对于CSV格式的文件，转义字符是一个"（双引号）。如果用户想用一个不同的转义字符，使用COPY、CREATE EXTERNAL TABLE或者gpload的ESCAPE子句来声明一个不同的转义字符。在用户选择的转义字符出现在数据的情况下，用户可以使用它来转义自身。

例如，假定用户有一个带有三列的表并且用户想载入下列的三个域：

- Free trip to A,B
- 5.89
- Special rate "1.79"

用户指派的定界符是，（逗号），并且用户指派的转义字符是"（双引号）。用户的数据文件中的已格式化行看起来像这样：

```
"Free trip to A,B", "5.89", "Special rate ""1.79"""
```

带有一个作为数据一部分的逗号字符的数据值被包裹在双引号中。作为数据一部分的双引号被用一个双引号转义，即使该域值已经被双引号包裹也应如此。

在一组双引号内嵌入整个域确保能保留下前导和拖尾的空白字符：

```
"Free trip to A,B ", "5.89 ", "Special rate ""1.79 " "
```

**Note:** 在CSV模式中，所有字符都是有意义的。一个被空白或者任何不同于DELIMITER的字符包围的引用值包括这些字符在内。如果用户导入的数据来自于一个会在CSV行中用空白填充到某种固定宽度的系统，这可能会导致错误。在这种情况下，可预处理CSV文件，在把该数据导入到Greenplum数据之前移除拖尾的空白。

**Parent topic:** 转义







## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

## □ 使用外部数据

## □ 装载和卸载数据

## 使用外部表装载数据

## □ 装载和写入非HDFS自定义数据

## □ 处理装载错误

## 用gupload装载数据

## 使用PXF访问外部数据

## 使用gpfldist和gupload转换外部数据

## 用COPY装载数据

## 在单行错误隔离模式中运行COPY

## 优化数据装载和查询性能

字符编码系统由将来自于一种字符集的每一个字符与其他某个东西（例如一个数字序列或者八位组序列）配对的代码构成，它们能让数据的传输和存储便利。Greenplum数据库支持多种字符集，包括ISO 8859系列等单字节字符集和EUC、UTF-8、Mule内部编码等多字节字符集。服务器端字符集在数据库初始化期间被定义，UTF-8是默认字符集但可以更改。客户端可以透明地使用所有受支持的字符集，但有一部分字符集不支持作为服务器端编码在服务器内使用。在把数据载入或者插入到Greenplum数据库中时，Greenplum透明地把数据从指定的客户端编码转换成服务器编码。在把数据送回到客户端时，Greenplum会把数据从服务器字符编码转换成指定的客户端编码。

数据文件必须使用一种Greenplum数据库能识别的字符编码。受支持的字符集请见*Greenplum*数据库参考指南。包含非法或者不受支持的编码序列的数据文件会在装载到Greenplum数据库时遇到错误。

**Note:** 在微软Windows操作系统生成的数据文件上，在载入到Greenplum数据库之前运行 dos2unix系统命令来移除任何Windows字符。

**Parent topic:** [格式化数据文件](#)

## 更改客户端字符编码

可以通过设置服务器配置参数`client_encoding`来改变一个会话的客户端字符编码。

```
SET client_encoding TO 'latin1';
```

将客户端字符编码更改回默认值：

```
RESET client_encoding;
```

显示当前客户端字符编码设置：

```
SHOW client_encoding;
```







## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

## □ 使用外部数据

## □ 装载和卸载数据

## 使用外部表装载数据

## □ 装载和写入非HDFS自定义数据

## □ 处理装载错误

## 用gupload装载数据

## 使用PXF访问外部数据

## 使用gpfldist和gupload转换外部数据

## 用COPY装载数据

## 在单行错误隔离模式中运行COPY

## 优化数据装载和查询性能

要使用例子中的外部表协议，用户需要用C编译器cc来编译和链接源代码，创建出一个能被Greenplum数据库动态载入的共享对象。在Linux系统上编译和链接该源代码的命令看起来与此类似：

```
cc -fpic -c gpextprotocol.c cc -shared -o gpextprotocol.so gpextprotocol.o
```

选项-fpic指定创建位置无关代码 (PIC)，-c选项只编译而不链接源代码并且创建一个对象文件。该对象文件需要被创建为位置无关代码 (PIC)，这样Greenplum数据库才能把它载入到内存中的任意位置。

标志-shared指定创建一个共享对象 (共享库)，-o选项指定该共享对象文件的名称为gpextprotocol.so。更多有关cc选项的信息请参考GCC手册。

gpextprotocol.c中声明为包括文件的头文件位于\$GPHOME/include/postgresql/的子目录中。

更多有关编译及链接动态载入函数的例子和其他操作系统上编译C源代码创建共享库的例子，请见<https://www.postgresql.org/docs/8.4/xfunc-c.html#DFUNC> 上的Postgres文档。

用户操作系统上的C编译器cc以及链接编辑器ld的手册页也包含有关在该系统上编译和链接源代码的信息。

自定义协议的已编译代码 (共享对象文件) 在用户的Greenplum数据库阵列中每一个主机 (Master和所有的Segment) 上必须被放置在相同的位置。这个位置还必须在LD\_LIBRARY\_PATH中，这样服务器才能定位该文件。推荐在Greenplum数据库阵列的所有Master和Segment实例上都将共享文件放置在相对于\$libdir (位于\$GPHOME/lib) 的位置或者动态库路径 (由dynamic\_library\_path服务器配置参数设置) 中的位置。用户可以使用Greenplum数据库工具gpssh 和gpscp来更新Segment。

- [gpextprotocol.c](#)

**Parent topic:** [自定义数据访问协议实例](#)







## Greenplum数据库® 6.0文档

## □ 管理员指南

## □ Greenplum数据库概念

## □ 管理一个Greenplum系统

## □ 管理Greenplum数据库访问

## □ 定义数据库对象

## 分布与倾斜

## 插入, 更新, 和删除数据

## □ 查询数据

## □ 使用外部数据

## □ 装载和卸载数据

## 使用外部表装载数据

## □ 装载和写入非HDFS自定义数据

## □ 处理装载错误

## 用gupload装载数据

## 使用PXF访问外部数据

## 使用gpfldist和gupload转换外部数据

## 用COPY装载数据

## 在单行错误隔离模式中运行COPY

## 优化数据装载和查询性能

## 从Greenplum数据库卸

```

#include "postgres.h"
#include "fmgr.h"
#include "funcapi.h"
#include "access/extprotocol.h"
#include "catalog/pg_proc.h"
#include "utils/array.h"
#include "utils/builtins.h"
#include "utils/memutils.h"

/* Our chosen URI format. We can change it however needed */
typedef struct DemoUri
{
 char *protocol;
 char *path;
} DemoUri;

static DemoUri *ParseDemoUri(const char *uri_str);
static void FreeDemoUri(DemoUri* uri);

/* Do the module magic dance */
PG_MODULE_MAGIC;
PG_FUNCTION_INFO_V1(demoprot_export);
PG_FUNCTION_INFO_V1(demoprot_import);
PG_FUNCTION_INFO_V1(demoprot_validate_urls);

Datum demoprot_export(PG_FUNCTION_ARGS);
Datum demoprot_import(PG_FUNCTION_ARGS);
Datum demoprot_validate_urls(PG_FUNCTION_ARGS);

/* A user context that persists across calls. Can be
declared in any other way */
typedef struct {
 char *url;
 char *filename;
 FILE *file;
} extprotocol_t;
/*
 * The read function - Import data into GPDB.
 */
Datum
myprot_import(PG_FUNCTION_ARGS)
{
 extprotocol_t *myData;
 char *data;
 int datlen;
 size_t nread = 0;

 /* Must be called via the external table format manager */
 if (!CALLED_AS_EXTPROTOCOL(fcinfo))
 elog(ERROR, "myprot_import: not called by external
protocol manager");
}

```

```

 /* Get our internal description of the protocol */
 myData = (extprotocol_t *)
EXTPROTOCOL_GET_USER_CTX(fcinfo);

 if(EXTPROTOCOL_IS_LAST_CALL(fcinfo))
 {
 /* we're done receiving data. close our connection */
 if(myData && myData->file)
 if(fclose(myData->file))
 ereport(ERROR,
 (errcode_for_file_access(),
 errmsg("could not close file \"%s\": %m",
 myData->filename)));
 }

 PG_RETURN_INT32(0);
}

if (myData == NULL)
{
 /* first call. do any desired init */

 const char *p_name = "myprot";
 DemoUri *parsed_url;
 char *url = EXTPROTOCOL_GET_URL(fcinfo);
 myData = palloc(sizeof(extprotocol_t));

 myData->url = pstrdup(url);
 parsed_url = ParseDemoUri(myData->url);
 myData->filename = pstrdup(parsed_url->path);

 if(strcasecmp(parsed_url->protocol, p_name) != 0)
 elog(ERROR, "internal error: myprot called with a
different protocol (%s)",
 parsed_url->protocol);

 FreeDemoUri(parsed_url);

 /* open the destination file (or connect to remote
server in
 other cases) */
 myData->file = fopen(myData->filename, "r");

 if (myData->file == NULL)
 ereport(ERROR,
 (errcode_for_file_access(),
 errmsg("myprot_import: could not open file
\"%s\""
 for reading: %m",
 myData->filename),
 errOmitLocation(true)));
}

EXTPROTOCOL_SET_USER_CTX(fcinfo, myData);
}
/* =====*
 * DO THE IMPORT
 * ===== */
data = EXTPROTOCOL_GET_DATABUF(fcinfo);

```

```

datlen = EXTPROTTOCOL_GET_DATALEN(fcinfo);

/* read some bytes (with fread in this example, but
normally
 in some other method over the network) */
if(datlen > 0)
{
 nread = fread(data, 1, datlen, myData->file);
 if (ferror(myData->file))
 ereport(ERROR,
 (errcode_for_file_access(),
 errmsg("myprot_import: could not write to file
 \"%s\": %m",
 myData->filename)));
}
PG_RETURN_INT32((int)nread);
}

/*
 * Write function - Export data out of GPDB
 */
Datum
myprot_export(PG_FUNCTION_ARGS)
{
 extprotocol_t *myData;
 char *data;
 int datlen;
 size_t wrote = 0;

 /* Must be called via the external table format manager
 */
 if (!CALLED_AS_EXTPROTTOCOL(fcinfo))
 elog(ERROR, "myprot_export: not called by external
 protocol manager");

 /* Get our internal description of the protocol */
 myData = (extprotocol_t *)
EXTPROTTOCOL_GET_USER_CTX(fcinfo);
 if(EXTPROTTOCOL_IS_LAST_CALL(fcinfo))
 {
 /* we're done sending data. close our connection */
 if(myData && myData->file)
 if(fclose(myData->file))
 ereport(ERROR,
 (errcode_for_file_access(),
 errmsg("could not close file \"%s\": %m",
 myData->filename)));
 }

 PG_RETURN_INT32(0);
}
if (myData == NULL)
{
 /* first call. do any desired init */
 const char *p_name = "myprot";
 DemoUri *parsed_url;
 char *url = EXTPROTTOCOL_GET_URL(fcinfo);

 myData = palloc(sizeof(extprotocol_t));
}

```

```

 myData->url = pstrdup(url);
 parsed_url = ParseDemoUri(myData->url);
 myData->filename = pstrdup(parsed_url->path);

 if(strcasecmp(parsed_url->protocol, p_name) != 0)
 elog(ERROR, "internal error: myprot called with a
 different protocol (%s)",
 parsed_url->protocol);

 FreeDemoUri(parsed_url);

 /* open the destination file (or connect to remote
 server in
 other cases) */
 myData->file = fopen(myData->filename, "a");
 if (myData->file == NULL)
 ereport(ERROR,
 (errcode_for_file_access(),
 errmsg("myprot_export: could not open file
 \"%s\""
 for writing: %m",
 myData->filename),
 errOmitLocation(true)));

 EXTPROTTOCOL_SET_USER_CTX(fcinfo, myData);
 }
/* =====
 * DO THE EXPORT
 * ===== */
data = EXTPROTTOCOL_GET_DATABUF(fcinfo);
datlen = EXTPROTTOCOL_GET_DATALEN(fcinfo);

if(datlen > 0)
{
 wrote = fwrite(data, 1, datlen, myData->file);

 if (ferror(myData->file))
 ereport(ERROR,
 (errcode_for_file_access(),
 errmsg("myprot_import: could not read from file
 \"%s\": %m",
 myData->filename)));
}

PG_RETURN_INT32((int)wrote);
}
Datum
myprot_validate_urls(PG_FUNCTION_ARGS)
{
 List *urls;
 int nurls;
 int i;
 ValidatorDirection direction;

 /* Must be called via the external table format manager
 */
 if (!CALLED_AS_EXTPROTOCOL_VALIDATOR(fcinfo))
 elog(ERROR, "myprot_validate_urls: not called by

```

```

 external
 protocol manager");

 nurls = EXTPROTOCOL_VALIDATOR_GET_NUM_URLS(fcinfo);
 urls = EXTPROTOCOL_VALIDATOR_GET_URL_LIST(fcinfo);
 direction =
EXTPROTOCOL_VALIDATOR_GET_DIRECTION(fcinfo);
/*
 * Dumb example 1: search each url for a substring
 * we don't want to be used in a url. in this example
 * it's 'secured_directory'.
 */
for (i = 1 ; i <= nurls ; i++)
{
 char *url = EXTPROTOCOL_VALIDATOR_GET_NTH_URL(fcinfo,
i);

 if (strstr(url, "secured_directory") != 0)
 {
 ereport(ERROR,
 (errcode(ERRCODE_PROTOCOL_VIOLATION),
 errmsg("using 'secured_directory' in a url
isn't allowed ")));
 }
}
/*
 * Dumb example 2: set a limit on the number of urls
 * used. In this example we limit readable external
 * tables that use our protocol to 2 urls max.
 */
if(direction == EXT_VALIDATE_READ && nurls > 2)
{
 ereport(ERROR,
 (errcode(ERRCODE_PROTOCOL_VIOLATION),
 errmsg("more than 2 urls aren't allowed in this
protocol ")));
}
PG_RETURN_VOID();
}
/* --- utility functions --- */
static
DemoUri *ParseDemoUri(const char *uri_str)
{
 DemoUri *uri = (DemoUri *) palloc0(sizeof(DemoUri));
 int protocol_len;

 uri->path = NULL;
 uri->protocol = NULL;
/*
 * parse protocol
 */
 char *post_protocol = strstr(uri_str, "://");

 if(!post_protocol)
 {
 ereport(ERROR,
 (errcode(ERRCODE_SYNTAX_ERROR),
 errmsg("invalid protocol URI \\'%s\\' ", uri_str)),
}
}

```

```
 errOmitLocation(true)));
 }

 protocol_len = post_protocol - uri_str;
 uri->protocol = (char *)palloc0(protocol_len + 1);
 strncpy(uri->protocol, uri_str, protocol_len);

 /* make sure there is more to the uri string */
 if (strlen(uri_str) <= protocol_len)
 ereport(ERROR,
 (errcode(ERRCODE_SYNTAX_ERROR),
 errmsg("invalid myprot URI \\'%s\\' : missing path",
 uri_str),
 errOmitLocation(true)));

 /* parse path */
 uri->path = pstrdup(uri_str + protocol_len +
 strlen("://"));

 return uri;
}
static
void FreeDemoUri(DemoUri *uri)
{
 if (uri->path)
 pfree(uri->path);
 if (uri->protocol)
 pfree(uri->protocol);

 pfree(uri);
}
```

---

**Parent topic:** [安装外部表协议](#)

## Greenplum数据库® 6.0文档

## □ 参考指南

## □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

## □ 系统目录参考

系统表

系统视图

## □ 系统目录定义

foreign\_data\_wrapper\_options

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

`gp_toolkit.gp_resgroup_config`视图允许管理员查看资源组的当前CPU, 内存和并发限制。该视图还显示建议的限制设置。当限制被更改时, 建议的限制将不同于当前限制, 但新值不能立即应用。

Note: 仅当基于资源组的资源管理处于活动状态时, `gp_resgroup_config`视图才有效。

Table 1. `gp_toolkit.gp_resgroup_config`

| 列                            | 类型   | 参考                                                                                                     | 描述                                     |
|------------------------------|------|--------------------------------------------------------------------------------------------------------|----------------------------------------|
| groupid                      | oid  | <code>pg_resgroup.oid</code>                                                                           | 资源组的OID。                               |
| groupname                    | name | <code>pg_resgroup.rsgname</code>                                                                       | 资源组的名称。                                |
| concurrency                  | text | <code>pg_resgroupcapability.value</code> for<br><code>pg_resgroupcapability.reslimittype = 1</code>    | 为资源组指定的并发 (CONCURRENCY) 值。             |
| proposed_concurrency         | text | <code>pg_resgroupcapability.proposed</code> for<br><code>pg_resgroupcapability.reslimittype = 1</code> | 资源组的挂起并发值。                             |
| cpu_rate_limit               | text | <code>pg_resgroupcapability.value</code> for<br><code>pg_resgroupcapability.reslimittype = 2</code>    | 为资源组指定的CPU限制 (CPU_RATE_LIMIT) 值, 或-1。  |
| memory_limit                 | text | <code>pg_resgroupcapability.value</code> for<br><code>pg_resgroupcapability.reslimittype = 3</code>    | 为资源组指定的内存限制 (MEMORY_LIMIT) 值。          |
| proposed_memory_limit        | text | <code>pg_resgroupcapability.proposed</code> for<br><code>pg_resgroupcapability.reslimittype = 3</code> | 资源组的挂起内存限制值。                           |
| memory_shared_quota          | text | <code>pg_resgroupcapability.value</code> for<br><code>pg_resgroupcapability.reslimittype = 4</code>    | 为资源组指定的共享内存配额 (MEMORY_SHARED_QUOTA) 值。 |
| proposed_memory_shared_quota | text | <code>pg_resgroupcapability.proposed</code> for<br><code>pg_resgroupcapability.reslimittype = 4</code> | 资源组的挂起共享内存配额值。                         |
| memory_spill_ratio           | text | <code>pg_resgroupcapability.value</code> for<br><code>pg_resgroupcapability.reslimittype = 5</code>    | 为资源组指定的内存溢出率 (MEMORY_SPILL_RATIO) 值。   |
| proposed_memory_spill_ratio  | text | <code>pg_resgroupcapability.proposed</code> for                                                        | 资源组的待处理内存溢出比率                          |

|                |      |                                                                        |                   |
|----------------|------|------------------------------------------------------------------------|-------------------|
|                |      | pg_resgroupcapability.reslimittype >= 5                                | 值。                |
| memory_auditor | text | pg_resgroupcapability.value for pg_resgroupcapability.reslimittype = 6 | 用于资源组的内存审计器。      |
| cpuset         | text | pg_resgroupcapability.value for pg_resgroupcapability.reslimittype = 7 | 为资源组保留的CPU核心，或-1。 |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_roles`视图提供关于数据库角色的信息。这是`pg_authid`的一个公共可读视图，它隐去了口令域。此视图显示了低层表的OID列，因为需要它来和其他目录做连接。

Table 1. pg\_catalog.pg\_roles

| 列             | 类型   | 参考 | 描述                                      |
|---------------|------|----|-----------------------------------------|
| rolname       | name |    | 角色名                                     |
| rolsuper      | bool |    | 角色是否具有超级用户权限                            |
| rolinherit    | bool |    | 如果此角色是另一个角色的成员，角色是否能自动继承另一个角色的权限        |
| rolcreaterole | bool |    | 角色能否创建更多角色                              |
| rolcreatedb   | bool |    | 角色能否创建数据库                               |
| rolcatupdate  | bool |    | 角色能够更新直接更新系统目录（除非该列设置为真，否则超级用户也不能执行该操作） |
| rolcanlogin   | bool |    | 角色是否能登录？即此角色能否被作为初始会话授权标识符              |
| rolconnlimit  | int4 |    | 对于一个可登录的角色，这里设置角色可以同时的最大并发连接数。-1表示无限制。  |
| rolpassword   | text |    | 不是口令（看起来                                |

|                   |             |                 | 是*****)                                        |
|-------------------|-------------|-----------------|------------------------------------------------|
| rolvaliduntil     | timestamptz |                 | 口令失效时间<br>(只用于口令<br>认证) , 如果<br>永不失效则<br>为NULL |
| rolconfig         | text[]      |                 | 运行时配置变<br>量的角色特定<br>默认值                        |
| rolresqueue       | oid         | pg_resqueue.oid | 该角色指定的<br>资源队列的对<br>象ID                        |
| oid               | oid         | pg_authid.oid   | 角色的ID                                          |
| rolcreaterextgpf  | bool        |                 | 角色能够创建<br>使用gpfdist协<br>议的可读的外<br>部表           |
| rolcreaterexthttp | bool        |                 | 角色能够创建<br>使用http协议<br>的可读的外部<br>表              |
| rolcreatewextgpf  | bool        |                 | 角色能否创建<br>使用<br>了gpfdist协议<br>的可写外部表           |
| rolresgroup       | oid         | pg_resgroup.oid | 分配了此角色<br>的资源组的对<br>象ID                        |

**Parent topic:** [系统目录定义](#)

## Greenplum数据库® 6.0文档

## □ 参考指南

## □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

## □ 系统目录参考

系统表

系统视图

## □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

Note: 仅当基于资源组的资源管理处于活动状态时, pg\_resgroup系统目录表才有效。

pg\_resgroup系统目录表包含有关Greenplum数据库资源组的信息, 这些资源组用于管理并发语句, CPU和内存资源。此表在pg\_global表空间中定义, 在系统中的所有数据库之间全局共享。

Table 1. pg\_catalog.pg\_resgroup

| 列       | 类型   | 参考 | 描述           |
|---------|------|----|--------------|
| rsgname | name |    | 资源组的名称。      |
| parent  | oid  |    | 未使用; 留作将来使用。 |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_options

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_progress

gpexpand.status

视图`pg_stat_activity`每行显示一个服务器进程同时详细描述与之关联的用户会话和查询。这些列报告当前查询上可用的数据，除非参数`stats_command_string`被关闭。此外，只有在检查视图的用户是超级用户或者是正在报告的进程的拥有者时，这些列才可见。

列`current_query`中存储的查询文本字符串的最大长度可以通过服务器配置参数`track_activity_query_size`来控制。

Table 1. pg\_catalog.pg\_stat\_activity

| 列                             | 类型                       | 参考                           | 描述                                                                                                                                                                                                          |
|-------------------------------|--------------------------|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>datid</code>            | <code>oid</code>         | <code>pg_database.oid</code> | 数据库OID                                                                                                                                                                                                      |
| <code>datname</code>          | <code>name</code>        |                              | 数据库名称                                                                                                                                                                                                       |
| <code>pid</code>              | <code>integer</code>     |                              | 服务进程的进程ID                                                                                                                                                                                                   |
| <code>sess_id</code>          | <code>integer</code>     |                              | 会话ID                                                                                                                                                                                                        |
| <code>usesysid</code>         | <code>oid</code>         | <code>pg_authid.oid</code>   | 登录此后的用户的OID                                                                                                                                                                                                 |
| <code>username</code>         | <code>name</code>        |                              | 登录到此后的用户的名称                                                                                                                                                                                                 |
| <code>application_name</code> | <code>text</code>        |                              | 连接到此后的应用程序的名称                                                                                                                                                                                               |
| <code>client_addr</code>      | <code>inet</code>        |                              | 连接到此后的客户端的IP地址。如果此字段为空，则表示客户端通过服务器计算机上的Unix套接字连接，或者这是内部进程（如autovacuum）。                                                                                                                                     |
| <code>client_hostname</code>  | <code>text</code>        |                              | 客户端的主机名，由 <code>client_addr</code> 的反向DNS查找报告。对于IP连接，此字段仅为非null，并且仅在启用 <code>log_hostname</code> 时才为空。                                                                                                      |
| <code>client_port</code>      | <code>integer</code>     |                              | 客户端用于与此后端通信的TCP端口号，如果使用Unix套接字，则为-1                                                                                                                                                                         |
| <code>backend_start</code>    | <code>timestamptz</code> |                              | 后端启动时间                                                                                                                                                                                                      |
| <code>xact_start</code>       | <code>timestamptz</code> |                              | 事务开始时间                                                                                                                                                                                                      |
| <code>query_start</code>      | <code>timestamptz</code> |                              | 查询开始执行时间                                                                                                                                                                                                    |
| <code>state_change</code>     | <code>timestamptz</code> |                              | 状态最后一次改变的时间                                                                                                                                                                                                 |
| <code>waiting</code>          | <code>boolean</code>     |                              | 如果等待一个锁为True，否则为false                                                                                                                                                                                       |
| <code>state</code>            | <code>text</code>        |                              | 此后的当前整体状态。可能的值是： <ul style="list-style-type: none"><li>• <code>active</code>: 后端正在执行查询。</li><li>• <code>idle</code>: 后端正在等待新的客户端命令。</li><li>• <code>idle in transaction</code>: 后端处于事务中，但当前未执行查询。</li></ul> |

|                  |          |                     |                                                                                                                                                                                                                                       |
|------------------|----------|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                  |          |                     | <ul style="list-style-type: none"> <li><b>idle in transaction (aborted)</b>: 此状态类似于事务中的空闲，除了事务中的一个语句导致错误。</li> <li><b>fastpath function call</b>: 后端正在执行快速路径功能。</li> <li><b>disabled</b>: 如果在此后端禁用track_activities，则报告此状态。</li> </ul> |
| query            | text     |                     | 此后端的最新查询的文本。如果状态为活跃，则此字段显示当前正在执行的查询。在所有其他状态中，它显示最后执行的查询。                                                                                                                                                                              |
| waiting_reason   | text     |                     | 服务器进程正在等待的原因。值可以是：lock, replication或resgroup                                                                                                                                                                                          |
| rsgid            | oid      | pg_resgroup.oid     | 资源组OID                                                                                                                                                                                                                                |
| rsgname          | text     | pg_resgroup.rsgname | 资源组名称                                                                                                                                                                                                                                 |
| rsgqueueduration | interval |                     | 对于排队查询，查询排队的总时间。                                                                                                                                                                                                                      |

**Parent topic:** [系统目录定义](#)

□ 工具指南

□ 管理工具参考

analyzedb

gpactivatestandby

gpaddmirrors

gpbackup

gpcheck

gpcheckcat

gpcheckperf

gpconfig

gpdeletesystem

gpexpand

gpfdist

gpinitstandby

gpinitsystem

gupload

gplogfilter

gpmapreduce

gpmovemirrors

gpperfmon\_install

gppkg

## 移动镜像实例到新位置

## 概要

```
gpmovemirrors -i move_config_file [-d master_data_directory]
[-l logfile_directory]
[-B parallel_processes] [-v]
```

```
gpmovemirrors -?
```

```
gpmovemirrors --version
```

## 描述

`gpmovemirrors`工具将镜像实例移动到新位置。客户可能会想移动镜像到新位置以优化分布 和数据存储

在移动Segment之前，该工具会验证Mirror是否存在，他们对应的主节点是否在线和目前同步状态在同步还是 异步模式。

默认情况下，工具会提示您输入所有镜像要移动到的文件系统路径。

客户必须保证运行`gpmovemirrors`的用户（`gpadmin`用户） 拥有该位置的访问权限。所以最好在运行`gpmovemirrors`工具之前在各个Segment主机 上创建好对应的路径，并使用`chown`命令付给指定用户相应的权限。

## 选项

**-B *parallel\_processes***

要并行移动的镜像Segment数量。如果不指定，工具默认启动4个并行进程，具体数量与 要移动多少镜像实例数量有关。

**-d *master\_data\_directory***

主机数据目录。如果不指定， 默认使用\$MASTER\_DATA\_DIRECTORY 的值。

**-i *move\_config\_file***

**gprecoverseg**

包含哪些镜像要被移动，被移动到哪里等信息的配置文件。  
每一个主Segment必须有一个对应的镜像Segment列出。配置文件中的每行都用以下格式（就像gp\_segment\_configuration元数据表中的信息一样）：

```
contentID:address:port:data_dir
new_address:port:data_dir
```

*contentID*是Segment实例的content ID，*address*是Segment主机的主机名或IP地址，*port*是沟通端口，*data\_dir*是Segment实例的数据目录。

**-l logfile\_directory**

日志写入路径。默认为~/gpAdminLogs。

**-v** (详细日志)

设置日志输出为详细级别。

**--version** (显示工具版本)

显示工具的版本信息。

**-?** (帮助)

显示在线帮助信息。

## 示例

将Greenplum数据库系统移动到一个不同的主机集合：

```
$ gpmovemirrors -i move_config_file
```

此处move\_config\_file的配置信息如下：

```
1:sdw2:50001:/data2/mirror/gpseg1
sdw3:50001:/data/mirror/gpseg1
2:sdw2:50001:/data2/mirror/gpseg2
sdw4:50001:/data/mirror/gpseg2
3:sdw3:50001:/data2/mirror/gpseg3
sdw1:50001:/data/mirror/gpseg3
```

Greenplum数据库® 6.0文档

## 参考指南

[Greenplum database 5管理员指南](#)[Greenplum database 4管理员指南](#)[FAQ](#)

如果您发现翻译不妥之处，欢迎点击“反馈”按钮提交详细信息

修改一个规则的定义。

## 概要

```
ALTER RULE name ON table_name RENAME TO new_name
```

## 描述

`ALTER RULE` 更改一个存在的规则的属性，当前可用的操作时修改规则的名称。

要使用`ALTER RULE`用户必须是这个规则应用的表或视图拥有者。

## 参数

*name*

要更改的存在的规则的名称。

*table\_name*

规则应用的表或者视图名称（可选的限定模式）。

*new\_name*

规则的新名称。

## 兼容性

`ALTER RULE` 是Greenplum数据库语言的扩展，是一个完整的查询重写系统。

## 另见

[CREATE RULE](#), [DROP RULE](#)





Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_partitions系统视图被用于显示分区表的结构。

**Table 1. pg\_catalog.pg\_partitions**

| 列                        | 类型       | 参考 | 描述                                                                   |
|--------------------------|----------|----|----------------------------------------------------------------------|
| schemaname               | name     |    | 分区表所属schema的名称。                                                      |
| tablename                | name     |    | 顶层父表的名称。                                                             |
| partitiontablename       | name     |    | 分区表的关系名(直接访问分区时使用的表名)。                                               |
| partitionname            | name     |    | 分区的名称(在ALTER TABLE命令引用分区时, 使用该名称)。如果在分区创建时或者由EVERY子句产生时没有给定名称则为NULL。 |
| parentpartitiontablename | name     |    | 该分区上一层父表的关系名。                                                        |
| parentpartitionname      | name     |    | 该分区上一层父表给定的名称。                                                       |
| partitiontype            | text     |    | 分区的类型(范围或者列表)。                                                       |
| partitionlevel           | smallint |    | 该分区在层次中级别。                                                           |
| partitionrank            | bigint   |    | 对于范围分区, 该分区相                                                         |

|                         |          |  |                        |
|-------------------------|----------|--|------------------------|
|                         |          |  | 对于同级其他分区的排名。           |
| partitionposition       | smallint |  | 该分区的规则顺序位置。            |
| partitionlistvalues     | text     |  | 对于列表分区，与该分区相关的列表值。     |
| partitionrangestart     | text     |  | 对于范围分区，该子分区的开始值。       |
| partitionstartinclusive | boolean  |  | 如果该子分区包含了起始值值则为T，否则为F。 |
| partitionrangeend       | text     |  | 对于范围分区，该子分区的结束值。       |
| partitionendinclusive   | boolean  |  | 如果该子分区包含了结束值则为T，否则为F。  |
| partitioneveryclause    | text     |  | 该子分区的EVERY子句（间隔）。      |
| partitionisdefault      | boolean  |  | 如果这是一个默认子分区则为T，否则为F。   |
| partitionboundary       | text     |  | 该子分区的整个分区说明。           |

**Parent topic:** 系统目录定义

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

该pg\_available\_extensions视图列出了可用于安装的扩展。

该pg\_extension系统目录表显示当前安装的扩展。

该视图是只读的。

Table 1. pg\_catalog.pg\_available\_extensions

| 列                 | 类型   | 描述                      |
|-------------------|------|-------------------------|
| name              | name | 扩展名称                    |
| default_version   | text | 默认版本名，如果没有指定的话为NULL。    |
| installed_version | text | 当前安装的扩展的版本，如果未安装则为NULL。 |
| comment           | text | 扩展控制文件的注释字符串。           |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

`pg_available_extension_versions`视图列出了可用于安装的特定扩展版本。`pg_extension`系统目录表显示当前安装的扩展。

- 参考指南

- SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

- 系统目录参考

系统表

系统视图

- 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

该视图是只读的。

**Table 1. pg\_catalog.pg\_available\_extensions**

| 列                        | 类型                   | 描述                                    |
|--------------------------|----------------------|---------------------------------------|
| <code>name</code>        | <code>name</code>    | 扩展名称。                                 |
| <code>version</code>     | <code>text</code>    | 版本名称。                                 |
| <code>installed</code>   | <code>boolean</code> | 如果此扩展程序的此版本已安装，则为True，否则为False。       |
| <code>superuser</code>   | <code>boolean</code> | 如果只允许超级用户安装此扩展，则为True，否则为False。       |
| <code>relocatable</code> | <code>boolean</code> | 如果扩展可以重新定位到另一个schema，则为True，否则为False。 |
| <code>schema</code>      | <code>name</code>    | 扩展必须装入的schema的名称，如果部分或完全可重定位，则为NULL。  |
| <code>requires</code>    | <code>name[]</code>  | 必备扩展的名称，如果没有则为NULL。                   |
| <code>comment</code>     | <code>text</code>    | 扩展控制文件的注释字符串。                         |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

gp\_segment\_configuration表包含有关Mirror和Segment实例配置的信息。

Table 1. pg\_catalog.gp\_segment\_configuration

| 列              | 类型       | 参考 | 描述                                               |
|----------------|----------|----|--------------------------------------------------|
| dbid           | smallint |    | Segment (或Master) 实例的唯一标识符。                      |
| content        | smallint |    | Segment实例的内容标识符。<br>主Segment实例及其镜像将始终具有相同的内容标识符。 |
|                |          |    | 对于Segment, 值为0到N-1, 其中N是系统中主Segment的数量。          |
|                |          |    | 对于master, 值始终为-1。                                |
| role           | char     |    | Segment当前正在运行的角色。值为p (主) 或m (镜像)。                |
| preferred_role | char     |    | 最初在初始化时分配Segment的角色。值为p (主) 或m (镜像)。             |
| mode           | char     |    | Segment实例及其镜像副本的同步状态。值为s (同步) 或n (未同步)。          |
| status         | char     |    | Segment实例的故障状态。值是u (存活) 或d (死机)。                 |
| port           | integer  |    | 数据库监听的TCP端口。T                                    |

|          |      |  |                                                           |
|----------|------|--|-----------------------------------------------------------|
| hostname | text |  | Segment主机的主机名。                                            |
| address  | text |  | 用于访问Segment主机上的特定Segment实例的主机名。此值可能与未配置每个接口主机名的系统上的主机名相同。 |
| datadir  | text |  | Segment实例数据目录。                                            |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

`pg_type`系统目录表存储有关数据类型的信息。基类（标量类型）由CREATE TYPE创建。而域由CREATE DOMAIN创建。数据库中的每一个表都会有一个自动创建的组合类型，用于表示表的行结构。也可以使用CREATE TYPE AS创建组合类型。

## □ 参考指南

### □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

### □ 系统目录参考

系统表

系统视图

### □ 系统目录定义

foreign\_data\_wrapper\_options

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_progress

gpexpand.status

**Table 1. pg\_catalog.pg\_type**

| 列              | 类型      | 参考               | 描述                                                                                                                                                                                                                                                                                                                                                   |
|----------------|---------|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| oid            | oid     |                  | 行标识符（隐藏属性：必须显示选择）                                                                                                                                                                                                                                                                                                                                    |
| typename       | name    |                  | 数据类型名称                                                                                                                                                                                                                                                                                                                                               |
| typnamespace   | oid     | pg_namespace.oid | 包含此类型的命名空间的OID                                                                                                                                                                                                                                                                                                                                       |
| typowner       | oid     | pg_authid.oid    | 类型的拥有者                                                                                                                                                                                                                                                                                                                                               |
| typlen         | int2    |                  | 对于一个固定尺寸的类型， <code>typlen</code> 是该类型内部表示的字节数。对于一个变长类型， <code>typlen</code> 为负值。-1表示一个“varlena”类型（具有长度字），-2表示一个以null结尾的C字符串。                                                                                                                                                                                                                         |
| typbyval       | boolean |                  | 决定内部例程传递这个类型的数值时是通过传值还是传引用方式。如果 <code>typlen</code> 不是1、2或4（或者在Datum为8字节的机器上为8）， <code>typlen</code> 最好是假。变长类型总是传引用。注意即使长度允许传值， <code>typbyval</code> 也可以为假。                                                                                                                                                                                         |
| typtype        | char    |                  | b表示基类，c表示组合类型，d表示域，e表示枚举类型，p表示伪类型，或r表示范围类型。另见 <code>typrelid</code> 和 <code>typbasetype</code> 。                                                                                                                                                                                                                                                     |
| typcategory    | char    |                  | 解析器使用的数据类型的任意分类，以确定应首选哪些隐式转换。请参阅 <a href="#">类别代码</a> 。                                                                                                                                                                                                                                                                                              |
| typispreferred | boolean |                  | 如果类型是其 <code>typcategory</code> 中的首选转换目标，则为True                                                                                                                                                                                                                                                                                                      |
| typisdefined   | boolean |                  | 如果定义了类型，则为True；如果是尚未定义的类型的占位符条目，则为false。如果为false，则可以依赖除类型名称，名称空间和OID之外的任何                                                                                                                                                                                                                                                                            |
| typdelim       | char    |                  | 解析数组输入时分隔此类型的两个值的字符。请注意，分隔符与数组元素数据类型相关联，而不是与数组数据类型相关联。                                                                                                                                                                                                                                                                                               |
| typrelid       | oid     | pg_class.oid     | 如果这是复合类型（请参阅 <code>typtype</code> ），则此列指向定义相应表的 <code>pg_class</code> 条目。（对于独立的复合类型， <code>pg_class</code> 条目实际上并不代表表，但无论如何都需要将类型的 <code>pg_attribute</code> 条目链接到。）非复合类型为零。                                                                                                                                                                         |
| typelem        | oid     | pg_type.oid      | 如果不为0，则它标识 <code>pg_type</code> 中的另一行。然后，当前类型可以像数组一样下标，产生类型为 <code>typelem</code> 的值。“true”数组类型是可变长度（ <code>typlen = -1</code> ），但某些固定长度（ <code>typlen &gt; 0</code> ）类型也具有非零 <code>typelem</code> ，例如 <code>name</code> 和 <code>point</code> 。如果固定长度类型有 <code>typelem</code> ，则其内部表示必须是 <code>typelem</code> 数据类型的某些值，而没有其他数据。可变长度数组类型具有由数组子例程定义的头。 |
| typarray       | oid     | pg_type.oid      | 如果不为0，则标识 <code>pg_type</code> 中的另一行，这是具有此类型作为其元素的“true”数组类型。使用 <code>pg_type.typarray</code> 查找与特定                                                                                                                                                                                                                                                  |

|              |         |                  |                                                                                                                                                                                                                                                                       |
|--------------|---------|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|              |         |                  | 类型关联的数组类型。                                                                                                                                                                                                                                                            |
| 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                                                                                                                                                                                                                                                 |
| typmodin     | regproc | pg_proc.oid      | 键入修饰符输入函数，如果类型不支持修饰符，则为0                                                                                                                                                                                                                                              |
| typmodout    | regproc | pg_proc.oid      | 键入修饰符输出函数，或0以使用标准格式                                                                                                                                                                                                                                                   |
| typanalyze   | regproc | pg_proc.oid      | 自定义ANALYZE函数，0表示使用标准函数                                                                                                                                                                                                                                                |
| typalign     | char    |                  | <p>存储此类型的值时所需的对齐方式。它适用于磁盘上的存储以及Greenplum数据库中值的大多数表示。当连续存储多个值时，例如在磁盘上的完整行的表示中，在此类型的数据之前插入填充，以便它在指定的边界上开始。对齐参考是序列中第一个数据的开头。可能的值是：</p> <p>c = char对齐，即不需要对齐。</p> <p>s = short对齐（在大部分机器上为2字节）。</p> <p>i = int对齐（在大部分机器上为4字节）。</p> <p>d = double对齐（在很多机器上为8字节，但绝不是全部）。</p> |
| typstorage   | char    |                  | <p>对于varlena类型（具有typlen = -1的那些），告知该类型是否准备好toasting以及该类型的属性的默认策略应该是什么。可能的值是：</p> <p>p: 值必须明文存储。</p> <p>e: 值可以存储在辅助关系中（如果关系有一个，请参阅pg_class.reltoastrelid）。</p> <p>m: 值可以压缩内联存储。</p> <p>x: 值可以内联压缩存储或存储在二级存储中。</p> <p>请注意，m列也可以移出到辅助存储，但仅作为最后的手段（首先移动e和x列）。</p>         |
| typnotnull   | boolean |                  | 表示类型的非空约束。仅用于域。                                                                                                                                                                                                                                                       |
| typbasetype  | oid     | pg_type.oid      | 标识域所基于的类型。如果此类型不是域，则为零。                                                                                                                                                                                                                                               |
| typtypmod    | int4    |                  | 域使用typtypmod来记录要应用于其基类型的typmod（如果基类型不使用typmod，则为-1）。-1如果此类型不是域。                                                                                                                                                                                                       |
| typndims     | int4    |                  | 域上数组的数组维数（如果typbasetype是数组类型）。除了数组类型的域以外的类型为零。                                                                                                                                                                                                                        |
| typcollation | oid     | pg_collation.oid | 指定类型的排序规则。如果类型不支持排序规则，则为零。对于支持排序规则的基本类型，该值为DEFAULT_COLLATION_OID。如果为域指定了一个域，则通过可折叠类型的域可以具有一些其他排序规则OID。                                                                                                                                                                |

|               |              |  |                                                                                                                                                                           |
|---------------|--------------|--|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| typdefaultbin | pg_node_tree |  | 如果不为null，则为该类型的默认表达式的nodeToString()表示。这仅用于域。                                                                                                                              |
| typdefault    | text         |  | 如果类型没有关联的默认值，则为null。如果typdefaultbin不为null，则typdefault必须包含由typdefaultbin表示的默认表达式的人类可读版本。如果typdefaultbin为null且typdefault不为null，则typdefault是类型默认值的外部表示，可以将其输入到类型的输入转换器以生成常量。 |
| typacl        | aclitem[]    |  | 访问权限；有关详细信息，请参阅 <a href="#">GRANT</a> 和 <a href="#">REVOKE</a> 。                                                                                                          |

下表列出了typcategory的系统定义值。此列表的任何未来添加项也将是大写ASCII字母。所有其他ASCII字符都保留给用户定义的类别。

Table 2. typcategory Codes

| 代码 | 种类      |
|----|---------|
| A  | 数组类型    |
| B  | 布尔类型    |
| C  | 复合类型    |
| D  | 日期/时间类型 |
| E  | 枚举类型    |
| G  | 几号类型    |
| I  | 网络地址类型  |
| N  | 数字类型    |
| P  | 伪类型     |
| R  | 范围类型    |
| S  | 字符串类型   |
| T  | 时间跨度类型  |
| U  | 用户自定义类型 |
| V  | 位串类型    |
| X  | 未知类型    |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

## □ 参考指南

## □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

## □ 系统目录参考

系统表

系统视图

## □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`gp_configuration_history`表包含有关故障检测和恢复操作相关的系统更改信息。`fts_probe`进程将数据记录到此表，和相关的`gpcheck`, `gprecoverseg`和`gpinitsystem`之类的管理工具一样。例如，当用户向系统添加新的Segment和镜像Segment时，这些事件会被记录到`gp_configuration_history`。

该表仅在Master上有数据。该表在`pg_global`表空间中定义，这意味着它在系统中的所有数据库之间全局共享。

**Table 1. pg\_catalog(gp\_configuration\_history)**

| 列                 | 类型                                    | 参考                                         | 描述                                   |
|-------------------|---------------------------------------|--------------------------------------------|--------------------------------------|
| <code>time</code> | <code>timestamp with time zone</code> |                                            | 记录事件的时间戳。                            |
| <code>dbid</code> | <code>smallint</code>                 | <code>gp_segment_configuration.dbid</code> | 系统分配的ID。Segment (或者Master) 实例的唯一标识符。 |
| <code>desc</code> | <code>text</code>                     |                                            | 时间的文本描述。                             |

有关`gpcheck`、`gprecoverseg`和`gpinitsystem`的信息，请参阅Greenplum数据库工具指南。

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

gp\_distribution\_policy表包含有关Greenplum数据库表及其分布表数据的策略的信息。该表仅在Master上填充。该表不是全局共享的，这意味着每个数据库都有自己的副本。

Table 1. pg\_catalog(gp\_distribution\_policy)

| 列           | 类型         | 参考                  | 描述                                                                                                                       |
|-------------|------------|---------------------|--------------------------------------------------------------------------------------------------------------------------|
| localoid    | oid        | pg_class.oid        | 表对象标识符(OID)。                                                                                                             |
| policytype  | char       |                     | 表分布策略： <ul style="list-style-type: none"><li>p - 分区策略。表数据在segment实例中分布。</li><li>r - 复制策略。表数据在每个segment实例中冗余分布。</li></ul> |
| numsegments | integer    |                     | 有表数据分布的节点数量。                                                                                                             |
| distkey     | int2vector | pg_attribute.attnum | 分布列的列编号。                                                                                                                 |
| distclass   | oidvector  | pg_opclass.oid      | 分布列的运算符类标识符。                                                                                                             |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

gp\_fastsequence表包含有关追加优化表和列存表的信息。  
last\_sequence值表示该表当前使用的最大行号。

## □ 参考指南

### □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

### □ 系统目录参考

系统表

系统视图

### □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

**Table 1. pg\_catalog(gp\_fastsequence)**

| 列             | 类型     | 参考           | 描述                                       |
|---------------|--------|--------------|------------------------------------------|
| objid         | oid    | pg_class.oid | 用于跟踪追加优化文件段的pg_aoseg.pg_aocsseg_*表的对象ID。 |
| objmod        | bigint |              | 对象修饰符。                                   |
| last_sequence | bigint |              | 对象使用的最后一个序列号。                            |

**Parent topic:** [系统目录定义](#)

[Greenplum数据库® 6.0文档](#)[参考指南](#)[Greenplum database 5管理员指南](#)[Greenplum database 4管理员指南](#)[FAQ](#)

如果您发现翻译不妥之处，欢迎点击“[反馈](#)”按钮提交详细信息

The `gp_global_sequence` table contains the log sequence number position in the transaction log, which is used by the file replication process to determine the file blocks to replicate from a primary to a mirror segment.

Table 1. `pg_catalog.gp_global_sequence`

| column                    | type                | references | description                                         |
|---------------------------|---------------------|------------|-----------------------------------------------------|
| <code>sequence_num</code> | <code>bigint</code> |            | Log sequence number position in the transaction log |



Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

gp\_id系统目录表标识了Greenplum数据库系统名称和系统的Segment数量。它还具有表所在的特定数据库实例（Segment或Master）的本地值。该表在pg\_global表空间中定义，这意味着它在系统中的所有数据库之间全局共享。

Table 1. pg\_catalog(gp\_id)

| 列           | 类型      | 参考 | 描述                                                      |
|-------------|---------|----|---------------------------------------------------------|
| gpname      | name    |    | 这个Greenplum数据库系统的名称。                                    |
| numsegments | integer |    | Greenplum数据库系统中的Segment数量。                              |
| dbid        | integer |    | 此Segment (或Master) 实例的唯一标识符。                            |
| content     | integer |    | 该Segment实例上的这部份数据的ID。<br>主Segment及其镜像Segment将具有相同的内容ID。 |
|             |         |    | 对于Segment, 值为0-N-1, 其中N是Greenplum数据库中的Segment数量。        |
|             |         |    | 对于Master,其值为-1。                                         |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

gp\_stat\_replication视图包含walsender进程的复制统计信息，该进程在启用master或segment镜像时用于Greenplum数据库预写日志记录(WAL)复制。

Table 1. gp\_catalog(gp\_stat\_replication)

| 列                | 类型        | 参考 | 描述                                                                                              |
|------------------|-----------|----|-------------------------------------------------------------------------------------------------|
| gp_segment_id    | integer   |    | segment (或master) 实例的唯一标识符。                                                                     |
| pid              | integer   |    | walsender后端进程的进程ID。                                                                             |
| usesysid         | oid       |    | 运行walsender后端进程的用户系统标识。                                                                         |
| username         | name      |    | 运行walsender后端进程的用户名。                                                                            |
| application_name | text      |    | 客户端应用名称。                                                                                        |
| client_addr      | inet      |    | 客户端IP地址。                                                                                        |
| client_hostname  | text      |    | 客户端主机名。                                                                                         |
| client_port      | integer   |    | 客户端端口号。                                                                                         |
| backend_start    | timestamp |    | 操作开始时间戳。                                                                                        |
| state            | text      |    | walsender state. The value can be:<br><br>startup<br><br>backup<br><br>catchup<br><br>streaming |
| sent_location    | text      |    | walsender xlog record sent location.                                                            |
| write_location   | text      |    | walreceiver record write location.                                                              |
| flush_location   | text      |    | walreceiver xlog record flush location.                                                         |

|                 |         |                                                               |
|-----------------|---------|---------------------------------------------------------------|
| replay_location | text    | Master standby or segment mirror xlog record replay location. |
| sync_priority   | integer | Priority. The value is 1.                                     |
| sync_state      | text    | walsendersynchronization state. The value is sync.            |
| sync_error      | text    | walsender synchronization error. none if no error.            |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

gp\_version\_at\_initdb表在Greenplum数据库系统的Master和每一个Segment上被填充。它标识着系统初始化时使用的Greenplum数据库版本。这个表被定义在pg\_global表空间中，意味着它在系统中的所有数据库中全局共享。

**Table 1. pg\_catalog\_gp\_version**

| 列              | 类型      | 参考 | 描述         |
|----------------|---------|----|------------|
| schemaversion  | integer |    | Schema版本号。 |
| productversion | text    |    | 产品版本号。     |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`gpexpand.status`表包含有关系统扩容操作状态的信息。扩容中涉及的特定表的状态存储在[gpexpand.status\\_detail](#)。

在正常的扩容操作中，不需要修改存储在该表中的数据。

**Table 1. gpexpand.status**

| 列       | 类型                          | 参考 | 描述                                                                                                  |
|---------|-----------------------------|----|-----------------------------------------------------------------------------------------------------|
| status  | text                        |    | 跟踪扩展操作的状态。有效值为：<br>SETUP<br>SETUP DONE<br>EXPANSION<br>STARTED<br>EXPANSION<br>STOPPED<br>COMPLETED |
| updated | timestamp without time zone |    | 最后状态变化的时间戳。                                                                                         |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`gpexpand.status_detail`表包含了有关系统扩容操作所涉及的表的状态的信息。 用户可以查询此表以确定正在扩容的表的状态，或查看已完成表的开始和结束时间。

此表还存储表的相关信息，如OID、磁盘尺寸。 扩容的整体状态信息存储在`gpexpand.status` 中。

In a normal expansion operation it is not necessary to modify the data stored in this table.

**Table 1. `gpexpand.status_detail`**

| 列                   | 类型      | 参考 | 描述                                          |
|---------------------|---------|----|---------------------------------------------|
| dbname              | text    |    | 表所属数据库的名称。                                  |
| fq_name             | text    |    | 表的完全限定名称。                                   |
| table_oid           | oid     |    | 表的OID.                                      |
| root_partition_name | text    |    | 对于分区表来说，根分区的名称。否则，None。                     |
| rank                | int     |    | 等级决定表被扩容的顺序。扩容工具将在rank上排序，并首先扩容排名最低的表。      |
| external_writable   | boolean |    | 标识表是否是外部可写表。(外部可写表需要不同的语法来扩容)。              |
| status              | text    |    | 此表的扩容状态。有效值为:<br>NOT STARTED<br>IN PROGRESS |

|                    |                             |  |                                                                                                 |
|--------------------|-----------------------------|--|-------------------------------------------------------------------------------------------------|
|                    |                             |  | FINISHED                                                                                        |
|                    |                             |  | NO LONGER EXISTS                                                                                |
| expansion_started  | timestamp without time zone |  | 此表扩容开始的时间戳。此字段仅在表成功扩容后填充。                                                                       |
| expansion_finished | timestamp without time zone |  | 此表扩容完成的时间戳。                                                                                     |
| source_bytes       |                             |  | 与源表相关的磁盘空间尺寸。由于堆表中的表膨胀和扩容后不同的Segment数量，最终的字节数是否与源字节相同是不可预测的。跟踪此信息有助于提供进度度量，以帮助进行端到端扩容操作的持续时间估计。 |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

## □ 参考指南

## □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

## □ 系统目录参考

系统表

系统视图

## □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_aggregate`存储关于聚集函数的信息。 聚集函数是对一个值集合（通常是来自于匹配某个查询条件的每个行的一个列值）进行操作并且返回从这些值计算出的一个值的函数。 典型的聚集函数是`sum`、`count`和`max`。`pg_aggregate`里的每个项都是一个`pg_proc`项的扩展。`pg_proc`的项记载该聚集的名字、输入输出的数据类型以及其他和普通函数类似的信息。

**Table 1. pg\_catalog.pg\_aggregate**

| 列             | 类型      | 参考                           | 描述                                                        |
|---------------|---------|------------------------------|-----------------------------------------------------------|
| aggfnoid      | regproc | <code>pg_proc.oid</code>     | 聚集函数的OID                                                  |
| aggtransfn    | regproc | <code>pg_proc.oid</code>     | 转移函数的OID                                                  |
| aggcombinefn  | regproc |                              | 预备函数的OID (如果没有就为0)                                        |
| aggfinalfn    | regproc | <code>pg_proc.oid</code>     | 最终函数的OID (如果没有就为0)                                        |
| agginitval    | text    |                              | 转移状态的初始值。这是一个文本域，它包含初始值的外部字符串表现形式。如果这个域为NULL，则转移状态从NULL开始 |
| aggordered    | Boolean |                              | 如果为true，则聚集定义为ORDERED。                                    |
| aggsortop     | oid     | <code>pg_operator.oid</code> | 相关的排序操作符的OID (如果没有则为零)                                    |
| aggtranstype  | oid     | <code>pg_type.oid</code>     | 聚集函数的内部转移(状态)数据的类型                                        |
| aggtransspace | int4    | <code>pg_type.int4</code>    | 估计的状态数据大小 (默认估计)                                          |

**Parent topic:** 系统目录定义

为0)

`pg_am`表存储有关索引访问方法的信息。系统所支持的每一种索引访问方法都有一行。

## □ 参考指南

### □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

### □ 系统目录参考

系统表

系统视图

### □ 系统目录定义

`foreign_data_wrapper_op`

`foreign_data_wrappers`

`foreign_server_options`

`foreign_servers`

`foreign_table_options`

`foreign_tables`

`gp_configuration_history`

`gp_distributed_log`

`gp_distributed_xacts`

`gp_distribution_policy`

`gpexpand.expansion_pro`

`gpexpand.status`

**Table 1. pg\_catalog.pg\_am**

| 列                           | 类型                   | 参考                       | 描述                                           |
|-----------------------------|----------------------|--------------------------|----------------------------------------------|
| <code>oid</code>            | <code>oid</code>     |                          | 行标识符 (隐藏属性;必须显式选择)                           |
| <code>amname</code>         | <code>name</code>    |                          | 访问方法的名称                                      |
| <code>amstrategies</code>   | <code>int2</code>    |                          | 此访问方法的运算符策略数, 如果访问方法没有一组固定的运算符策略, 则为零        |
| <code>amsupport</code>      | <code>int2</code>    |                          | 此访问方法的支持例程数量                                 |
| <code>amcanorder</code>     | <code>boolean</code> |                          | 访问方法是否支持按索引列的值排序的有序扫描?                       |
| <code>amcanorderbyop</code> | <code>boolean</code> |                          | 访问方法是否支持按索引列上的运算符结果排序的有序扫描?                  |
| <code>amcanbackward</code>  | <code>boolean</code> |                          | 访问方法是否支持后向扫描?                                |
| <code>amcanunique</code>    | <code>boolean</code> |                          | 访问方法是否支持唯一索引?                                |
| <code>amcanmulticol</code>  | <code>boolean</code> |                          | 访问方法是否支持多列索引?                                |
| <code>amoptionalkey</code>  | <code>boolean</code> |                          | 访问方法是否支持对第一个索引列没有任何约束的扫描?                    |
| <code>amsearcharray</code>  | <code>boolean</code> |                          | 访问方法是否支持 <code>ScalarArrayOpExpr</code> 搜索?  |
| <code>amsearchnulls</code>  | <code>boolean</code> |                          | 访问方法是否支持 <code>IS NULL / NOT NULL</code> 搜索? |
| <code>amstorage</code>      | <code>boolean</code> |                          | 索引存储数据类型可以与列数据类型不同吗?                         |
| <code>amclusterable</code>  | <code>boolean</code> |                          | 这种类型的索引可以聚集在一起吗?                             |
| <code>ampredlocks</code>    | <code>boolean</code> |                          | 这种类型的索引是否管理细粒度的谓词锁?                          |
| <code>amkeytype</code>      | <code>oid</code>     | <code>pg_type.oid</code> | 存储在索引中的数据类型,                                 |

|                 |         |             |                             |
|-----------------|---------|-------------|-----------------------------|
|                 |         |             | 如果不是固定类型，则为零                |
| amininsert      | regproc | pg_proc.oid | “插入此元组”函数                   |
| ambeginscan     | regproc | pg_proc.oid | “准备索引扫描”函数                  |
| amgettuple      | regproc | pg_proc.oid | “下一个有效元组”函数，如果没有则为零         |
| amgetbitmap     | regproc | pg_proc.oid | “获取所有元组”函数，如果没有，则返回零        |
| amrescan        | regproc | pg_proc.oid | “（重新）启动索引扫描”函数              |
| amendscan       | regproc | pg_proc.oid | “索引扫描后清理”函数                 |
| ammarkpos       | regproc | pg_proc.oid | “标记当前扫描位置”函数                |
| amrestrpos      | regproc | pg_proc.oid | “恢复标记的扫描位置”函数               |
| ambuild         | regproc | pg_proc.oid | “建立新的索引”函数                  |
| ambuildempty    | regproc | pg_proc.oid | “构建空索引”函数                   |
| ambulkdelete    | regproc | pg_proc.oid | 批量删除函数                      |
| amvacuumcleanup | regproc | pg_proc.oid | Post-VACUUM清理功能             |
| amcanreturn     | regproc | pg_proc.oid | 用于检查索引是否支持仅索引扫描的函数，如果没有则支持零 |
| amcostestimate  | regproc | pg_proc.oid | 用于估计索引扫描成本的函数               |
| amoptions       | regproc | pg_proc.oid | 为索引解析和验证reloptions的函数       |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

- 参考指南

- SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

- 系统目录参考

系统表

系统视图

- 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_amop表存储有关与索引访问方法操作符类关联的运算符的信息。每个运算符都有一行是运算符类的成员。

条目的amopmethod必须与其包含的运算符族的opfmethod匹配（包括amopmethod这里出于性能原因故意对catalog结构进行非规范化）。此外，amoplefttype和amoprighttype必须与引用的pg\_operator条目的oprleft和oprright字段匹配。

Table 1. pg\_catalog.pg\_amop

| 列              | 类型   | 参考              | 描述                                      |
|----------------|------|-----------------|-----------------------------------------|
| oid            | oid  |                 | 行标识符（隐藏属性；必须显式选择）                       |
| amopfamily     | oid  | pg_opfamily.oid | 此条目所针对的运算符系列                            |
| amoplefttype   | oid  | pg_type.oid     | 左侧输入数据类型的运算符                            |
| amoprighttype  | oid  | pg_type.oid     | 右侧输入数据类型的运算符                            |
| amopstrategy   | int2 |                 | 元算符策略编号                                 |
| amoppurpose    | char |                 | 运算符目的，s用于搜索或o用于订购                       |
| amopopr        | oid  | pg_operator.oid | 运算符的OID                                 |
| amopmethod     | oid  | pg_am.oid       | 运算符成员的索引访问方法                            |
| amopsortfamily | oid  | pg_opfamily.oid | 如果是一个排序运算符，则该条目按此排序的B树运算符系列；如果是搜索运算符则为零 |

Parent topic: 系统目录定义



Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_amproc表存储与索引访问操作符相关的支持过程的信息。对于属于一个操作符类的每个支持过程都有一行。

Table 1. pg\_catalog.pg\_amproc

| 列               | 类型      | 参考              | 描述                 |
|-----------------|---------|-----------------|--------------------|
| oid             | oid     |                 | 行标识符 (隐藏属性;必须显式选择) |
| amprocfamily    | oid     | pg_opfamily.oid | 此条目适用的运算符系列        |
| amproclefttype  | oid     | pg_type.oid     | 左侧输入数据类型的运算符       |
| amprocrighttype | oid     | pg_type.oid     | 右侧输入数据类型的运算符       |
| amprocnum       | int2    |                 | 支持过程编号             |
| amproc          | regproc | pg_proc.oid     | 过程的OID             |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_appendonly表包含有关追加优化表的储存选项和其他特性的信息。

Table 1. pg\_catalog.pg\_appendonly

| 列               | 类型       | 参考 | 描述                                                                               |
|-----------------|----------|----|----------------------------------------------------------------------------------|
| relid           | oid      |    | 压缩表的表对象标识符 (OID)。                                                                |
| blocksize       | integer  |    | 用于追加优化表压缩的块尺寸。有效值为8K - 2M。默认值为32K。                                               |
| safefswritesize | integer  |    | 在非成熟文件系统中追加优化表的安全写操作的最小尺寸。通常设置为文件系统扩展块尺寸的倍数，例如Linux ext3是4096字节，所以通常使用32768的值。   |
| compresslevel   | smallint |    | 压缩级别，压缩比从1增加到19。                                                                 |
| majorversion    | smallint |    | pg_appendonly表的主要版本号。                                                            |
| minorversion    | smallint |    | pg_appendonly表的次要版本号。                                                            |
| checksum        | boolean  |    | 存储的校验和值，在压缩和扫描时用来比较数据块的状态，以确保数据完整性。                                              |
| comprestype     | text     |    | 用于追加优化表的压缩类型。有效值为： <ul style="list-style-type: none"><li>• zlib (gzip压</li></ul> |

|              |         |  |                                   |
|--------------|---------|--|-----------------------------------|
|              |         |  | 缩)<br>● zstd<br>(Zstandard压<br>缩) |
| columnstore  | boolean |  | 1为列存， 0为行<br>存。                   |
| segrelid     | oid     |  | 表在磁盘上<br>的Segment文<br>件ID。        |
| segidxid     | oid     |  | 索引在磁盘上<br>的Segment文<br>件ID。       |
| blkdirrelid  | oid     |  | 用于磁盘上列存<br>表文件的块。                 |
| blkdiridxid  | oid     |  | 用于磁盘上列存<br>索引文件的块。                |
| visimaprelid | oid     |  | 表的可见性映<br>射。                      |
| visimapidxid | oid     |  | 可见性映射上<br>的B-树索引。                 |

**Parent topic:** 系统目录定义

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_attrdef表储存列的默认值，关于列的主要信息储存  
在[pg\\_attribute](#) 中。只有那些显式指定了默认值的列（在创建表或者  
添加列时）才会在该表中有一项。

Table 1. pg\_catalog.pg\_attrdef

| 列       | 类型   | 参考                  | 描述                                                                                 |
|---------|------|---------------------|------------------------------------------------------------------------------------|
| adrelid | oid  | pg_class.oid        | 该列所属的表                                                                             |
| adnum   | int2 | pg_attribute.attnum | 列编号                                                                                |
| adbin   | text |                     | 列默认值的内部表示                                                                          |
| adsrc   | text |                     | 人类可读的默认值表示。这个字段是历史遗留下来的，最好别用。A human-readable representation of the default value. |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_attribute`存储关于表列的信息。 数据库中每个表的每一列都正好对应`pg_attribute`表的一行（还有有索引的属性项，以及所有有`pg_class`项的对象的属性）。 术语属性等效于列。

Table 1. `pg_catalog.pg_attribute`

| 列              | 类型      | 参考                        | 描述                                                                                                                             |
|----------------|---------|---------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| attrelid       | oid     | <code>pg_class.oid</code> | 该列所属的表。                                                                                                                        |
| attname        | name    |                           | 列名。                                                                                                                            |
| atttypid       | oid     | <code>pg_type.oid</code>  | 该列的数据类型。                                                                                                                       |
| attstatttarget | int4    |                           | 控制由 <code>ANALYZE</code> 为此列积累的统计信息的详细程度。0值表示不应收集统计信息。负值表示使用系统默认的统计信息目标。正值的确切含义依赖于数据类型。对于标量数据类型，它既是要收集的“最常用值”的目标，也是要创建的柱状图的目标。 |
| attlen         | int2    |                           | 该列类型的 <code>pg_type.typlen</code> 的副本。                                                                                         |
| attnum         | int2    |                           | 列编号。普通列从1开始编号。系统列（如OID），具有（任意）负编号。                                                                                             |
| attndims       | int4    |                           | 如果列是一个数组类型则是维度数；否则为0（目前，数组的维数不是强制的，所以任何非0值都能有效地表示它为一个数组）。                                                                      |
| attcacheoff    | int4    |                           | 在存储中始终为-1，但是当加载到内存中的行描述符时，这可能会被更新以缓存该属性在行中的偏移量。                                                                                |
| atttypmod      | int4    |                           | 记录在表创建时的特定类型的数据（例如， <code>varchar</code> 列的最大长度）。它被传递到特定类型的输入函数和长度强制函数。对于不需要它的类型，该值通常为-1。                                      |
| attbyval       | boolean |                           | 该列类型的 <code>pg_type.typbyval</code> 副                                                                                          |

|               |           |                  |                                                                        |
|---------------|-----------|------------------|------------------------------------------------------------------------|
|               |           |                  | 本。                                                                     |
| attstorage    | char      |                  | 通常是该列类型的pg_type.typstorage副本。对于可TOAST的数据类型来说，可以在列创建之后更改这些数据类型，以控制存储策略。 |
| attalign      | char      |                  | 该列类型的pg_type.typalign副本。                                               |
| attnotnull    | boolean   |                  | 这表示一个非空约束。可以更改此列以启用或禁用该约束。                                             |
| atthasdef     | boolean   |                  | 此列具有默认值，这种情况下，将在pg_attrdef catalog中存在相应的条目实际定义默认值。                     |
| attisdropped  | boolean   |                  | 该列已被删除，不再有效。已删除的列仍然物理存在于表中，但是会被解析器忽略，所以无法通过SQL访问。                      |
| attislocal    | boolean   |                  | 该列在表中本地定义。请注意，列可以同时在本地定义和继承。                                           |
| attinhcount   | int4      |                  | 这列的直接祖先的数量。具有非0数量祖先的列不能被删除或重命名。                                        |
| attcollation  | oid       | pg_collation.oid | 列的已定义排序规则，如果不是可合并数据类型，则为零。                                             |
| attacl        | aclitem[] |                  | 列级访问权限（如果已在此列上专门授予）。                                                   |
| attoptions    | text[]    |                  | 属性级选项，作为“keyword = value”字符串。                                          |
| attfdwoptions | text[]    |                  | 属性级外部数据包装器选项，作为“keyword = value”字符串。                                   |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_auth_members`系统目录表显示角色之间的成员关系。允许任何非循环关系。因为角色是系统范围的，所以`pg_auth_members`表在Greenplum数据库系统的所有数据库之间共享。

**Table 1. pg\_catalog.pg\_auth\_members**

| 列            | 类型      | 参考                         | 描述                        |
|--------------|---------|----------------------------|---------------------------|
| roleid       | oid     | <code>pg_authid.oid</code> | 父级（组）角色的ID                |
| member       | oid     | <code>pg_authid.oid</code> | 成员角色的ID                   |
| grantor      | oid     | <code>pg_authid.oid</code> | 授予此成员关系的角色的ID             |
| admin_option | boolean |                            | 如果角色成员可以向其他人授予成员关系，则为true |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_options

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_progress

gpexpand.status

`pg_authid`表包含了有关数据库认证标识符（角色）的信息。角色包含用户和组的概念。用户是设置了`rolcanlogin`标志的角色。任何角色（有或者没有`rolcanlogin`）可能有其他角色作为其成员。请参阅[pg\\_auth\\_members](#)。

由于此目录包含密码，因此不是公众可读的。`pg_roles`是`pg_authid`上的一个公开可读的视图，其中模糊化了密码字段。

由于用户身份是全系统范围的，因此`pg_authid`在Greenplum数据库系统中的所有数据库之间共享：每个系统只有一个`pg_authid`副本，而不是每个数据库一个。

Table 1. pg\_catalog.pg\_authid

| 列                             | 类型                       | 参考 | 描述                                                                                                                                                                                                                         |
|-------------------------------|--------------------------|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>oid</code>              | <code>oid</code>         |    | 行标识符（隐藏属性；必须显式选择）                                                                                                                                                                                                          |
| <code>rolname</code>          | <code>name</code>        |    | 角色名称                                                                                                                                                                                                                       |
| <code>rolsuper</code>         | <code>boolean</code>     |    | 角色具有超级用户特权                                                                                                                                                                                                                 |
| <code>rolinherit</code>       | <code>boolean</code>     |    | 角色自动继承其所属角色的权限                                                                                                                                                                                                             |
| <code>rolcreaterole</code>    | <code>boolean</code>     |    | 角色可以创建其他更多角色                                                                                                                                                                                                               |
| <code>rolcreatedb</code>      | <code>boolean</code>     |    | 角色可以创建数据库                                                                                                                                                                                                                  |
| <code>rolcatupdate</code>     | <code>boolean</code>     |    | 角色可以直接更新系统目录（即使超级用户也不能这样做，除非此列为真）                                                                                                                                                                                          |
| <code>rolcanlogin</code>      | <code>boolean</code>     |    | 角色可以登录。也就是说，该角色可以作为初始会话授权标识符                                                                                                                                                                                               |
| <code>rolreplication</code>   | <code>boolean</code>     |    | 角色是复制角色。也就是说，此角色可以使用户 <code>pg_start_backup</code> 和 <code>pg_stop_backup</code> 启动流复制并设置/取消设置系统备份模式。                                                                                                                      |
| <code>rolconnlimit</code>     | <code>int4</code>        |    | 对于那些可以登录的角色，这一列设置此角色可以创建的最大并发连接数。-1表示没有限制。                                                                                                                                                                                 |
| <code>rolpassword</code>      | <code>text</code>        |    | 密码（可能是加密的）；如果没有则为NULL。如果密码已加密，则此列将以字符串 <code>md5</code> 开头，后跟32个字符的十六进制MD5哈希。MD5哈希将是用户的连接到他们的用户名。例如，如果用户 <code>joe</code> 的密码为 <code>xyzzy</code> ，则Greenplum数据库将存储 <code>xyzzyjoe</code> 的MD5哈希值。Greenplum假定不遵循该格式的密码未加密。 |
| <code>rolvaliduntil</code>    | <code>timestamptz</code> |    | 密码到期时间（仅用于密码验证）；不过期则为NULL                                                                                                                                                                                                  |
| <code>rolresqueue</code>      | <code>oid</code>         |    | <code>pg_resqueue</code> 中关联资源队列ID的对象ID                                                                                                                                                                                    |
| <code>rolcreaterextgpf</code> | <code>boolean</code>     |    | 使用 <code>gpfdist</code> 或 <code>gpfdists</code> 协议创建读取外部表的权限                                                                                                                                                               |
| <code>rolcreaterexhttp</code> | <code>boolean</code>     |    | 使用 <code>http</code> 协议创建读取外部表的权限                                                                                                                                                                                          |
| <code>rolcreatewextgpf</code> | <code>boolean</code>     |    | 使用 <code>gpfdist</code> 或 <code>gpfdists</code> 协议创建写入外部表的权限                                                                                                                                                               |
| <code>rolresgroup</code>      | <code>oid</code>         |    | <code>pg_resgroup</code> 中关联资源组ID的对象ID                                                                                                                                                                                     |

Parent topic: [系统目录定义](#)



## Greenplum数据库® 6.0文档

## □ 参考指南

## □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

## □ 系统目录参考

系统表

系统视图

## □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_cast`表存储数据类型转换路径，包括内置路径和用CREATE CAST定义的路径。

请注意，`pg_cast`不代表系统已知的每种类型转换，只代表那些无法从某些通用规则推导出的转换。例如，在`pg_cast`中未显式表示域与其基类型之间的转换。另一个重要的例外是“自动I/O转换强制转换”，使用数据类型自己的I/O函数执行转换为文本或其他字符串类型的那些，在`pg_cast`中没有显式表示。

`pg_cast`中列出的强制转换函数必须始终将强制转换源类型作为其第一个参数类型，并返回强制转换目标类型作为其结果类型。强制转换函数最多可以包含三个参数。第二个参数（如果存在）必须是整数类型；它接收与目标类型关联的类型修饰符，如果没有，则返回-1。第三个参数（如果存在）必须是boolean类型；如果强制转换是显式转换，则接收为true，否则为false。

如果关联函数需要多个参数，则创建一个源和目标类型相同的`pg_cast`条目是合法的。这些条目表示“长度强制函数”，其强制该类型的值对于特定类型修饰符值是合法的。

当`pg_cast`条目具有不同的源和目标类型以及具有多个参数的函数时，该条目将从一种类型转换为另一种类型，并在单个步骤中应用长度强制。当没有这样的条目可用时，对使用类型修饰符的类型的强制涉及两个步骤，一个用于在数据类型之间进行转换，另一个用于应用修饰符。

Table 1. `pg_catalog.pg_cast`

| 列           | 类型   | 参考                       | 描述                                                                          |
|-------------|------|--------------------------|-----------------------------------------------------------------------------|
| castsource  | oid  | <code>pg_type.oid</code> | 源数据类型的OID。                                                                  |
| casttarget  | oid  | <code>pg_type.oid</code> | 目标数据类型的OID。                                                                 |
| castfunc    | oid  | <code>pg_proc.oid</code> | 用于执行此强制转换的函数的OID。<br>如果强制转换方法不需要函数，则存储零。 <span style="color: red;">□</span> |
| castcontext | char |                          | 指示可以调用强制转换的上下文。<br>e仅表示显式强制转                                                |

|            |      |  |                                                                                                        |
|------------|------|--|--------------------------------------------------------------------------------------------------------|
|            |      |  | 换 (使用CAST或::语法)。 a表示隐式分配给目标列, 和显式一样。 i表示隐式在表达式中,以及其他情况。                                                |
| castmethod | char |  | <p>指出转换的执行方式:</p> <p>f - 使用castfunc字段中标识的函数。</p> <p>i - 使用输入/输出函数。</p> <p>b - 这些类型是二进制可强制的, 不需要转换。</p> |

**Parent topic:** 系统目录定义

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_options

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_progress

gpexpand.status

系统目录表pg\_class记录表以及其他大部分具有列或者与表（也称为关系）相似的东西。这包括索引（另见[pg\\_index](#)）、序列、视图、组合类型和TOAST表。并不是所有的列对所有的关系类型都有意义。

Table 1. pg\_catalog.pg\_class

| 列               | 类型      | 参考                | 描述                                                                                                         |
|-----------------|---------|-------------------|------------------------------------------------------------------------------------------------------------|
| relname         | name    |                   | 表、索引、视图等的名字。                                                                                               |
| relnamespace    | oid     | pg_namespace.oid  | 包含这个关系的命名空间 (schema) 的OID                                                                                  |
| reltype         | oid     | pg_type.oid       | 如果有的话（索引为0，没有pg_type项），对应与此表的行类型的数据类型的OID                                                                  |
| reloftype       | oid     | pg_type.oid       | pg_type中条目的OID，用于隐含的复合类型。                                                                                  |
| relowner        | oid     | pg_authid.oid     | 关系的所有者                                                                                                     |
| relam           | oid     | pg_am.oid         | 如果这是一个索引，则表示访问方法（B树、位图、哈希等。）                                                                               |
| relfilenode     | oid     |                   | 此关系的磁盘文件的名称，如果没有则为0。                                                                                       |
| reltablespace   | oid     | pg_tablespace.oid | 存储此关系的表空间。如果为0，则表示数据库的默认表空间（如果关系没有磁盘文件，则无意义）。The tablespace in which this relation is stored. If zero, the |
| relpages        | int4    |                   | 该表的磁盘尺寸，以页面为单位（每页32k）。这只是优化器使用的估计值。它由VACUUM、ANALYZE和一些DDL命令更新。                                             |
| reltuples       | float4  |                   | 表中的行数。这只是优化器使用的估计值。它由VACUUM、ANALYZE和一些DDL命令更新。                                                             |
| relallvisible   | int32   |                   | 全部可见块的数量（此值可能不是最新的）。                                                                                       |
| reltoastrelid   | oid     | pg_class.oid      | 与这张表关联的TOAST表的OID，没有的就为0。TOAST表在辅助表中存储“行溢出”的大型属性。                                                          |
| relhasindex     | boolean |                   | 如果这是一个表并且它（或最近有）任何索引，则为True。这由CREATE INDEX设置，但不会立即由DROP INDEX清除。如果发现表没有索引，VACUUM将清除。                       |
| relisshared     | boolean |                   | 如果此表在系统中的所有数据库之间共享，则为True。仅共享某些系统目录表。                                                                      |
| relopersistence | char    |                   | 对象持久性的类型：p =堆或追加优化表，u =未记录的临时表，t =临时表。                                                                     |
| relkind         | char    |                   | 对象的类型<br>r =堆或追加优化表，i =索引，s =                                                                              |

|                |           |  |                                                                                                                                                       |
|----------------|-----------|--|-------------------------------------------------------------------------------------------------------------------------------------------------------|
|                |           |  | 序列, t = TOAST值, v = 视图, c = 组合类型, f = 外部表, u = 未登记的临时堆表, o = 内部追加优化的segment文件和EOF, b = 追加的块目录, m = 追加的可视化映射                                           |
| relstorage     | char      |  | 表的存储模式<br>a=追加优化, c=面向列的, h=堆表, v=虚拟, x=外部表。                                                                                                          |
| relnatts       | int2      |  | 关系中用户列的数量（系统列不计入）。pg_attribute中必须有这么多个相应的项。                                                                                                           |
| relchecks      | int2      |  | 表中检查约束的个数。                                                                                                                                            |
| relhasoids     | boolean   |  | 如果为关系的每一行生成OID，则为True。                                                                                                                                |
| relhaspkey     | boolean   |  | 如果表具有（或曾经拥有）主键，则为True。                                                                                                                                |
| relhasrules    | boolean   |  | 如果表有规则，则为True。                                                                                                                                        |
| relhastriggers | boolean   |  | 如果表有（或曾经有）触发器，则为True。                                                                                                                                 |
| relhassubclass | boolean   |  | 如果表具有（或曾经有）任何继承子项，则为True。                                                                                                                             |
| relispopulated | boolean   |  | 如果关系被填充，则为真（对于除某些物化视图之外的所有关系都是如此）。                                                                                                                    |
| relreplident   | char      |  | 用于为行形成“副本标识”的列：d=默认（主键，如果有），n=无，f=所有列i=具有indisreplident设置的索引，或默认值                                                                                     |
| relfrozenxid   | xid       |  | 此表中该值之前的所有事务ID都已替换为此表中的永久（冻结）事务ID。这用于跟踪表是否需要被清理以防止事务ID环绕或允许pg_clog收缩。<br><br>如果关系不是表，或者表不需要清理以防止事务ID环绕，则值为0 (InvalidTransactionId)。该表仍可能需要清理才能回收磁盘空间。 |
| relminmxid     | xid       |  | 此表中该值之前的所有multixact ID都已被此表中的事务ID替换。这用于跟踪表是否需要被清理以防止多重ID环绕或允许pg_multixact收缩。如果关系不是表，则为零 (InvalidMultiXactId)。                                         |
| relacl         | aclitem[] |  | GRANT和REVOKE分配的访问权限。                                                                                                                                  |
| reloptions     | text[]    |  | 特定于访问方法的选项，作为“keyword = value”字符串。                                                                                                                    |

**Parent topic:** 系统目录定义



Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_options

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_progress

gpexpand.status

pg\_constraint系统目录表储存表上的检查、主键、唯一和外键约束。列约束不被特别对待。每个列约束都等效于某种表约束。非空约束在[pg\\_attribute](#)目录表中表示。域上的检查约束也储存在这里。

Table 1. pg\_catalog.pg\_constraint

| 列             | 类型      | 参考                  | 描述                                                                                                                          |
|---------------|---------|---------------------|-----------------------------------------------------------------------------------------------------------------------------|
| connname      | name    |                     | 约束名称（不一定唯一！）                                                                                                                |
| connnamespace | oid     | pg_namespace.oid    | 包含约束的命名空间（schema）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 |                     | 约束是否可以延迟？                                                                                                                   |
| condeferred   | boolean |                     | 约束是否默认延迟？                                                                                                                   |
| conrelid      | oid     | pg_class.oid        | 约束所在的表；如果不是一个表约束则为0。                                                                                                        |
| contypid      | oid     | pg_type.oid         | 该约束所在的域；如果不是一个域约束则为0。                                                                                                       |
| confrelid     | oid     | pg_class.oid        | 如果是一个外键，则表示所引用的表；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 | 如果是一个表约束，则表示所约束的列的列表。                                                                                                       |
| confkey       | int2[]  | pg_attribute.attnum | 如果是一个外键，则表示所引用列的列表。                                                                                                         |
| conbin        | text    |                     | 如果是一个检查约束，则表示表达式的内部表示。                                                                                                      |
| consrc        | text    |                     | 如果是一个检查约束，则表示表达式的人类可读的表示。当被引用对象改变时候，这个属性不会被更新；例如，不会跟踪列的重命名。最好使用 <a href="#">pg_get_constraintdef()</a> 来提取检查约束的定义，而不是依赖此字段。 |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_conversion系统目录表描述了由CREATE CONVERSION定义的可用编码转换过程。

Table 1. pg\_catalog.pg\_conversion

| 列              | 类型      | 参考               | 描述                          |
|----------------|---------|------------------|-----------------------------|
| connname       | name    |                  | 转换的名称<br>(在命名空间中唯一)。        |
| connnamespace  | oid     | pg_namespace.oid | 包含此转换的命名空间<br>(schema)的OID。 |
| conowner       | oid     | pg_authid.oid    | 转换的拥有者。                     |
| conforencoding | int4    |                  | 源编码ID。                      |
| contoencoding  | int4    |                  | 目标编码ID。                     |
| conproc        | regproc | pg_proc.oid      | 转换过程。                       |
| condefault     | boolean |                  | 如果是默认转换，则为真。                |

Parent topic: 系统目录定义



Greenplum数据库® 6.0文档

## □ 参考指南

## □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

## □ 系统目录参考

系统表

系统视图

## □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_database系统目录表储存了可用数据库的信息。数据库由SQL命令CREATE DATABASE创建。和大多数系统目录不同，pg\_database在系统中所有数据库之间共享。每个系统只有一个pg\_database副本，而不是每个数据库一个。

Table 1. pg\_catalog.pg\_database

| 列             | 类型      | 参考            | 描述                                                                                                                                                                                                                                                                                |
|---------------|---------|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| datname       | name    |               | 数据库名称。                                                                                                                                                                                                                                                                            |
| datdba        | oid     | pg_authid.oid | 数据库的拥有者，通常是创建它的人。                                                                                                                                                                                                                                                                 |
| encoding      | int4    |               | 数据库的字符编码。<br>pg_encoding_to_char()可以将此编号转换为编码名称。                                                                                                                                                                                                                                  |
| datcollate    | name    |               | 此数据库的LC_COLLATE。                                                                                                                                                                                                                                                                  |
| datctype      | name    |               | 此数据库的LC_CTYPE。                                                                                                                                                                                                                                                                    |
| datistemplate | boolean |               | 如果为真则这个数据库可以被用在CREATE DATABASE的TEMPLATE子句中来创建一个新的数据库作为这个数据库的克隆体。                                                                                                                                                                                                                  |
| datallowconn  | boolean |               | 如果为假，则该数据库不可连接。这用于保护数据库template0不被修改。                                                                                                                                                                                                                                             |
| datconnlimit  | int4    |               | 设置该数据库最大并发连接数。-1表示没有限制。                                                                                                                                                                                                                                                           |
| datlastsysoid | oid     |               | 数据库中的最后一个系统OID。                                                                                                                                                                                                                                                                   |
| datfrozenxid  | xid     |               | 这个数据库中在此值之前的所有事务ID已被替换为的永久(冻结)事务ID。这用于追踪该数据库是否需要清理，以防止事务ID回卷或允许pg_clog收缩，它是每个表的pg_class.relfrozenxid的最小值。                                                                                                                                                                         |
| datminmxid    | xid     |               | A Multixact ID is used to support row locking by multiple transactions. All multixact IDs before this one have been replaced with a transaction ID in this database. This is used to track whether the database needs to be vacuumed in order to prevent multixact ID wraparound. |

|                            |                        |                                |                                                                                                                                 |
|----------------------------|------------------------|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
|                            |                        |                                | or to allow <code>pg_multixact</code> to be shrunk. It is the minimum of the per-table <code>pg_class.rexminmxid</code> values. |
| <code>dattablespace</code> | <code>oid</code>       | <code>pg_tablespace.oid</code> | 数据库的默认表空间。在此数据库中，所有 <code>pg_class.reltablespace</code> 为0的表都将存储在此表空间中。所有非共享的系统目录也将放在那里。                                        |
| <code>datacl</code>        | <code>aclitem[]</code> |                                | 由GRANT和REVOKE所给予的数据访问特权。                                                                                                        |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_db_role_setting`系统目录表记录每个角色和数据库组合的服务器配置设置的默认值。

每个Greenplum数据库集群都有一个`pg_db_role_settings`副本。此系统目录表在所有数据库中共享。

您可以使用`psql`的`\drds`元命令查看Greenplum数据库集群的服务器配置设置。

**Table 1. pg\_catalog.pg\_database**

| 列                        | 类型                  | 参考                           | 描述                                              |
|--------------------------|---------------------|------------------------------|-------------------------------------------------|
| <code>setdatabase</code> | <code>oid</code>    | <code>pg_database.oid</code> | 设置适用的数据<br>库，如果设置不<br>是特定于数据<br>库，则为零。          |
| <code>setrole</code>     | <code>oid</code>    | <code>pg_authid.oid</code>   | 设置适用的角<br>色，如果设置不<br>是特定于角色，<br>则为零。            |
| <code>setconfig</code>   | <code>text[]</code> |                              | 用户可设置的服<br>务器配置参数的<br>每个数据库和每<br>个角色特定的默<br>认值。 |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

## □ 参考指南

## □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

## □ 系统目录参考

系统表

系统视图

## □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_depend`系统目录表记录数据库对象之间依赖关系。此信息允许`DROP`命令查找哪些其他对象必须由`DROP CASCADE`删除或者必须防止在`DROP RESTRICT`情况中被删除。另请参见[pg\\_shdepend](#)，它对涉及Greenplum系统共享对象的依赖执行类似的功能。

在所有的情况下，`pg_depend`项表示在不删除依赖对象的前提下也不能删除被引用对象。不过，有几种由`deptype`区分的子类型：

- **DEPENDENCY\_NORMAL (n)** — 单独创建的对象之间的正常关系。可以删除依赖对象而不影响被引用的对象。被引用对象只能通过指定`CASCADE`来删除，在这种情况下依赖对象也会被删除。例如：表列对其数据类型具有正常的依赖性。
- **DEPENDENCY\_AUTO (a)** — 依赖对象可以独立于被引用对象而删除，并且如果被引用对象被删除，则依赖对象应该自动被删除（不管是`RESTRICT`还是`CASCADE`模式）。例如：表上的命名约束自动依赖表，因此如果表被删除，该约束也将消失。
- **DEPENDENCY\_INTERNAL (i)** — 依赖对象作为被引用对象的一部分创建，实际只是其内部实现的一部分。依赖对象的`DROP`操作将被完全禁止（我们会告诉用户针对被引用对象发出`DROP`命令来代替）。被引用对象的`DROP`命令会被传播以删除依赖对象，不管是否指定`CASCADE`（级联删除）。
- **DEPENDENCY\_PIN (p)** — 没有依赖对象；这种类型的条目是系统本身依赖于引用对象的信号，因此绝不能删除该对象。此类型的条目仅由系统初始化创建。从属对象的列包含零。

Table 1. `pg_catalog.pg_depend`

| 列          | 类型   | 参考                        | 描述                        |
|------------|------|---------------------------|---------------------------|
| classid    | oid  | <code>pg_class.oid</code> | 依赖对象所在的系统目录的OID。          |
| objid      | oid  | any OID column            | 特定依赖对象的OID。               |
| objsubid   | int4 |                           | 对于表列，这是列编号。对于其他对象类型，此列为0。 |
| refclassid | oid  | <code>pg_class.oid</code> | 被引用对象所在的系统目录的OID。         |
| refobjid   | oid  | any OID                   | 特定被引用对象                   |

|             |      | column | 的OID。                         |
|-------------|------|--------|-------------------------------|
| refobjsubid | int4 |        | 对于表列，这是被引用的列编号。对于其他对象类型，该列为0。 |
| deptype     | char |        | 定义此依赖关系的特定语义的代码。              |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_description`系统目录表存储每个数据库对象的可选描述（注释）。可以使用`COMMENT`命令来操作描述，并使用`psql`的`\d`元命令来查看。`pg_description`的初始内容中提供了许多内建系统对象的描述。另请参见[pg\\_shdescription](#)，它对涉及Greenplum数据库系统共享对象的描述执行类似的功能。

Table 1. pg\_catalog.pg\_description

| 列                        | 类型                | 参考                          | 描述                           |
|--------------------------|-------------------|-----------------------------|------------------------------|
| <code>objoid</code>      | <code>oid</code>  | <code>any OID column</code> | 该描述所涉及对象的OID。                |
| <code>classoid</code>    | <code>oid</code>  | <code>pg_class.oid</code>   | 该对象所出现的系统目录的OID。             |
| <code>objsubid</code>    | <code>int4</code> |                             | 对于表列的注释，这是列编号。对于其他对象类型，此列为0。 |
| <code>description</code> | <code>text</code> |                             | 作为此对象描述的任意文本。                |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_exttable`系统目录表用来追踪由CREATE EXTERNAL TABLE命令创建的外部表和Web表。

Table 1. pg\_catalog.pg\_exttable

| 列               | 类型      | 参考                        | 描述                              |
|-----------------|---------|---------------------------|---------------------------------|
| reloid          | oid     | <code>pg_class.oid</code> | 该外部表的OID。                       |
| urilocation     | text[]  |                           | 外部表文件的URI地址。                    |
| execlocation    | text[]  |                           | 为外部表定义的ON Segment位置。            |
| fmttype         | char    |                           | 外部表文件的格式: t是文本, c是csv。          |
| fmtopts         | text    |                           | 外部表文件的格式化选项, 如字段定界符、空字符串、转义字符等。 |
| options         | text[]  |                           | 为外部表定义的选项。                      |
| command         | text    |                           | 访问外部表所执行的OS命令。                  |
| rejectlimit     | integer |                           | 每个Segment拒绝错误行的限制, 之后装载将失败。     |
| rejectlimittype | char    |                           | 拒绝限制的阀值类型: r表示行数。               |
| logerrors       | bool    |                           | 1记录错误, 0不记录。                    |
| encoding        | text    |                           | 客户端编码。                          |
| writable        | boolean |                           | 0表示可读外部表, 1表示可写。                |

|                             |  |  |      |
|-----------------------------|--|--|------|
|                             |  |  | 外部表。 |
| <b>Parent topic:</b> 系统目录定义 |  |  |      |

Greenplum数据库® 6.0文档

系统目录表pg\_foreign\_data\_wrapper存储外部数据包装器定义。外部数据包装器是一种访问驻留在外部服务器上的外部数据的机制。

## □ 参考指南

### □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

## □ 系统目录参考

系统表

系统视图

### □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

**Table 1. pg\_catalog.pg\_foreign\_data\_wrapper**

| 列            | 类型        | 参考            | 描述                                                                            |
|--------------|-----------|---------------|-------------------------------------------------------------------------------|
| fdwname      | name      |               | 外部数据包装器的名称。                                                                   |
| fdwowner     | oid       | pg_authid.oid | 外部数据包装器的所有者。                                                                  |
| fdwhandler   | oid       | pg_proc.oid   | 对处理程序函数的引用，该函数负责为外部数据包装器提供执行例程。如果没有提供处理程序则为零。                                 |
| fdwvalidator | oid       | pg_proc.oid   | 对验证器函数的引用，该函数负责检查提供给外部数据包装器的选项的有效性。此函数还使用外部数据包装器检查外部服务器和用户映射的选项。如果未提供验证器，则为零。 |
| fdwacl       | aclitem[] |               | 访问权限；有关详细信息，请参阅 <a href="#">GRANT</a> 和 <a href="#">REVOKE</a> 。              |
| fdwoptions   | text[]    |               | 外部数据包装器特定选项，作为“keyword = value”字符串。                                           |

**Parent topic:** [系统目录定义](#)

系统目录表pg\_foreign\_server存储外部服务器定义。外部服务器描述外部数据源，例如远程服务器。您可以通过外部数据包装器访问外部服务器。

## □ 参考指南

### □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

### □ 系统目录参考

系统表

系统视图

### □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

**Table 1. pg\_catalog.pg\_foreign\_server**

| 列          | 类型        | 参考                          | 描述                                                               |
|------------|-----------|-----------------------------|------------------------------------------------------------------|
| srvname    | name      |                             | 外部服务器的名称。                                                        |
| srvowner   | oid       | pg_authid.oid               | 外部服务器的所有者。                                                       |
| srvid      | oid       | pg_foreign_data_wrapper.oid | 此外部服务器的外部数据包装器的OID。                                              |
| srvtype    | text      |                             | 服务器类型（可选）。                                                       |
| srvversion | text      |                             | 服务器版本（可选）。                                                       |
| srvacl     | aclitem[] |                             | 访问权限；有关详细信息，请参阅 <a href="#">GRANT</a> 和 <a href="#">REVOKE</a> 。 |
| srvoptions | text[]    |                             | 外部服务器特定选项，作为“keyword = value”字符串。                                |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

系统目录表pg\_foreign\_table包含有关外部表的辅助信息。外表主要由pg\_class条目表示，就像常规表一样。它的pg\_foreign\_table条目包含仅与外表相关的信息，而不包含任何其他类型的关系。

**Table 1. pg\_catalog.pg\_foreign\_table**

| 列         | 类型     | 参考                    | 描述                            |
|-----------|--------|-----------------------|-------------------------------|
| ftrelid   | oid    | pg_class.oid          | 此外表的pg_class条目的OID。           |
| ftserver  | oid    | pg_foreign_server.oid | 此外表的外部服务器的OID。                |
| ftoptions | text[] |                       | 外表选项，如“keyword = value”字符串。 . |

**Parent topic:** [系统目录定义](#)

pg\_index系统目录表包含部分关于索引的信息。剩下的信息主要在[pg\\_class](#) 表中。

Greenplum数据库® 6.0文档

## □ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

## □ 系统目录参考

系统表

系统视图

## □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

**Table 1. pg\_catalog.pg\_index**

| 列              | 类型      | 参考           | 描述                                                                                  |
|----------------|---------|--------------|-------------------------------------------------------------------------------------|
| indexrelid     | oid     | pg_class.oid | 该索引的pg_class项的OID。                                                                  |
| indrelid       | oid     | pg_class.oid | 该索引所对应的表的pg_class项的OID。                                                             |
| indnatts       | int2    |              | 索引中的列数 (与其pg_class.relnatts重複)。                                                     |
| indisunique    | boolean |              | 如果为真，则这是一个唯一索引。                                                                     |
| indisprimary   | boolean |              | 如果为真，则这个索引表示表的主键 (此属性为真时indisunique总是为真)。                                           |
| indisexclusion | boolean |              | 如果为true，则此索引支持排除约束                                                                  |
| indimmediate   | boolean |              | 如果为true，则在插入时立即强制执行唯一性检查 (如果indisunique不成立则无关紧要)                                    |
| indisclustered | boolean |              | 如果为真，表最后一次是通过CLUSTER命令在此索引上聚簇。                                                      |
| indisvalid     | boolean |              | 如果为真，则该索引目前对查询有效。如果为假，意味着该索引目前可能不完整：它仍然必须被INSERT/UPDATE操作所修改，但是它不能被安全地用于查询。         |
| indcheckxmin   | boolean |              | 如果为true，必须不使用索引，直到此pg_index行的xmin低于其TransactionXmin事件范围，因为该表可能包含具有可以看到的不兼容行的断开的HOT链 |
| indisready     | boolean |              | 如果为true，则索引当前已准备好进行插入。                                                              |

|                |            |                     |                                                                                                |
|----------------|------------|---------------------|------------------------------------------------------------------------------------------------|
|                |            |                     | False意味着INSERT/UPDATE操作必须忽略索引                                                                  |
| indislive      | boolean    |                     | 如果为false，则索引正在被删除，并且应该为所有目的而忽略                                                                 |
| indisreplident | boolean    |                     | 如果为true，则使用ALTER TABLE ... REPLICA IDENTITY USING INDEX ...将此索引选为“副本标识”                        |
| indkey         | int2vector | pg_attribute.attnum | 这是一个indnatts值数组，指示此索引索引的表列。例如，值为1 3意味着第一个和第三个表列构成索引键。此数组中的零表示相应的索引属性是表列上的表达式，而不是简单的列引用。        |
| indcollation   | oidvector  |                     | 对于索引键中的每一列，它包含用于索引的排序规则的OID。                                                                   |
| indclass       | oidvector  | pg_opclass.oid      | 对于索引键中的每一列，它包含要使用的运算符类的OID。                                                                    |
| indooption     | int2vector |                     | 这是一个indnatts值数组，用于存储每列标志位。位的含义由索引的访问方法定义。                                                      |
| indexprs       | text       |                     | 表达式树<br>(在nodeToString()表示中)，用于不是简单列引用的索引属性。这是一个列表，其中包含indkey中每个零条目的一个元素。如果所有索引属性都是简单引用，则为NULL |
| indpred        | text       |                     | 部分索引谓词的表达式树<br>(在nodeToString()表示中)。如果不是部分索引，则为NULL。                                           |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_inherits`系统目录表记录有关表继承层次结构的信息。数据库中的每个直接子表都有一个条目。（间接继承可以通过以下条目链来确定。）在Greenplum数据库中，继承关系由`CREATE TABLE`的`INHERITS`子句（独立继承）和`PARTITION BY`子句（分区子表继承）创建。

**Table 1. pg\_catalog.pg\_inherits**

| 列         | 类型   | 参考                        | 描述                                             |
|-----------|------|---------------------------|------------------------------------------------|
| inhrelid  | oid  | <code>pg_class.oid</code> | 子表的OID。                                        |
| inhpARENT | oid  | <code>pg_class.oid</code> | 父表的OID。                                        |
| inhseqno  | int4 |                           | 如果对于一个子表有多个直接父表（多继承），则这个数字将指出被继承列的排列顺序。计数从1开始。 |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_language系统目录表注册用户可以用来编写函数和存储程序的语言。它由CREATE LANGUAGE填充。

Table 1. pg\_catalog.pg\_language

| 列             | 类型      | 参考            | 描述                                                                               |
|---------------|---------|---------------|----------------------------------------------------------------------------------|
| lanname       | name    |               | 语言的名称。                                                                           |
| lanowner      | oid     | pg_authid.oid | 语言的拥有者。                                                                          |
| lanispl       | boolean |               | 对内部语言(如SQL)而言,值为假。而对于用户自定义的语言为真。目前,pg_dump仍然使用它来确定哪些语言需要被转存,但是,在将来它可能会被不同的机制所代替。 |
| lanpltrusted  | boolean |               | 如果这是一种可信的语言,则为真,表示它不会为正常SQL执行环境之外的任何东西授予访问。只有超级用户才能用不可信语言创建函数。                   |
| lanplcallfoid | oid     | pg_proc.oid   | 对于非内部的语言,该属性引用了一个语言处理程序,该程序是一个特殊的函数,负责执行所有以特定语言编写的函数。                            |
| laninline     | oid     | pg_proc.oid   | 这个属性引用一个函数名或执行内联匿名块的函数(参见DO命令)。如果不支                                              |

|              |           |             |                                                      |
|--------------|-----------|-------------|------------------------------------------------------|
|              |           |             | 持匿名块，则为0。                                            |
| lanvalidator | oid       | pg_proc.oid | 这个属性引用一个语言验证器函数，负责在创建新函数时检查新函数的语法和合法性。如果没有提供验证器，则为0。 |
| lanacl       | aclitem[] |             | 语言的访问特权。                                             |

**Parent topic:** [系统目录定义](#)

## Greenplum数据库® 6.0文档

## □ 参考指南

## □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

## □ 系统目录参考

系统表

系统视图

## □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

Note: Greenplum数据库不支持PostgreSQL[大对象工具](#)，用于流式传输存储在大型对象结构中的用户数据。

`pg_largeobject`系统目录表保存构成“大对象”的数据。大对象由创建时分配的OID来识别。每个大对象被分解成便于在`pg_largeobject`表中储存为行的足够小的段或者“页”。每页的数据量被定义为`LOBLKSIZE`（当前是`BLCKSZ/4`，或者通常是8K）。

`pg_largeobject`的每一行都保存一个大对象的一个页，从对象内的字节偏移量 (`pageno * LOBLKSIZE`) 开始。该实现允许稀疏存储：页面可以丢失，并且即使它们不是对象的最后一页也可能比`LOBLKSIZE`字节更短。大对象中缺少的区域被读作0。

Table 1. `pg_catalog.pg_largeobject`

| 列                   | 类型                 | 参考 | 描述                                                      |
|---------------------|--------------------|----|---------------------------------------------------------|
| <code>loid</code>   | <code>oid</code>   |    | 包含这一页的大对象的标识符。                                          |
| <code>pageno</code> | <code>int4</code>  |    | 大对象中该页的页号（从0开始计数）。                                      |
| <code>data</code>   | <code>bytea</code> |    | 存储在大对象中的实际数据。该数据不会超过 <code>LOBLKSIZE</code> 字节数而且可能会更小。 |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_listener系统目录表支持LISTEN和NOTIFY命令。 监听器在pg\_listener中为每个正在监听的通知名称创建一项。 通知者扫描和更新每个匹配的项以显示通知已经发生。 通知者也会发送一个信号（使用表中记录的PID）将监听器从睡眠中唤醒。

此表目前不在Greenplum数据库中使用。

**Table 1. pg\_catalog.pg\_listener**

| 列            | 类型   | 参考 | 描述                                            |
|--------------|------|----|-----------------------------------------------|
| relname      | name |    | 通知条件名称（该名称不需要匹配数据库中任何实际关系）。                   |
| listenerpid  | int4 |    | 创建这个项的服务器进程的PID。                              |
| notification | int4 |    | 如果这个监听器没有事件待处理，则为0。如果有待处理事件，则为发送通知的服务器进程的PID。 |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_namespace系统目录表存储命名空间。 命名空间是SQL模式的底层结构：每个命名空间都可以具有单独的没有命名冲突的关系、类型等集合。

Table 1. pg\_catalog.pg\_namespace

| 列        | 类型        | 参考            | 描述                   |
|----------|-----------|---------------|----------------------|
| oid      | oid       |               | 行标识符（隐藏属性；必须显式选择）    |
| nspname  | name      |               | 命名空间的名字              |
| nspowner | oid       | pg_authid.oid | 命名空间的所有者             |
| nspacl   | aclitem[] |               | 由GRANT和REVOKE给予的访问特权 |

Parent topic: 系统目录定义



Greenplum数据库® 6.0文档

- 参考指南

- SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

- 系统目录参考

系统表

系统视图

- 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_opclass`系统目录表定义索引访问方法操作符类。每个运算符类为特定数据类型的索引列和特定索引访问方法定义语义。运算符类实质上指定特定运算符族适用于特定的可索引列数据类型。系列中实际可用于索引列的运算符集是那些接受列的数据类型作为其左侧输入的运算符。

运算符类的`opcmethod`必须与其包含运算符族的`opfmethod`匹配。此外，对于任何给定的`opcmethod`和`opcintype`组合，必须有不超过一个`pg_opclass`行具有`opcdefault`为`true`。

**Table 1. pg\_catalog.pg\_opclass**

| 列                         | 类型                   | 参考                            | 描述                                                            |
|---------------------------|----------------------|-------------------------------|---------------------------------------------------------------|
| <code>oid</code>          | <code>oid</code>     |                               | 行标识符（隐藏属性；必须显式选择）                                             |
| <code>opcmethod</code>    | <code>oid</code>     | <code>pg_am.oid</code>        | 运算符类用的索引访问方法                                                  |
| <code>opcname</code>      | <code>name</code>    |                               | 运算符类的名称                                                       |
| <code>opcnamespace</code> | <code>oid</code>     | <code>pg_namespace.oid</code> | 运算符类所属的命名空间                                                   |
| <code>opcowner</code>     | <code>oid</code>     | <code>pg_authid.oid</code>    | 运算符类的拥有者                                                      |
| <code>opcfamily</code>    | <code>oid</code>     | <code>pg_opfamily.oid</code>  | 包含运算符类的运算符族                                                   |
| <code>opcintype</code>    | <code>oid</code>     | <code>pg_type.oid</code>      | 运算符类索引的数据类型                                                   |
| <code>opcdefault</code>   | <code>boolean</code> |                               | 如果此运算符类是数据类型 <code>opcintype</code> 的缺省值，则为 <code>True</code> |
| <code>opckeystype</code>  | <code>oid</code>     | <code>pg_type.oid</code>      | 存储在索引中的数据类型，如果与 <code>opcintype</code> 相同则                    |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

系统目录表pg\_operator存储关于操作符的信息，包括内建的和通过CREATE OPERATOR语句定义的操作符。未用的列包含零值。例如，一个前缀操作符的oprleft为0。

Table 1. pg\_catalog.pg\_operator

| 列            | 类型      | 参考               | 描述                                         |
|--------------|---------|------------------|--------------------------------------------|
| oid          | oid     |                  | 行标识符 (隐藏属性:必须显式选择)                         |
| oprname      | name    |                  | 操作符的名称                                     |
| oprnamespace | oid     | pg_namespace.oid | 操作符所属的名字空间的OID                             |
| oprowner     | oid     | pg_authid.oid    | 操作符的拥有者                                    |
| oprkind      | char    |                  | b = 中缀 (前缀和后缀), l = 前缀 ("左"), r = 后缀 ("右") |
| oprcanmerge  | boolean |                  | 此运算符是否支持合并连接                               |
| oprcanhash   | boolean |                  | 该操作符是否支持哈希连接                               |
| oprleft      | oid     | pg_type.oid      | 左操作数的类型                                    |
| oprright     | oid     | pg_type.oid      | 右操作数的类型                                    |
| oprresult    | oid     | pg_type.oid      | 结果类型                                       |
| oprcom       | oid     | pg_operator.oid  | 此运算符的转换器, 如果有的话                            |
| oprnegate    | oid     | pg_operator.oid  | 该操作符的求反器, 如果有的                             |
| oprcode      | regproc | pg_proc.oid      | 实现该操作符的函数                                  |

|         |         |             |                |
|---------|---------|-------------|----------------|
| oprrest | regproc | pg_proc.oid | 该运算符的限制选择性估计函数 |
| oprjoin | regproc | pg_proc.oid | 加入此运算符的选择性估计函数 |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

目录`pg_opfamily`定义了运算符系列。每个运算符族都是运算符和相关支持例程的集合，这些例程实现了为特定索引访问方法指定的语言。此外，一个系列中的运算符都以访问方法指定的方式兼容。运算符族概念允许跨数据类型的运算符与索引一起使用，并被推导为使用访问方法语义的知识。

定义运算符族的大多数信息不在其`pg_opfamily`行中，而是在`pg_amop`, `pg_amproc`和`pg_opclass`中的关联行中。

**Table 1. pg\_opfamily**

| 名称                        | 类型                | 参考                            | 描述                |
|---------------------------|-------------------|-------------------------------|-------------------|
| <code>oid</code>          | <code>oid</code>  |                               | 行标识符（隐藏属性：必须显式选择） |
| <code>opfmethod</code>    | <code>oid</code>  | <code>pg_am.oid</code>        | 此系列的索引访问方法运算符     |
| <code>opfname</code>      | <code>name</code> |                               | 此运算符系列的名称         |
| <code>opfnamespace</code> | <code>oid</code>  | <code>pg_namespace.oid</code> | 此运算符系列的命名空间       |
| <code>opfowner</code>     | <code>oid</code>  | <code>pg_authid.oid</code>    | 此运算符系列的所有者        |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

## □ 参考指南

## □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

## □ 系统目录参考

系统表

系统视图

## □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_partition`系统目录表被用来跟踪分区表以及它们的继承层级关系。  
`pg_partition`中的每一行要么代表了一个分区表在分区层级关系中的等级，要么是一个子分区模板的描述。`paristemplate`的属性值决定了一个特定行代表的含义。

Table 1. pg\_catalog.pg\_partition

| 列             | 类型         | 参考                          | 描述                                                         |
|---------------|------------|-----------------------------|------------------------------------------------------------|
| parrelid      | oid        | <code>pg_class.oid</code>   | 表的对象标识符。                                                   |
| parkind       | char       |                             | 分区类型 - R表示范围或 L表示列表。                                       |
| parlevel      | smallint   |                             | 该行的分区级别：0代表最顶层的父表，1代表父表下的第一个级别，2代表第二个级别，以此类推。              |
| paristemplate | boolean    |                             | 该行是否代表一个子分区模板定义 (true) 或者实际分区块级 (false)。                   |
| parnatts      | smallint   |                             | 定义该级别的属性数目。                                                |
| paratts       | smallint() |                             | 参与定义该级别的属性编号 (正如在 <code>pg_attribute.attnum</code> 中的) 数组。 |
| parclass      | oidvector  | <code>pg_opclass.oid</code> | 分区列上的操作符类的标识符。                                             |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_options

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_progress

gpexpand.status

`pg_partition_rule`系统目录表被用来跟踪分区表、它们的检查约束以及数据包含规则。`pg_partition_rule`表中的每一行要么代表了一个叶子分区（最底层包含数据的分区），要么是一个分支分区（用于定义分区层次的顶层或者中间层分区，但不包含数据）。

Table 1. pg\_catalog.pg\_partition\_rule

| 列                 | 类型       | 参考                                    | 描述                                                                                                                             |
|-------------------|----------|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| paroid            | oid      | <code>pg_partition.oid</code>         | 这个分区所属的分区级别的行标识符（来自 <code>pg_partition</code> ）。对于分支分区，相应的表（由 <code>pg_partition_rule</code> 标识）是一个空的容器表。对于叶子分区，这个表含有分区包含规则的行。 |
| parchildrelid     | oid      | <code>pg_class.oid</code>             | 分区（子表）的表标识符。                                                                                                                   |
| parparentrule     | oid      | <code>pg_partition_rule.paroid</code> | 与该分区的父表相关的规则的行标识符。                                                                                                             |
| parname           | name     |                                       | 该分区的给定名称。                                                                                                                      |
| parisdefault      | boolean  |                                       | 该分区是否为默认分区。                                                                                                                    |
| parruleord        | smallint |                                       | 对于范围分区表，该分区在分区层次的这个级别上的排名。                                                                                                     |
| parrangestartincl | boolean  |                                       | 对于范围分区表，是否包括起始值。                                                                                                               |
| parrangeendincl   | boolean  |                                       | 对于范围分区表，是否包括结束值。                                                                                                               |
| 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子句的间隔值。                                                                                                           |
| parlistvalues     | text     |                                       | 对于列表分区表，指派给该分区的表。                                                                                                              |
| parreloptions     | text     |                                       | 一个描述特定分区存储特性的数组。                                                                                                               |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

- 参考指南

- SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

- 系统目录参考

系统表

系统视图

- 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_pltemplate`系统目录表存储了过程语言的模板信息。一种语言的模板允许我们在一个特定数据库中以简单的CREATE LANGUAGE命令创建该语言，而不需要指定实现细节。和大部分系统目录不同，`pg_pltemplate`在Greenplum系统的所有数据库之间共享：在一个系统中只有一份`pg_pltemplate`拷贝，而不是每个数据库一份。这使得在每个需要的数据库中都可以访问该信息。

目前任何命令都不能操纵过程语言模板。要改变内建信息，超级用户必须使用普通的INSERT、DELETE或UPDATE命令修改该表。

**Table 1. pg\_catalog.pg\_pltemplate**

| 列             | 类型        | 参考 | 描述               |
|---------------|-----------|----|------------------|
| tmplname      | name      |    | 该模板适用的语言名称       |
| tmpltrusted   | boolean   |    | 如果该语言被认为可信的则为真   |
| tmplhandler   | text      |    | 调用处理器函数的名字       |
| tmplvalidator | text      |    | 验证器函数的名字，如果没有则为空 |
| tmpllibrary   | text      |    | 实现语言的共享库的路径      |
| tmplacl       | aclitem[] |    | 模板的访问特权（并未实现）。   |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_options

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_progress

gpexpand.status

`pg_proc`系统目录表存储关于函数（或过程）的信息，包括所有的内建函数以及由CREATE FUNCTION定义的函数。该表也包含了聚集和窗口函数以及普通函数的数据。如果`proisagg`为真，在`pg_aggregate`中应该有一个相匹配的行。

对于内置和动态加载的已编译函数，`prosrc`包含函数的C语言名称（链接符号）。对于所有其他当前已知的语言类型，`prosrc`包含函数的源文本。除了动态加载的C函数之外，`probin`是未使用的，它为此提供了包含该函数的共享库文件的名称。

Table 1. pg\_catalog.pg\_proc

| 列            | 类型      | 参考               | 描述                                                                       |
|--------------|---------|------------------|--------------------------------------------------------------------------|
| oid          | oid     |                  | 行标识符（隐藏属性：必须显式选择）                                                        |
| proname      | name    |                  | 函数的名字                                                                    |
| pronamespace | oid     | pg_namespace.oid | 函数所属的名字空间的OID                                                            |
| proowner     | oid     | pg_authid.oid    | 函数的拥有者                                                                   |
| prolang      | oid     | pg_language.oid  | 该函数的实现语言或调用接口                                                            |
| procost      | float4  |                  | 估计的执行代价（以cpu_operator_cost为单位），如果 <code>proretset</code> 为true，这是返回每行的代价 |
| prorows      | float4  |                  | 估计的结果行数（如果不是 <code>proretset</code> 则为零）                                 |
| provariadic  | oid     | pg_type.oid      | 可变数组参数的元素的数据类型，如果函数没有可变参数则为0                                             |
| protransform | regproc | pg_proc.oid      | 这个函数可以简化对此函数的调用                                                          |
| proisagg     | boolean |                  | 函数是否为一个聚集函数                                                              |
| proiswindow  | boolean |                  | □ 是否为一个窗口函数                                                              |
| prosecdef    | boolean |                  | 函数是一个安全性定义器（例如，一个"setuid"函数）                                             |
| proleakproof | boolean |                  | 该功能没有副作用。除了通过返回值之外，不会传达有关参数的信息。任何可能根据其参数的值抛出错误的函数都不是防漏的。                 |
| proisstrict  | boolean |                  | 当任意调用参数为空时，函数是否会返回空值。在那种情况下函数实际上根本不会被调用。非“strict”函数必须准备好处理空值输入。          |

|                              |                        |                          |                                                                                                                                                                                           |
|------------------------------|------------------------|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>proretset</code>       | <code>boolean</code>   |                          | 函数是否返回一个集合<br>(即, 指定数据类型的多个值)                                                                                                                                                             |
| <code>provolatile</code>     | <code>char</code>      |                          | 说明函数的结果是仅依赖于它的输入参数, 还是会被外部因素影响。值 <i>i</i> 表示“不变的”函数, 它对于相同的输入总是输出相同的结果。值 <i>s</i> 表示“稳定的”函数, 它的结果(对于固定输入)在一次扫描内不会变化。值 <i>v</i> 表示“不稳定的”函数, 它的结果在任何时候都可能变化(使用具有副作用, 结果可能在任何时刻发生改变)。        |
| <code>pronargs</code>        | <code>int2</code>      |                          | 输入参数的个数                                                                                                                                                                                   |
| <code>pronargdefaults</code> | <code>int2</code>      |                          | 具有默认值的参数个数                                                                                                                                                                                |
| <code>prorettype</code>      | <code>oid</code>       | <code>pg_type.oid</code> | 返回值的数据类型                                                                                                                                                                                  |
| <code>proargtypes</code>     | <code>oidvector</code> | <code>pg_type.oid</code> | 函数参数的数据类型的数组。这包括输入参数(包括INOUT和VARIADIC参数), 因此也表现了函数的调用特征。                                                                                                                                  |
| <code>proallargtypes</code>  | <code>oid[]</code>     | <code>pg_type.oid</code> | 函数参数的数据类型的数组。这包括所有参数(含OUT和INOUT参数)。不过, 如果所有参数都是IN参数, 这个属性将为空。注意下标是从1开始, 然而由于历史原因 <code>proargtypes</code> 的下标是从0开始。                                                                       |
| <code>proargmodes</code>     | <code>char[]</code>    |                          | 函数参数的模式的数据组: <i>i</i> 表示IN参数, <i>o</i> 表示OUT参数, <i>b</i> 表示INOUT参数, <i>v</i> 表示VARIADIC参数。如果所有的参数都是IN参数, 这个属性为空。注意这里的下标对应着 <code>proallargtypes</code> 而不是 <code>proargtypes</code> 中的位置。 |
| <code>proargnames</code>     | <code>text[]</code>    |                          | 函数参数的名字的数据组。没有名字的参数在数组中设置为空字符串。如果没有一个参数有名字, 这个属性为空。注意这里的下标对应着 <code>proallargtypes</code> 而不是 <code>proargtypes</code> 中的位置。                                                              |

|                 |              |  |                                                                                                                         |
|-----------------|--------------|--|-------------------------------------------------------------------------------------------------------------------------|
|                 |              |  | 置。                                                                                                                      |
| proargdefaults  | pg_node_tree |  | 表达式树<br>(在nodeToString()表示中) 用于默认参数值。这是一个带有pronargdefaults元素的列表，对应于最后N个输入参数(即最后N个proargtypes位置)。如果没有参数具有默认值，则此字段将为null。 |
| prosrc          | text         |  | 这个域告诉函数处理器如何调用该函数。它可能是针对解释型语言的实际源码、一个符号链接、一个文件名或任何其他东西，这取决于实现语言/调用规范。                                                   |
| probin          | text         |  | 关于如何调用函数的附加信息。同样，其解释是与语言相关的。                                                                                            |
| proconfig       | text[]       |  | 函数的运行时配置变量的本地设置。                                                                                                        |
| proacl          | aclitem[]    |  | GRANT/REVOKE给出的函数访问权限                                                                                                   |
| prodataaccess   | char         |  | 提供有关函数中包含的SQL语句类型的提示：n - 不包含SQL，c - 包含SQL，r - 包含读取数据的SQL，m - 包含修改数据的SQL                                                 |
| proexeclocation | char         |  | 函数在调用时执行的位置：m - 仅限master，a - 任何段实例，s - 所有段实例。                                                                           |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

Note: 仅当基于资源组的资源管理处于活动状态时, pg\_resgroupcapability系统目录表才有效。

pg\_resgroupcapability系统目录表包含有关已定义的Greenplum数据库资源组的功能和限制的信息。您可以按资源组对象ID将此表连接到[pg\\_resgroup](#) 表。

[pg\\_global](#)表空间中定义的pg\_resgroupcapability表在系统中的所有数据库之间全局共享。

Table 1. pg\_catalog.pg\_resgroupcapability

| 列            | 类型          | 参考                              | 描述                                                                                                      |
|--------------|-------------|---------------------------------|---------------------------------------------------------------------------------------------------------|
| resgroupid   | oid         | <a href="#">pg_resgroup.oid</a> | 关联资源组的对象ID。                                                                                             |
| reslimittype | smallint    |                                 | 资源组限制类型:<br>0 - 未知<br>1 - 并发度<br>2 - CPU<br>3 - 内存<br>4 - 内存共享配额<br>5 - 内存溢出比例<br>6 - 内存审查<br>7 - CPU集合 |
| value        | opaque type |                                 | 为此记录中引用的资源限制设置的特定值。此值具有固定类型文本, 并将根据引用的 <a href="#">口</a> 转换为不同的数据类型。                                    |
| proposed     | opaque      |                                 | 如果您更改了资                                                                                                 |

|  |      |                                                            |
|--|------|------------------------------------------------------------|
|  | type | 源限制并且无法立即更新限制，此记录中引用的限制的proposed值。否则，proposed反映当前设定的value。 |
|--|------|------------------------------------------------------------|

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_resourcetype`系统目录表包含有关可以分配给Greenplum数据库资源队列的扩展属性的信息。每行详细说明属性和固有质量，例如其默认设置，是否需要，以及禁用它的值（如果允许）。

此表仅在master服务器上填充。此表在`pg_global`表空间中定义，这意味着它在系统中的所有数据库中全局共享。

**Table 1. pg\_catalog.pg\_resourcetype**

| 列                               | 类型                    | 参考 | 描述                                                                             |
|---------------------------------|-----------------------|----|--------------------------------------------------------------------------------|
| <code>restypid</code>           | <code>smallint</code> |    | 资源类型ID。                                                                        |
| <code>resname</code>            | <code>name</code>     |    | 资源类型名称。                                                                        |
| <code>resrequired</code>        | <code>boolean</code>  |    | 资源类型是否是有效资源队列所必需的。                                                             |
| <code>reshasdefault</code>      | <code>boolean</code>  |    | 资源类型是否具有默认值。如果为 <code>true</code> ，则在 <code>resdefaultsetting</code> 中指定默认值。   |
| <code>rescanable</code>         | <code>boolean</code>  |    | 是否可以删除或禁用该类型。如果为 <code>true</code> ，则在 <code>resdisabledsetting</code> 中指定默认值。 |
| <code>resdefaultsetting</code>  | <code>text</code>     |    | 资源类型的默认设置（如果适用）。                                                               |
| <code>resdisabledsetting</code> | <code>text</code>     |    | 禁用此资源类型的值（如果允许）。                                                               |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

- 参考指南

- SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

- 系统目录参考

系统表

系统视图

- 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

Note: 仅当基于资源队列的资源管理处于活动状态时, pg\_resqueue系统目录表才有效。

pg\_resqueue系统目录表包含有关Greenplum数据库资源队列的信息, 这些队列用于资源管理功能。此表仅在master服务器上填充。此表在pg\_global表空间中定义, 这意味着它在系统中的所有数据库中全局共享。

Table 1. pg\_catalog.pg\_resqueue

| 列                  | 类型      | 参考 | 描述                                    |
|--------------------|---------|----|---------------------------------------|
| rsqname            | name    |    | 资源队列的名称。                              |
| rsqcountlimit      | real    |    | 资源队列的活跃查询阈值。                          |
| rsqcostlimit       | real    |    | 资源队列的查询开销阈值。                          |
| rsqovercommit      | boolean |    | 允许超出成本阈值的查询在系统空闲时运行。                  |
| rsqignorecostlimit | real    |    | 被视为“小查询”的查询成本限制。成本低于此限制的查询将不会排队并立即运行。 |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

Note: 仅当基于资源队列的资源管理处于活动状态时, `pg_resqueuecapability` 系统目录表才有效。

## □ 参考指南

### □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

## □ 系统目录参考

系统表

系统视图

## □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_resqueuecapability` 系统目录表包含有关现有Greenplum数据库资源队列的扩展属性或功能的信息。只有已分配了扩展功能的资源队列（例如优先级设置）才会记录在此表中。此表通过资源队列对象ID连接到`pg_resqueue` 表，并通过资源类型ID (`restypid`) 连接到`pg_resourcetype` 表。

此表仅在master服务器上填充。此表在`pg_global`表空间中定义，这意味着它在系统中的所有数据库中全局共享。

**Table 1. pg\_catalog.pg\_resqueuecapability**

| 列                      | 类型                       | 参考                                    | 描述                                               |
|------------------------|--------------------------|---------------------------------------|--------------------------------------------------|
| <code>rsqueueid</code> | <code>oid</code>         | <code>pg_resqueue.oid</code>          | 关联资源队列的对象ID。                                     |
| <code>restypid</code>  | <code>smallint</code>    | <code>pg_resourcetype.restypid</code> | 资源类型，派生自 <code>pg_resqueuecapability</code> 系统表。 |
| <code>resetting</code> | <code>opaque type</code> |                                       | 为此记录中引用的功能设置的特定值。根据实际资源类型，此值可能具有不同的数据类型。         |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_rewrite系统目录表存储表和视图的重写规则。如果一个表在这个目录中有任何规则，其pg\_class.relhasrules必须为真。

Table 1. pg\_catalog.pg\_rewrite

| 列          | 类型      | 参考           | 描述                                                        |
|------------|---------|--------------|-----------------------------------------------------------|
| rulename   | name    |              | 规则名称。                                                     |
| ev_class   | oid     | pg_class.oid | 使用该规则的表。                                                  |
| ev_attr    | int2    |              | 使用该规则的列（当前总是0，表示在整个表上使用）。                                 |
| ev_type    | char    |              | 使用该规则的事件类型：1 = SELECT, 2 = UPDATE, 3 = INSERT, 4 = DELETE |
| is_instead | boolean |              | 如果规则是一个INSTEAD规则，则为真。                                     |
| ev_qual    | text    |              | 规则条件的表达式树（按照nodeToString()的表现形式）。                         |
| ev_action  | text    |              | 规则动作的查询树（按照nodeToString()的表现形式）。                          |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

## □ 参考指南

## □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

## □ 系统目录参考

系统表

系统视图

## □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_shdepend系统目录表记录数据库对象和共享对象（例如角色）之间的依赖关系。这些信息使得Greenplum数据库可以确保对象在被删除时没有被其他对象引用。另见[pg\\_depend](#)，它对单个数据库中对象之间的依赖提供了相似的功能。与大部分其他系统目录不同，pg\_shdepend在Greenplum系统的所有数据库之间共享：在每一个系统中只有一份pg\_shdepend拷贝，而不是每个数据库一份。

在所有情况下，一个pg\_shdepend项表明被引用对象不能在没有删除其依赖对象的情况下被删除。但是，其中也有多种依赖类型，由deptype标识：

- **SHARED\_DEPENDENCY\_OWNER (o)** — 被引用对象（必须是一个角色）是依赖对象的拥有者。
- **SHARED\_DEPENDENCY\_ACL (a)** — 被引用对象（必须是一个角色）在依赖对象的ACL（访问控制列表）中被提到。
- **SHARED\_DEPENDENCY\_PIN (p)** — 没有依赖对象；这种类型的项是系统本身依赖被引用对象的信号，因此对象绝不能被删除。此类型的项仅通过系统初始化创建，依赖对象列包含0。

Table 1. pg\_catalog.pg\_shdepend

| 列          | 类型   | 参考              | 描述                              |
|------------|------|-----------------|---------------------------------|
| dbid       | oid  | pg_database.oid | 依赖对象所在的数据库OID，如果是一个共享对象则值为0。    |
| classid    | oid  | pg_class.oid    | 依赖对象所在的系统目录的OID。                |
| objid      | oid  | any OID column  | 特定依赖对象的OID。                     |
| objsubid   | int4 |                 | 对于一个表列，这将是列编号。对于所有其他对象类型，该列值为0。 |
| refclassid | oid  | pg_class.oid    | 被  对象所在的系统目录的OID（必须是一个共享的目      |

|             |      |                |                                      |
|-------------|------|----------------|--------------------------------------|
|             |      |                | 录)。                                  |
| refobjid    | oid  | any OID column | 被引用对象的OID。                           |
| refobjsubid | int4 |                | 对于一个表列，这将是被引用列的列编号。对于所有其他对象类型，该列值为0。 |
| deptype     | char |                | 一个定义该依赖关系的特定语义的代码。                   |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_shdescription`系统目录表存储共享数据库对象的可选描述（注释）。描述可以通过`COMMENT`命令操作，并且可以使`psql`的`\d`元命令来查看。另见`pg_description`，它对单个数据库中对象的描述提供了相似的功能。与大部分其他系统目录不同，`pg_shdescription`在Greenplum系统的所有数据库之间共享：在每一个系统中只有一份`pg_shdescription`拷贝，而不是每个数据库一份。

Table 1. pg\_catalog.pg\_shdescription

| 列                        | 类型                | 参考                          | 描述             |
|--------------------------|-------------------|-----------------------------|----------------|
| <code>objoid</code>      | <code>oid</code>  | <code>any OID column</code> | 该描述所属的对象的OID。  |
| <code>classoid</code>    | <code>oid</code>  | <code>pg_class.oid</code>   | 该对象所在系统目录的OID。 |
| <code>description</code> | <code>text</code> |                             | 作为该对象描述的任意文本。  |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_stat\_last\_operation表包含关于数据库对象（表、视图等）的元数据跟踪信息。

**Table 1. pg\_catalog.pg\_stat\_last\_operation**

| 列             | 类型                      | 参考             | 描述                                                          |
|---------------|-------------------------|----------------|-------------------------------------------------------------|
| classid       | oid                     | pg_class.oid   | 包含对象的系统目录的OID。                                              |
| objid         | oid                     | any OID column | 对象在其系统目录内的对象OID。                                            |
| staactionname | name                    |                | 在一个对象上采取的动作。                                                |
| stasysid      | oid                     | pg_authid.oid  | pg_authid.oid的外键。                                           |
| stausename    | name                    |                | 在该对象上执行操作的角色的名称。                                            |
| stasubtype    | text                    |                | 被执行操作的对象的类型或者被执行操作的子类。                                      |
| statime       | timestamp with timezone |                | 操作的时间戳。这和写到Greenplum数据库服务器日志文件的时间戳是相同的，以便在日志中查询更多关于操作细节的信息。 |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_stat\_last\_shoperation表包含全局对象（角色，表空间等）的元数据跟踪信息。

**Table 1. pg\_catalog.pg\_stat\_last\_shoperation**

| 列             | 类型                      | 参考             | 描述                                                           |
|---------------|-------------------------|----------------|--------------------------------------------------------------|
| classid       | oid                     | pg_class.oid   | 包含该对象的系统目录的OID。                                              |
| objid         | oid                     | any OID column | 系统目录中对象的OID。                                                 |
| staactionname | name                    |                | 对该对象采取的操作。                                                   |
| stasysid      | oid                     |                |                                                              |
| stausename    | name                    |                | 对此对象执行操作的角色的名称。                                              |
| stasubtype    | text                    |                | 操作对象的类型或执行的操作的子类。                                            |
| statime       | timestamp with timezone |                | 操作的时间戳。这与写入Greenplum数据库服务器日志文件的时间戳相同，以防您需要在日志中查找有关操作的更多详细信息。 |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_stat_replication`视图包含用于Greenplum数据库主镜像的walsender进程的元数据。

Table 1. pg\_catalog.pg\_stat\_replication

| 列                | 类型        | 参考 | 描述                                                         |
|------------------|-----------|----|------------------------------------------------------------|
| procpid          | integer   |    | WAL-sender后台进程的进程ID。                                       |
| usesysid         | integer   |    | 运行WAL-sender后台进程的用户系统ID。                                   |
| username         | name      |    | 运行WAL-sender后台进程的用户名。                                      |
| application_name | oid       |    | 客户端应用名。                                                    |
| client_addr      | name      |    | 客户端IP地址。                                                   |
| client_port      | integer   |    | 客户端端口号。                                                    |
| backend_start    | timestamp |    | 操作开始的时间戳。                                                  |
| state            | text      |    | WAL发送状态。可取值有:<br>startup<br>backup<br>catchup<br>streaming |
| sent_location    | text      |    | WAL-sender已发送的xlog记录位置。                                    |
| write_location   | text      |    | WAL-receiver的xlog记录写入位置。                                   |
| flush_location   | text      |    | WAL-receiver的xlog记录刷新位置。                                   |

|                 |      |  |                          |
|-----------------|------|--|--------------------------|
|                 |      |  | 录刷入位置。                   |
| replay_location | text |  | Standby xlog记录的重放位置。     |
| sync_priority   | text |  | 优先级，值为1。                 |
| sync_state      | text |  | WAL-sender的同步状态。该值为sync。 |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

## □ 参考指南

## □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

## □ 系统目录参考

系统表

系统视图

## □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_statistic`系统目录表存储有关数据库内容的统计数据。其中的项由`ANALYZE`创建，查询规划器会使用这些数据来进行查询规划。每个已经被分析的表列都有一项。注意所有的统计数据天然就是近似的，即使它刚刚被更新。

`pg_statistic`还存储有关索引表达式值的统计数据。这些被描述为它们是实际的数据列；特别是，`starelid`引用了索引。但是，没有为普通的非表达式索引列创建条目，因为它对于基础表列的条目是多余的。目前，索引表达式的条目始终具有`stainherit = false`。

当`stainherit = false`时，每个表格列通常有一个条目已被分析。如果表具有继承子项，则Greenplum数据库会创建第二个条目，其中`stainherit = true`。此行表示列在继承树上的统计信息，例如，您将使用`SELECT column FROM table*`看到的数据的统计信息，而`stainherit = false`行表示`SELECT column FROM ONLY table`表的结果。

由于不同类型的统计数据可能适用于不同类型的数据，因此`pg_statistic`不会非常假设它存储的是哪种统计数据。在`pg_statistic`中只给出了非常普通的统计信息（例如`nullness`）。其他所有内容都存储在插槽中，插槽是相关列的组，其内容由插槽列之一中的代码编号标识。

`pg_statistic`不应该被公众读取，因为即使关于表的内容的统计信息也可能被认为是敏感的（例如：工资列的最小值和最大值）。`pg_stats`是`pg_statistic`上的公共可读视图，它只公开有关当前用户可读的表的信息。

Table 1. `pg_catalog.pg_statistic`

| 列                        | 类型                  | 参考                               | 描述                                               |
|--------------------------|---------------------|----------------------------------|--------------------------------------------------|
| <code>starelid</code>    | <code>oid</code>    | <code>pg_class.oid</code>        | 被描述列所属的表或索引。                                     |
| <code>staattnum</code>   | <code>int2</code>   | <code>pg_attribute.attnum</code> | 被描述列的编号。                                         |
| <code>stainherit</code>  | <code>bool</code>   |                                  | 如果为 <code>true</code> ，则统计信息包括继承子列，而不仅仅是指定关系中的值。 |
| <code>stanullfrac</code> | <code>float4</code> |                                  | 列项为空的比例。                                         |
| <code>stawidth</code>    | <code>int4</code>   |                                  | 非空项的平均存储宽度，以字节计。                                 |
| <code>stadistinct</code> | <code>float4</code> |                                  | 列中不同的非空数据                                        |

|                          |                       |                              |  |                                                                                                          |
|--------------------------|-----------------------|------------------------------|--|----------------------------------------------------------------------------------------------------------|
|                          |                       |                              |  | 值的数量。大于零的值是不同值的实际数量。小于零的值是表中行数的乘数的负值（例如，其中值平均显示两次的列可以由 <code>stadistinct = -0.5</code> 表示）。零值表示不同值的数量未知。 |
| <code>stakindN</code>    | <code>int2</code>     |                              |  | 一个代码，它表示存储在该 <code>pg_statistic</code> 行中第N个“槽位”的统计类型。                                                   |
| <code>staopN</code>      | <code>oid</code>      | <code>pg_operator.oid</code> |  | 一个用于生成这些存储在第N个“槽位”的统计信息的操作符。比如，一个柱状图槽位会用<操作符，该操作符定义了该数据的排序顺序。                                            |
| <code>stanumbersN</code> | <code>float4[]</code> |                              |  | 第N个“槽位”的类型的数值类型统计，如果该槽位不涉及数值类型则为NULL。                                                                    |
| <code>stavalueN</code>   | <code>anyarray</code> |                              |  | 第N个“槽位”的类型的列值，如果该槽位类型不存储任何数据值则为NULL。每个数组的元素值实际上都是指定列的数据类型，因此，除了把这些列的类型定义成 <code>anyarray</code> 之外别无他法。  |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_tablespace`系统目录表存储关于可用表空间的信息。表可以被放置在特定表空间中以实现磁盘布局的管理。与大部分其他系统目录不同，`pg_tablespace`在Greenplum系统的所有数据库之间共享：在每一个系统中只有一份`pg_tablespace`的拷贝，而不是每个数据库一份。

Table 1. pg\_catalog.pg\_tablespace

| 列          | 类型        | 参考            | 描述                 |
|------------|-----------|---------------|--------------------|
| spcname    | name      |               | 表空间名。              |
| spcowner   | oid       | pg_authid.oid | 表空间的拥有者，通常是创建它的用户。 |
| spcacl     | aclitem[] |               | 表空间访问权限。           |
| spcoptions | text[]    |               | 表空间contentID位置。    |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_trigger系统目录表存储表上的触发器。

Note: Greenplum数据库不支持触发器。

Table 1. pg\_catalog.pg\_trigger

| 列                 | 类型      | 参考                                                             | 描述                                                                  |
|-------------------|---------|----------------------------------------------------------------|---------------------------------------------------------------------|
| tgrelid           | oid     | <i>pg_class.oid</i><br>请注意<br>意, Greenplum数<br>据库不强制引用<br>完整性。 | 触发器所在的<br>表。                                                        |
| tgname            | name    |                                                                | 触发器名 (同<br>一个表的触发<br>器名必须唯<br>一)。                                   |
| tgfoid            | oid     | <i>pg_proc.oid</i><br>请注意<br>意, Greenplum数<br>据库不强制引用<br>完整性。  | 被触发器调用<br>的函数。                                                      |
| tgttype           | int2    |                                                                | 触发器触发条<br>件的位掩码。                                                    |
| tgenabled         | boolean |                                                                | 启用触发器则<br>为True。                                                    |
| tgisinternal      | boolean |                                                                | 如果触发器是<br>内部生成的,<br>则为True (通<br>常, 强制执<br>行tgconstraint标<br>识的约束)。 |
| tgconstraintrelid | oid     | <i>pg_class.oid</i><br>请注意<br>意, Greenplum数<br>据库不强制引用<br>完整性。 | 被一个参照完<br>整的表引用<br>的                                                |

|                |            |  |                           |
|----------------|------------|--|---------------------------|
| tgdeferrable   | boolean    |  | 如果可延迟则为True。              |
| tginitdeferred | boolean    |  | 如果初始可延迟则为True。            |
| tgnargs        | int2       |  | 传递给触发器函数的参数字符串个数。         |
| tgattr         | int2vector |  | 当前没有使用。                   |
| tgargs         | bytea      |  | 传递给触发器的参数字符串，每一个都以NULL结尾。 |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

系统目录表pg\_user\_mapping存储从本地用户到远程用户的映射。您必须具有管理员权限才能查看此目录。普通用户仅限访问此目录，请改用pg\_user\_mappings视图。

Table 1. pg\_catalog.pg\_user\_mapping

| 列         | 类型     | 参考                    | 描述                                 |
|-----------|--------|-----------------------|------------------------------------|
| umuser    | oid    | pg_authid.oid         | 要映射的本地角色的OID，如果用户映射是公共的，则为0。       |
| umserver  | oid    | pg_foreign_server.oid | 包含此映射的外部服务器的OID。                   |
| umoptions | text[] |                       | 特定于用户映射的选项，作为“keyword = value”字符串。 |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`gp_distributed_log`视图包含有关分布式事务及其关联的本地事务的状态信息。 分布式事务是涉及修改Segment实例上数据的事务。 Greenplum的分布式事务管理器确保了这些Segment保持同步。 此视图允许用户查看分布式事务的状态。

Table 1. pg\_catalog(gp\_distributed\_log)

| 列                 | 类型        | 参考                               | 描述                                   |
|-------------------|-----------|----------------------------------|--------------------------------------|
| segment_id        | smallint  | gp_segment_configuration.content | 如果是Segment，则是其内容ID。Master总是为-1（无内容）。 |
| dbid              | small_int | gp_segment_configuration.dbid    | Segment实例的唯一ID。                      |
| distributed_xid   | xid       |                                  | 全局事务ID。                              |
| distributed_id    | text      |                                  | 分布式事务的系统分配ID。                        |
| status            | text      |                                  | 分布式事务的状态（提交或者中止）。                    |
| local_transaction | xid       |                                  | 本地事务ID。                              |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

gp\_distributed\_xacts视图包含有关Greenplum数据库分布式事务的信息。分布式事务是涉及修改Segment实例上数据的事务。Greenplum的分布式事务管理器确保了这些Segment保持同步。此视图允许用户查看当前活动的会话及其关联的分布式事务。

**Table 1. pg\_catalog(gp\_distributed\_xacts**

| 列                         | 类型   | 参考 | 描述                                         |
|---------------------------|------|----|--------------------------------------------|
| distributed_xid           | xid  |    | 分布式事务在Greenplum数据库阵列中使用的事务ID。              |
| distributed_id            | text |    | 分布式事务标识符。它有2个部分 - 一个唯一的时间戳和分布式事务编号。        |
| state                     | text |    | 本次会话对于分布式事务的当前状态。                          |
| gp_session_id             | int  |    | 与此事务关联的Greenplum数据库会话的ID编号。                |
| xmin_distributed_snapshot | xid  |    | 在此事务开始时，所有打开事务中发现的最小分布式事务号。它用于MVCC分布式快照目的。 |

**Parent topic:** 系统目录定义

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_options

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_progress

gpexpand.status

`gp_pgdatabase`视图显示Greenplum中Segment实例的状态信息，以及它们是作为镜像Segment还是主Segment。Greenplum的故障检测和恢复工具在内部使用此视图来确定失效的Segment。

Table 1. pg\_catalog.gp\_pgdatabase

| 列              | 类型       | 参考                                      | 描述                                                                                     |
|----------------|----------|-----------------------------------------|----------------------------------------------------------------------------------------|
| dbid           | smallint | gp_segment_configuration.dbid           | 系统分配的ID。Segment（或者Master）实例的唯一标识符。                                                     |
| isprimary      | boolean  | gp_segment_configuration.role           | 该实例是否处于活动状态。它目前是否作为主Segment（还是镜像Segment）。                                              |
| content        | smallint | gp_segment_configuration.content        | 实例上部分数据的ID。主Segment实例及其镜像将具有相同的内容ID。<br>对于Segment，值为0-N-1，其中N是Greenplum数据库中的Segment数量。 |
| definedprimary | boolean  | gp_segment_configuration.preferred_role | 对于Master，值为-1。<br>在初始化系统时，该实例是否被定义为主Segment（而不是镜像Segment）。                             |

Parent topic: [系统目录定义](#)





Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`gp_toolkit.gp_resgroup_status`视图允许管理员查看资源组的状态和活动。它显示了等待运行的查询数以及系统中每个资源组当前有多少查询处于活动状态。该视图还显示资源组的当前内存和CPU使用情况。

Note: 仅当基于资源组的资源管理处于活动状态时, `gp_resgroup_status`视图才有效。

**Table 1. gp\_toolkit.gp\_resgroup\_status**

| 列                    | 类型       | 参考                               | 描述                                                               |
|----------------------|----------|----------------------------------|------------------------------------------------------------------|
| rsgname              | name     | <code>pg_resgroup.rsgname</code> | 资源组的名称。                                                          |
| groupid              | oid      | <code>pg_resgroup.oid</code>     | 资源组的ID。                                                          |
| num_running          | integer  |                                  | 当前在资源组中执行的事务数。                                                   |
| num_queueing         | integer  |                                  | 资源组的当前排队事务数。                                                     |
| num_queued           | integer  |                                  | 自Greenplum数据库集群上次启动以来资源组的排队事务总数, 不包括 <code>num_queueing</code> 。 |
| num_executed         | integer  |                                  | 自Greenplum数据库集群上次启动以来资源组中执行的事务总数, 不包括 <code>num_running</code> 。 |
| total_queue_duration | interval |                                  | 自上次启动Greenplum数据库集群以来, 任何事务排队的总时间。                               |
| cpu_usage            | json     |                                  | 每个Greenplum数据库网段主机上资源组的实时CPU使用情况。                                |
| memory_usage         | json     |                                  | 每个Greenplum数据库segment上资源组的实时内存使用情况。                              |

`cpu_usage`字段是JSON格式的key: value字符串, 用于为每个资源组标识每个segment的CPU使用率百分比。key是segment ID, value是segment主机上资源组的CPU使用率百分比。在segment主机上运行的所有segment的CPU总使用率不应超过`gp_resource_group_cpu_limit`。示例`cpu_usage`列输出:

```
{""-1":0.01, "0":0.31, "1":0.31}
```

`memory_usage`字段也是JSON格式的key: value字符串。字符串内容因资源组的类型而异。对于分配给角色的每个资源组（默认内存审核器`vmtracker`），此字符串标识每个segment上已使用，可用，已授予和建议的固定和共享内存配额分配。`key`是segment ID，`value`是以MB为单位显示的内存值。下面的示例显示分配给角色的资源组的单个segment的`memory_usage`列输出：

```
"0": { "used": 0, "available": 76, "quota_used": -1, "quota_available": 60,
"quota_granted": 60, "quota_proposed": 60, "shared_used": 0,
"shared_available": 16, "shared_granted": 16, "shared_proposed": 16}
```

对于分配给外部组件的每个资源组，`memory_usage` JSON格式的字符串标识了每个segment上使用的内存和内存限制。以下示例显示单个segment的外部组件资源组的`memory_usage`列输出：

```
"1": { "used": 11, "limit_granted": 15}
```

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

- 参考指南

- SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

- 系统目录参考

系统表

系统视图

- 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`gp_toolkit.gp_resqueue_status`视图允许管理员查看资源队列的状态和活动。它显示了有多少查询正在等待运行以及系统中当前有多少查询来自特定资源队列。

Note: The `gp_resqueue_status` view is valid only when resource queue-based resource management is active.

**Table 1. `gp_toolkit.gp_resqueue_status`**

| 列             | 类型   | 参考                                                | 描述                     |
|---------------|------|---------------------------------------------------|------------------------|
| queueid       | oid  | <code>gp_toolkit.gp_resqueue_queueid</code>       | 资源队列的ID。               |
| rsqname       | name | <code>gp_toolkit.gp_resqueue_rsqname</code>       | 资源队列的名称。               |
| rsqcountlimit | real | <code>gp_toolkit.gp_resqueue_rsqcountlimit</code> | 资源队列的活动查询阈值。值-1表示没有限制。 |
| rsqcountvalue | real | <code>gp_toolkit.gp_resqueue_rsqcountvalue</code> | 当前在资源队列中使用的活动查询槽的数量。   |
| rsqcostlimit  | real | <code>gp_toolkit.gp_resqueue_rsqcostlimit</code>  | 资源队列的查询开销阈值。值-1表示没有限制。 |
| rsqcostvalue  | real | <code>gp_toolkit.gp_resqueue_rsqcostvalue</code>  | 当前在资源队列中的              |

|                |         |                                       |  |                      |
|----------------|---------|---------------------------------------|--|----------------------|
|                |         |                                       |  | 所有语句的总成本。            |
| rsqmemorylimit | real    | gp_toolkit.gp_resqueue_rsqmemorylimit |  | 资源队列的内存限制。           |
| rsqmemoryvalue | real    | gp_toolkit.gp_resqueue_rsqmemoryvalue |  | 当前在资源队列中的所有语句使用的总内存。 |
| rsqwaiters     | integer | gp_toolkit.gp_resqueue_rsqwaiter      |  | 当前在资源队列中等待的语句数。      |
| rsqholders     | integer | gp_toolkit.gp_resqueue_rsqholders     |  | 当前在此资源队列中在系统上运行的语句数。 |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

gp\_transaction\_log视图包含特定Segment的本地事务的状态信息。此视图允许用户查看本地事务的状态。

Table 1. pg\_catalog(gp\_transaction\_log)

| 列           | 类型       | 参考                               | 描述                                                      |
|-------------|----------|----------------------------------|---------------------------------------------------------|
| segment_id  | smallint | gp_segment_configuration.content | 如果<br>是Segment，则<br>是内<br>容ID。Master总<br>为-1(没有内<br>容)。 |
| dbid        | smallint | gp_segment_configuration.dbid    | Segment实例的<br>唯一ID。                                     |
| transaction | xid      |                                  | 本地事务<br>的ID。                                            |
| status      | text     |                                  | 本地事务的状<br>态(提交或者<br>中止)。                                |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

`gpexpand.expansion_progress`视图包含有关系系统扩容操作状态的信息。该表提供了表重新分布速度和完成扩容的时间的估算。

## □ 参考指南

### □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

## □ 系统目录参考

系统表

系统视图

### □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_progress

gpexpand.status

**Table 1. gpexpand.expansion\_progress**

| 列     | 类型   | 参考 | 描述                                                                   |
|-------|------|----|----------------------------------------------------------------------|
| name  | text |    | 提供的数据域的名称，包括:<br>剩余字节数<br>完成字节数<br>扩展速度估计<br>估计完成时间<br>已扩展的表<br>剩余的表 |
| value | text |    | 进度数据的值。例如：<br>扩展速度估计 -<br>9.75667095996092<br>MB/s                   |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_max\_external\_files视图显示使用外部表文件协议时每台Segment主机允许的外部表文件最大数量。

Table 1. pg\_catalog.pg\_max\_external\_files

| 列        | 类型     | 参考 | 描述                               |
|----------|--------|----|----------------------------------|
| hostname | name   |    | 在Segment主机上访问特定Segment实例所使用的主机名。 |
| maxfiles | bigint |    | 该主机上的主Segment实例数。                |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_partition\_columns系统视图被用来显示一个分区表的分区键列。

**Table 1. pg\_catalog.pg\_partition\_columns**

| 列                         | 类型       | 参考 | 描述                                 |
|---------------------------|----------|----|------------------------------------|
| schemaname                | name     |    | 该分区表所在schema的名称。                   |
| tablename                 | name     |    | 顶层父表的表名。                           |
| columnname                | name     |    | 分区键列的名称。                           |
| partitionlevel            | smallint |    | 该子分区在层次中的级别。                       |
| position_in_partition_key | integer  |    | 对于列表分区可以有组合(多列)分区键。此处显示了列在组合键中的位置。 |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_partition\_templates系统视图用来显示通过子分区模板创建的子分区。

Table 1. pg\_catalog.pg\_partition\_templates

| 列                   | 类型       | 参考 | 描述                                                                            |
|---------------------|----------|----|-------------------------------------------------------------------------------|
| schemaname          | name     |    | 该分区表所在的schema的名称。                                                             |
| tablename           | name     |    | 顶层父表的表名。                                                                      |
| partitionname       | name     |    | 子分区的名称（如果在ALTER TABLE命令中引用分区，用的就是这个名称）。如果该分区在创建时或者通过EVERY子句产生时没有给定一个名称则为NULL。 |
| partitiontype       | text     |    | 子分区的类型（范围或者列表）。                                                               |
| partitionlevel      | smallint |    | 该分区在层次中的级别。                                                                   |
| partitionrank       | bigint   |    | 对于范围分区，该分区相对于同级中其他分区的排名。                                                      |
| partitionposition   | smallint |    | 该子分区的规则位置。                                                                    |
| partitionlistvalues | text     |    | 对于列表分区，该子分区相关的列表                                                              |

|                         |         |  |                        |
|-------------------------|---------|--|------------------------|
|                         |         |  | 值。                     |
| partitionrangestart     | text    |  | 对于范围分区，该子分区的开始值。       |
| partitionstartinclusive | boolean |  | 如果该子分区包含了起始值值则为T，否则为F。 |
| partitionrangeend       | text    |  | 对于范围分区，该子分区的结束值。       |
| partitionendinclusive   | boolean |  | 如果该子分区包含了结束值则为T，否则为F。  |
| partitioneveryclause    | text    |  | 该子分区的EVERY子句(间隔)。      |
| partitionisdefault      | boolean |  | 如果这是一个默认子分区则为T，否则为F。   |
| partitionboundary       | text    |  | 该子分区的整个分区说明。           |

**Parent topic:** 系统目录定义

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

Note: 仅当基于资源队列的资源管理处于活动状态时, pg\_resqueue\_attributes视图才有效。

pg\_resqueue\_attributes视图允许管理员查看为资源队列设置的属性, 例如其活动语句限制, 查询成本限制和优先级。

**Table 1. pg\_catalog.pg\_resqueue\_attributes**

| 列         | 类型      | 参考                  | 描述           |
|-----------|---------|---------------------|--------------|
| rsqname   | name    | pg_resqueue.rsqname | 资源队列的名称。     |
| resname   | text    |                     | 资源队列属性的名称。   |
| resetting | text    |                     | 资源队列属性的当前值。  |
| restypid  | integer |                     | 系统分配的资源类型ID。 |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

- 参考指南

- SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

- 系统目录参考

系统表

系统视图

- 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

Note: 仅当基于资源队列的资源管理处于活动状态时, pg\_stat\_resqueues视图才有效。

pg\_stat\_resqueues视图允许管理员查看有关资源队列工作负载的指标。要允许为此视图收集统计信息, 必须在Greenplum数据库master实例上启用stats\_queue\_level服务器配置参数。启用这些度量标准的收集会导致性能损失很小, 因为通过资源队列提交的每个语句都必须记录在系统目录表中。

**Table 1. pg\_catalog.pg\_stat\_resqueues**

| 列              | 类型     | 参考 | 描述                             |
|----------------|--------|----|--------------------------------|
| queueoid       | oid    |    | 资源队列的OID。                      |
| queuename      | name   |    | 资源队列的名称。                       |
| n_queries_exec | bigint |    | 已提交的要从该资源队列中执行的查询数目。           |
| n_queries_wait | bigint |    | 已提交到该资源队列中, 在执行之前必须进行等待的查询数目。  |
| elapsed_exec   | bigint |    | 通过该资源队列提交的语句的总执行时间。            |
| elapsed_wait   | bigint |    | 通过该资源队列提交的语句, 在执行之前必须进行等待的总时间。 |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_options

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

foreign\_data\_wrapper\_options视图包含为当前数据库中的外部数据包装器定义的所有选项。 Greenplum数据库仅显示当前用户可以访问的外部数据包装器（通过作为所有者或具有某些权限）。

Table 1. foreign\_data\_wrapper\_options

| 列                            | 类型             | 参考 | 描述                           |
|------------------------------|----------------|----|------------------------------|
| foreign_data_wrapper_catalog | sql_identifier |    | 定义外部数据包装器的数据仓库的名称（始终是当前数据库）。 |
| foreign_data_wrapper_name    | sql_identifier |    | 外部数据包装器的名称。                  |
| option_name                  | sql_identifier |    | 选项的名称。                       |
| option_value                 | character_data |    | 选项的值。                        |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`foreign_data_wrappers`视图包含当前数据库中定义的所有外部数据包装器。Greenplum数据库仅显示当前用户可以访问的外部数据包装器（通过作为所有者或具有某些权限）。

Table 1. `foreign_data_wrappers`

| 列                                          | 类型                          | 参考 | 描述                          |
|--------------------------------------------|-----------------------------|----|-----------------------------|
| <code>foreign_data_wrapper_catalog</code>  | <code>sql_identifier</code> |    | 定义外部数据包装器的数据库的名称（始终是当前数据库）。 |
| <code>foreign_data_wrapper_name</code>     | <code>sql_identifier</code> |    | 外部数据包装器的名称。                 |
| <code>authorization_identifier</code>      | <code>sql_identifier</code> |    | 外部服务器所有者的名称。                |
| <code>library_name</code>                  | <code>character_data</code> |    | 实现此外部数据包装器的库的文件名。           |
| <code>foreign_data_wrapper_language</code> | <code>character_data</code> |    | 用于实现外部数据包装器的语言。             |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

foreign\_servers视图包含当前数据库中定义的所有外部服务器。Greenplum数据库仅显示当前用户可以访问的外部服务器（通过作为所有者或具有某些权限）。foreign\_server\_options视图包含为当前数据库中的外部服务器定义的所有选项。Greenplum数据库仅显示当前用户可以访问的外部服务器（通过作为所有者或具有某些权限）。

**Table 1. foreign\_server\_options**

| 列                      | 类型             | 参考 | 描述                        |
|------------------------|----------------|----|---------------------------|
| foreign_server_catalog | sql_identifier |    | 定义外部服务器的数据库的名称（始终是当前数据库）。 |
| foreign_server_name    | sql_identifier |    | 外部服务器的名称。                 |
| option_name            | sql_identifier |    | 选项的名称。                    |
| option_value           | character_data |    | 选项的值。                     |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`foreign_servers`视图包含当前数据库中定义的所有外部服务器。Greenplum数据库仅显示当前用户可以访问的外部服务器（通过作为所有者或具有某些权限）。

Table 1. `foreign_servers`

| 列                                         | 类型             | 参考 | 描述                                 |
|-------------------------------------------|----------------|----|------------------------------------|
| <code>foreign_server_catalog</code>       | sql_identifier |    | 定义外部服务器的数据的名称（始终是当前数据库）。           |
| <code>foreign_server_name</code>          | sql_identifier |    | 外部服务器的名称。                          |
| <code>foreign_data_wrapper_catalog</code> | sql_identifier |    | 定义外部服务器使用的外部数据包装器的数据的名称（始终是当前数据库）。 |
| <code>foreign_data_wrapper_name</code>    | sql_identifier |    | 外部服务器使用的外部数据包装器的名称。                |
| <code>foreign_server_type</code>          | character_data |    | 外部服务器类                             |

|                          |                |  |                     |
|--------------------------|----------------|--|---------------------|
|                          |                |  | 型信息，如果在创建时指定。       |
| foreign_server_version   | character_data |  | 外部服务器版本信息，如果在创建时指定。 |
| authorization_identifier | sql_identifier |  | 外部服务器所有者的名称。        |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`foreign_table_options`视图包含为当前数据库中的外部表定义的所有选项。 Greenplum数据库仅显示当前用户可以访问的外部表（通过作为所有者或具有某些权限）。

**Table 1. `foreign_table_options`**

| 列                                  | 类型                          | 参考 | 描述                      |
|------------------------------------|-----------------------------|----|-------------------------|
| <code>foreign_table_catalog</code> | <code>sql_identifier</code> |    | 定义外部表的数据库的名称（始终是当前数据库）。 |
| <code>foreign_table_schema</code>  | <code>sql_identifier</code> |    | 包含外部表的schema的名称。        |
| <code>foreign_table_name</code>    | <code>sql_identifier</code> |    | 外部表的名称。                 |
| <code>option_name</code>           | <code>sql_identifier</code> |    | 选项的名称。                  |
| <code>option_value</code>          | <code>character_data</code> |    | 选项的值。                   |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

foreign\_tables视图包含当前数据库中定义的所有外部表。Greenplum数据库仅显示当前用户可以访问的外部表（通过作为所有者或具有某些权限）。

Table 1. foreign\_tables

| 列                      | 类型             | 参考 | 描述                        |
|------------------------|----------------|----|---------------------------|
| foreign_table_catalog  | sql_identifier |    | 定义外部表的数据库的名称（始终是当前数据库）。   |
| foreign_table_schema   | sql_identifier |    | 包含外部表的schema的名称。          |
| foreign_table_name     | sql_identifier |    | 外部表的名称。                   |
| foreign_server_catalog | sql_identifier |    | 定义外部服务器的数据库的名称（始终是当前数据库）。 |
| foreign_server_name    | sql_identifier |    | 外部服务器的名称。                 |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

pg\_attribute\_encoding系统目录表包含列存储信息。

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

**Table 1. pg\_catalog.pg\_attribute\_encoding**

| 列          | 类型       | 修正值      | 存储       | 描述                           |
|------------|----------|----------|----------|------------------------------|
| attrelid   | oid      | not null | plain    | 外键<br>于pg_attribute.attrelid |
| attnum     | smallint | not null | plain    | 外键<br>于pg_attribute.attnum   |
| attoptions | text [ ] |          | extended | 选项                           |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_compression系统目录表描述了可用的压缩方法。

Table 1. pg\_catalog.pg\_compression

| 列                | 类型      | 修饰符      | 存储    | 描述            |
|------------------|---------|----------|-------|---------------|
| compname         | name    | not null | plain | 压缩方法的名字       |
| compconstructor  | regproc | not null | plain | 压缩方法构造函数名称    |
| compdestructor   | regproc | not null | plain | 压缩方法析构函数名称    |
| compcompressor   | regproc | not null | plain | 压缩器的名称        |
| compdecompressor | regproc | not null | plain | 解压缩器的名称       |
| compvalidator    | regproc | not null | plain | 压缩验证器的名称      |
| compowner        | oid     | not null | plain | pg_authid的oid |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

该pg\_enum表包含与其相关联的值和标签匹配的枚举类型条目。给定枚举值的内部表示其实是pg\_enum表中相关行的OID。特定枚举类型的Oid保证按照应有类型排序方式进行排序，但是不保证不相关枚举类型Oid的排序。

**Table 1. pg\_catalog.pg\_enum**

| 列         | 类型   | 参考         | 描述                   |
|-----------|------|------------|----------------------|
| enumtypid | oid  | pgtype.oid | 拥有此枚举类型pg_type条目的OID |
| enumlabel | name |            | 此枚举值的文本标签            |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

该系统目录表pg\_extension存储了关于安装的扩展的信息。

**Table 1. pg\_catalog.pg\_extension**

| 列              | 类型      | 参考               | 描述                                   |
|----------------|---------|------------------|--------------------------------------|
| extname        | name    |                  | 扩展的名称。                               |
| extowner       | oid     | pg_authid.oid    | 扩展所有者                                |
| extnamespace   | oid     | pg_namespace.oid | 包含扩展导出对象的Schema                      |
| extrelocatable | boolean |                  | 如果该扩展能够重定位到其他schema，则为真。             |
| extversion     | text    |                  | 扩展的版本名字                              |
| extconfig      | oid[]   | pg_class.oid     | 扩展配置表的regclass OIDs数组，如果为空则为NULL。    |
| extcondition   | text[]  |                  | 扩展配置表的WHERE-clause过滤条件数组，如果没有则为NULL。 |

与大多数命名空间列的目录不同，extnamespace并不意味着该扩展属于这个schema。扩展的名称不符合schema限制。extnamespace schema指示其包含大部分或全部扩展对象的schema。如果extrelocatable为true，则该schema必须包含所有属于扩展名的符合schema限定的对象。

**Parent topic:** [系统目录定义](#)



## Greenplum数据库® 6.0文档

## □ 参考指南

## □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

## □ 系统目录参考

系统表

系统视图

## □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_locks`视图提供了有关在Greenplum数据库中由打开的事务持有的锁的信息的访问。

`pg_locks`包含一行关于每个积极可锁对象，请求的锁模式和相关事务。因此，如果多个事务正在持有或等待其上的锁，同样的可锁对象可能会出现多次。但是，目前没有锁的对象根本就不会出现。

存在几种不同类型的可锁定对象：完整关系（例如表），关系的各个页面，关系的单个元组，事务ID（虚拟和永久ID）以及通用数据库对象。此外，扩展关系的权利表示为单独的可锁定对象。

Table 1. pg\_catalog.pg\_locks

| 列             | 类型       | 参考                           | 描述                                                                                                                                                                                                                             |
|---------------|----------|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| locktype      | text     |                              | 可锁对象的类型: <code>relation</code> , <code>extend</code> , <code>page</code> , <code>tuple</code> , <code>transactionid</code> , <code>object</code> , <code>userlock</code> , <code>resource queue</code> 或 <code>advisory</code> |
| database      | oid      | <code>pg_database.oid</code> | 该对象存在的数据库的Oid，如果该对象是共享对象，则为0。如果对象是事务ID，则为空。                                                                                                                                                                                    |
| relation      | oid      | <code>pg_class.oid</code>    | 关系的OID，如果对象不是关系或者关系的一部分，则为NULL。                                                                                                                                                                                                |
| page          | integer  |                              | 关系中的页码，如果对象不是元组或者关系页则为NULL                                                                                                                                                                                                     |
| tuple         | smallint |                              | 页中的元组号，如果该对象不是个元组则为NULL                                                                                                                                                                                                        |
| virtualxid    | text     |                              | 事务的虚拟ID，如果对象不是虚拟事务ID，则为NULL                                                                                                                                                                                                    |
| transactionid | xid      |                              | 事务的ID，如果对象不是一个事务ID，则为NULL                                                                                                                                                                                                      |
| classid       | oid      | <code>pg_class.oid</code>    | 包含对象的系统目录的OID，如果对象不是一般数据库对象，则为NULL                                                                                                                                                                                             |
| objid         | oid      | any OID column               | 其系统目录中对象的OID，如果对象不是一                                                                                                                                                                                                           |

|                    |          |  |                                                                       |
|--------------------|----------|--|-----------------------------------------------------------------------|
|                    |          |  | 般数据库对象，则为NULL                                                         |
| objsubid           | smallint |  | 对一个表列来说，这是列号（ classid和objid引用表本身）。对于所有其他的对象类型，此列为0。如果对象不是数据库对象，则为NULL |
| virtualtransaction | text     |  | 持有或等待此锁定的事务的虚拟ID                                                      |
| pid                | integer  |  | 持有或等待该锁的事务进程的进程Id。如果锁定由准备好的事务持有，则为NULL                                |
| mode               | text     |  | 该进程所持有或期望的锁模式的名称                                                      |
| granted            | boolean  |  | 锁被持有为真，锁为等待为假                                                         |
| fastpath           | boolean  |  | 如果通过快速路径进行锁定则为True，如果通过主锁定表进行锁定则为false。                               |
| mppsessionid       | integer  |  | 与锁相关的客户端会话的id。                                                        |
| mppiswriter        | boolean  |  | 指明该锁是否由一个写进程所持有。                                                      |
| gp_segment_id      | integer  |  | Greenplum持有该锁的segment的id (dbid) 。                                     |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_partition\_encoding系统目录表描述一个分区模板的可用的列压缩选项。

Table 1. pg\_catalog.pg\_attribute\_encoding

| 列                | 类型       | 修饰符      | 存储       | 描述 |
|------------------|----------|----------|----------|----|
| parencoid        | oid      | not null | plain    |    |
| parencattnum     | smallint | not null | plain    |    |
| parencattoptions | text [ ] |          | extended |    |

**Parent topic:** [系统目录定义](#)



Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

视图pg\_stat\_operations显示了关于最后一个执行在数据库对象（例如，表、索引、视图或者数据库）上或者全局对象（例如角色）的操作的细节信息。

Table 1. pg\_catalog.pg\_stat\_operations

| 列          | 类型          | 参考 | 描述                                                                                                     |
|------------|-------------|----|--------------------------------------------------------------------------------------------------------|
| classname  | text        |    | pg_catalog schema中存储有关此对象的记录的系统表的名称 (pg_class=关系, pg_database=数据库, pg_namespace=schemas, pg_authid=角色) |
| objname    | name        |    | 对象的名称。                                                                                                 |
| objid      | oid         |    | 对象的OID。                                                                                                |
| schemaname | name        |    | 对象所在的schema的名称。                                                                                        |
| usestatus  | text        |    | 对对象执行最后一次操作的角色的状态 (CURRENT = 系统中当前活动的角色, DROPPED = 系统中不再存在的角色, CHANGED = 系统中存在的角色名称, 但自上次操作以来已更改)。     |
| username   | name        |    | 对此对象执行操作的角色的名称。                                                                                        |
| actionname | name        |    | 对该对象采取的操作。                                                                                             |
| subtype    | text        |    | 操作对象的类型或执行的操作的子类。                                                                                      |
| statime    | timestamptz |    | 操作的时间戳。这与写入Greenplum数据库服务器日志文件的时间戳相同, 以防您需要在日志中查找有关操作的更多详细信息。                                          |

Parent topic: [系统目录定义](#)



Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`pg_stat_partition_operations`视图显示了执行在一个分区表上的上一个操作的细节信息。

Table 1. `pg_catalog.pg_stat_partition_operations`

| 列          | 类型          | 参考 | 描述                                                                                                                                    |
|------------|-------------|----|---------------------------------------------------------------------------------------------------------------------------------------|
| classname  | text        |    | <code>pg_catalog</code> schema中系统表的名称，其中存储有关此对象的记录（对于表和分区，始终为 <code>pg_class</code> ）。                                                |
| objname    | name        |    | 对象的名称。                                                                                                                                |
| objid      | oid         |    | 对象的OID。                                                                                                                               |
| schemaname | name        |    | 对象所在的schema的名称。                                                                                                                       |
| usestatus  | text        |    | 对对象执行最后一次操作的角色的状态（ <code>CURRENT</code> =系统中当前活动的角色， <code>DROPPED</code> =系统中不再存在的角色， <code>CHANGED</code> =系统中存在的角色名称，但自上次操作以来已更改）。 |
| username   | name        |    | 对此对象执行操作的角色的名称。                                                                                                                       |
| actionname | name        |    | 对该对象采取的操作。                                                                                                                            |
| subtype    | text        |    | 操作对象的类型或执行的操作的子类。                                                                                                                     |
| statime    | timestamptz |    | 操作时间戳。这与进入Greenplum数据库服务器日志文件                                                                                                         |

|                  |          |  |                                |
|------------------|----------|--|--------------------------------|
|                  |          |  | 的时间戳相同，以防您需要在日志中查找有关操作的更多详细信息。 |
| partitionlevel   | smallint |  | 层次结构中此分区的级别。                   |
| parenttablename  | name     |  | 父表的关系名称从此分区向上一级。               |
| parentschemaname | name     |  | 父表所在的schema的名称。                |
| parent_relid     | oid      |  | 父表的OID从此分区向上一级。                |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

pg\_type\_encoding系统目录表包含了列存储类型信息。

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

**Table 1. pg\_catalog.pg\_type\_encoding**

| 列          | 类型       | 描述符      | 存储       | 描述                               |
|------------|----------|----------|----------|----------------------------------|
| typeid     | oid      | not null | plain    | <a href="#">pg_attribute</a> 的外键 |
| typoptions | text [ ] |          | extended | 实际的选项                            |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

系统表

系统视图

□ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

pg\_user\_mappings视图提供对用户映射信息的访问。此视图本质上是pg\_user\_mappings系统目录表的公共可读视图，如果用户没有查看它的访问权限，则会忽略选项字段。

Table 1. pg\_user\_mappings

| 列         | 类型     | 参考                        | 描述                                 |
|-----------|--------|---------------------------|------------------------------------|
| umid      | oid    | pg_user_mapping.oid       | 用户映射的OID。                          |
| srvrid    | oid    | pg_foreign_server.oid     | 包含此映射的外部服务器的OID。                   |
| srvname   | text   | pg_foreign_server.srvname | 外部服务器的名称。                          |
| umuser    | oid    | pg_authid.oid             | 要映射的本地角色的OID，如果用户映射是public的，则为0。   |
| username  | name   |                           | 要映射的本地用户的名称。                       |
| umoptions | text[] |                           | 特定于用户映射的选项，作为“keyword = value”字符串。 |

要保护存储为用户映射选项的密码信息，umoptions列将显示为null，除非以下应用情况之一：

- 当前用户是被映射的用户，拥有服务器或拥有USAGE权限。
- 当前用户是服务器所有者，映射是PUBLIC。
- 当前用户是超级用户。

**Parent topic:** [系统目录定义](#)





Greenplum数据库® 6.0文档

## □ 参考指南

## □ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

## □ 系统目录参考

系统表

系统视图

## □ 系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

`user_mapping_options`视图包含为当前数据库中的用户映射定义的所有选项。 Greenplum数据库仅显示当前用户可以访问的那些用户映射（通过作为所有者或具有某些权限）。

Table 1. `user_mapping_options`

| 列                                     | 类型                          | 参考 | 描述                                                                                                                                                                          |
|---------------------------------------|-----------------------------|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>authorization_identifier</code> | <code>sql_identifier</code> |    | 要映射的用户的名称，如果映射是公共的，则为 <code>PUBLIC</code> 。                                                                                                                                 |
| <code>foreign_server_catalog</code>   | <code>sql_identifier</code> |    | 定义此映射使用的外部服务器的数据的名称（始终是当前数据库）。                                                                                                                                              |
| <code>foreign_server_name</code>      | <code>sql_identifier</code> |    | 此映射使用的外部服务器的名称。                                                                                                                                                             |
| <code>option_name</code>              | <code>sql_identifier</code> |    | 选项的名称。                                                                                                                                                                      |
| <code>option_value</code>             | <code>character_data</code> |    | 选项的值。此列将显示 <code>null</code> ，除非： <ul style="list-style-type: none"> <li>• 当前用户是被映射的用户。</li> <li>• 映射适用于<code>PUBLIC</code>，当前用户是外部服务器所有者。</li> <li>• 当前用户是超级用户。</li> </ul> |
|                                       |                             |    | 目的是保护存储为用户映射选项的密码信息。                                                                                                                                                        |

**Parent topic:** [系统目录定义](#)

Greenplum数据库® 6.0文档

参考指南

SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

系统目录参考

系统表

系统视图

系统目录定义

foreign\_data\_wrapper\_op

foreign\_data\_wrappers

foreign\_server\_options

foreign\_servers

foreign\_table\_options

foreign\_tables

gp\_configuration\_history

gp\_distributed\_log

gp\_distributed\_xacts

gp\_distribution\_policy

gpexpand.expansion\_pro

gpexpand.status

user\_mappings视图包含当前数据库中定义的所有用户mappgins。Greenplum数据库仅显示当前用户可以访问的那些用户映射（通过作为所有者或具有某些权限）。

Table 1. user\_mappings

| 列                        | 类型             | 参考 | 描述                              |
|--------------------------|----------------|----|---------------------------------|
| authorization_identifier | sql_identifier |    | 要映射的用户的名称，如果映射是公共的，则为PUBLIC。    |
| foreign_server_catalog   | sql_identifier |    | 定义此映射使用的外部服务器的数据库的名称（始终是当前数据库）。 |
| foreign_server_name      | sql_identifier |    | 此映射使用的外部服务器的名称。                 |

Parent topic: [系统目录定义](#)

Greenplum数据库® 6.0文档

□ 参考指南

□ SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

□ 系统目录参考

□ gp\_toolkit管理模式

□ gpperfmon 数据库

□ Greenplum 数据库数据类型

**日期/时间类型**

伪类型

全文搜索类型

范围类型

字符集支持

□ 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum BI / DataV 五

Greenplum 完整支持全部 SQL 日期和时间类型, 参见 [Table 1](#)。对于这些数据类型可用的操作说明, 参见 PostgreSQL 文档 [日期/时间函数和运算符](#)。日期根据罗马日历计算, 在日历引入之前的年也这样计算(参见 PostgreSQL 文档 [日期单位的历史](#) 了解更多信息)。

Table 1. 日期/时间类型

| 名称                                               | 存储大小  | 描述             | 最小值           | 最大值           | 精度            |
|--------------------------------------------------|-------|----------------|---------------|---------------|---------------|
| timestamp [ ( <i>p</i> ) ] [ without time zone ] | 8 字节  | 包括时间和日期(无时区)   | 4713 BC       | 294276 AD     | 1 微秒 / 14 位数字 |
| timestamp [ ( <i>p</i> ) ] with time zone        | 8 字节  | 包括时间和日期(有时区)   | 4713 BC       | 294276 AD     | 1 微秒 / 14 位数字 |
| date                                             | 4 字节  | 日期(无一天内的时间)    | 4713 BC       | 5874897 AD    | 1 天           |
| time [ ( <i>p</i> ) ] [ without time zone ]      | 8 字节  | 一天内的时间(无日期和时区) | 00:00:00      | 24:00:00      | 1 微秒 / 14 位数字 |
| time [ ( <i>p</i> ) ] with time zone             | 12 字节 | 一天内的时间, 无日期, 有 | 00:00:00+1459 | 24:00:00-1459 | 1 微秒 /        |

|                                           |              |      |                   |                                                    |                   |
|-------------------------------------------|--------------|------|-------------------|----------------------------------------------------|-------------------|
|                                           |              | 时区   |                   |                                                    | 14<br>位<br>数<br>字 |
| interval [ <i>fields</i> ] [( <i>p</i> )] | 16<br>字<br>节 | 时间间隔 | -178000000 年<br>年 | 178000000<br>1<br>微<br>秒<br>/<br>14<br>位<br>数<br>字 |                   |

Note: SQL 标准要求 timestamp 与 timestamp without time zone 写法一致, Greenplum 实现了这一点。timestamptz 是 timestamp with time zone 的缩写; 这是 PostgreSQL 的一点扩展。

time, timestamp, 和 interval 接受一个可选的精度值 *p*, 来指定秒的小数部分数字。默认情况下, 没有明确精度指定。对于 timestamp 和 interval 类型, 允许的 *p* 值范围为 0 到 6。

Note: 当 timestamp 值按八字节整数存储(当前默认), 精度可全范围(0 到 6 位)任意设置。当 timestamp 值按双精度浮点数存储(一个过时的编译选项), 精度有效值可能小于 6 位。timestamp 值存储的是从 2000-01-01 午夜开始的秒的数值。当 timestamp 值按浮点数实现时, 2000-01-01 附近的一些年可以达到微秒精度, 但是距离这个日期稍远的一些时间精度就会有所降低。注意: 浮点存储的 timestamp 值允许更大的日期范围, 比上表中给出的要大, 即可以表示从 4713 BC 到 5874897 AD 的日期。

同样的编译选项也决定 time 和 interval 类型值存储为浮点数还是八字节整数。当存储位浮点数的时候, 当时间间隔增大时, interval 的精度有所降低。

对于 time 类型, 使用八字节整数存储时, 允许的 *p* 值范围从 0 到 6; 使用浮点数存储时, 允许的 *p* 值范围从 0 到 10。

另外, interval 数据类型还有一个附加选项, 可以通过书写以下时间单位后缀(*fields*)来限制存储内容:

YEAR  
MONTH  
DAY  
HOUR  
MINUTE

```

SECOND
YEAR TO MONTH
DAY TO HOUR
DAY TO MINUTE
DAY TO SECOND
HOUR TO MINUTE
HOUR TO SECOND
MINUTE TO SECOND

```

注意：如果同时指定 *fields* 与 *p* 参数, *fields* 必须包含 SECOND, 因为精度参数 *p* 只对秒起作用。

`time with time zone` 类型由 SQL 标准定义, 但是定义给出的属性导致人们怀疑它的有用性。大部分情况下, 复合使用 `date`, `time`, `timestamp without time zone`, 以及 `timestamp with time zone` 已经提供了任何应用程序所需的完整日期/时间功能。

`abstime` 和 `reltime` 是内部使用的低精度数据类型。不建议你在应用程序中使用; 这些内部数据类型可能在未来的某一个版本中消失。

## 日期/时间输入

几乎任何合理的日期/时间输入格式都可以被接受。包括 ISO 8601, SQL-兼任格式, 传统 POSTGRES 格式, 等等。对于一些格式, 数据输入中的年、月、日的顺序非常含糊, 所以支持指定这些字段的期望顺序。参见 [DateStyle](#) 通过参数 MDY 指定月-日-年 格式, DMY 指定日-月-年 格式, 或者 YMD 指定年-月-日 格式。

Greenplum 在处理日期/时间输入上比 SQL 标准要求的更加弹性。参见 PostgreSQL 文档 [附录 B. 时间/日期支持](#), 了解时间/日期输入的确切解析规则, 以及如何识别文本字段, 包括月份, 星期和时区。

记住, 任何日期或时间字面值需要用单引号引起, 就像字符串一样。SQL 需要这样的语法

```
type [(p)] ' value '
```

这里, *p* 是一个可选精度参数, 指定秒的小数位数。它对以下类型起作用, `time`, `timestamp`, 以及 `interval` 类型。允许设定的值范围已经在上面文档中说明; 如果没有指定精度, 默认为字面值的精度。

## 日期

[Table 2](#) 一些可能的 `date` 类型输入格式.

Table 2. 日期输入

| 举例               | 说明                                                                     |
|------------------|------------------------------------------------------------------------|
| 1999-01-08       | ISO 8601; 1 月 8 日 (推荐格式)                                               |
| January 8, 1999  | 对任何日期风格都很明确的输入模式                                                       |
| 1/8/1999         | MDY 模式: 1 月 8 日 ; DMY 模式: 8 月 1 日                                      |
| 1/18/1999        | MDY 模式: 1 月 18 日; 其他模式: 无效                                             |
| 01/02/03         | MDY 模式: 2003 年 1 月 2 日; DMY 模式: 2003 年 2 月 1 日; YMD 模式: 2001 年 2 月 3 日 |
| 1999-Jan-08      | 任何模式: 1999 年 1 月 8 日                                                   |
| Jan-08-1999      | 任何模式: 1999 年 1 月 8 日                                                   |
| 08-Jan-1999      | 任何模式: 1999 年 1 月 8 日                                                   |
| 99-Jan-08        | YMD 模式: 1999 年 1 月 8 日, 其他: 错误                                         |
| 08-Jan-99        | YMD 模式: 错误; 其他模式: 1999 年 1 月 8 日                                       |
| Jan-08-99        | YMD 模式: 错误; 其他模式: 1999 年 1 月 8 日                                       |
| 19990108         | ISO 8601; 任何模式: 1999 年 1 月 8 日                                         |
| 990108           | ISO 8601; 任何模式: 1999 年 1 月 8 日                                         |
| 1999.008         | 年和一年中的天                                                                |
| J2451187         | 儒略日期 (天文学常用)                                                           |
| January 8, 99 BC | 公元前 99 年 1 月 8 日                                                       |

## 时间

时间类型包括 `time [ ( p ) ] without time zone` 和 `time [ ( p ) ] with time zone`。`time` 与 `time without time zone` 等价。

这些类型的有效的输入格式由时间加上一个可选的时区。(参见 [Table 3](#) 和 [Table 4](#).) 如果在输入中给 `time without time zone` 类型指定了

,

时区 则时区会被忽略。你如果指定了一个日期，也会被忽略，除非你用了包含夏令时的时区，例如 America/New\_York. 这种情况下，指定日期是必要的，因为需要决定当前是否是标准时间，还是夏令时时间. 使用 time with time zone 类型时，合适的时区偏移会被记录.

Table 3. 时间输入

| 举例                                      | 说明                      |
|-----------------------------------------|-------------------------|
| 04:05:06.789                            | ISO 8601                |
| 04:05:06                                | ISO 8601                |
| 04:05                                   | ISO 8601                |
| 040506                                  | ISO 8601                |
| 04:05 AM                                | 与 04:05 相同; AM 不影响值     |
| 04:05 PM                                | 与 16:05 相同; 小时值必须 <= 12 |
| 04:05:06.789-8                          | ISO 8601                |
| 04:05:06-08:00                          | ISO 8601                |
| 04:05-08:00                             | ISO 8601                |
| 040506-08                               | ISO 8601                |
| 04:05:06 PST                            | 时区用缩写指定                 |
| 2003-04-12 04:05:06<br>America/New_York | 时区用全称指定                 |

Table 4. 时区输入

| 举例               | 说明               |
|------------------|------------------|
| PST              | 缩写 (太平洋标准时间)     |
| America/New_York | 时区全称             |
| PST8PDT          | POSIX-样式时区格式     |
| -8:00            | ISO-8601 偏移(PST) |
| -800             | ISO-8601 偏移(PST) |
| -8               | ISO-8601 偏移(PST) |
| zulu             | 军方缩写(UTC)        |
| z                | zulu 的短格式        |

参考 [时区](#) 了解更多时区格式输入信息.

# 时间戳

时间戳的有效输入格式由以下几个部分组成：日期，时间，时区(可选)，接着可选的 AD 或 BC。(另外，AD / BC 也可以出现在时区之前，但不是推荐的顺序。) 因而：1999-01-08 04:05:06 和：1999-01-08 04:05:06 -8:00 都是有效的时间戳值，它们满足 ISO 8601 标准要求。另外，常用格式：January 8 04:05:06 1999 PST 也被支持。

SQL 标准中 `timestamp without time zone` 与 `timestamp with time zone` 字面值的差异主要体现在时间后面由一个 + 或 - 号标识的时区。因此，根据标准，`TIMESTAMP '2004-10-19 10:23:54'` 是一个 `timestamp without time zone` 类型字面值，而 `TIMESTAMP '2004-10-19 10:23:54+02'` 是一个 `timestamp with time zone` 类型的字面值。Greenplum 在确定类型之前从不检查字面值内容，所以上面两个字面值都会被认为是 `timestamp without time zone` 类型。为确保当作 `timestamp with time zone` 类型对待，请像这样给出明确类型：`TIMESTAMP WITH TIME ZONE '2004-10-19 10:23:54+02'` 对于一个已经确定为 `timestamp without time zone` 类型的字面值，Greenplum 会抛弃时区相关信息。也就是说，结果时间戳仅仅根据日期/时间确定，不会根据时区进行调整。

对于 `timestamp with time zone` 类型，内部值总是按照 UTC (统一协调时间，通常叫做格林威治时间，GMT) 存储。带有明确时区的输入值会使用合适的时区偏移转为 UTC 时间。如果输入字符串中没有指定时区，会假定为当前系统的 `TimeZone` 参数，并使用这个参数的偏移转换为 UTC 时间。

当输出一个 `timestamp with time zone` 类型值，总是根据当前 `timezone` 时区值进行转换，并显示为本地时间。要想以另一个时区查看时间，要么改变 `timezone` 设置，要么使用 `AT TIME ZONE` 构造 (参见 PostgreSQL 文档的 [AT TIME ZONE](#))。

在 `timestamp without time zone` 与 `timestamp with time zone` 间转换通常假定 `timestamp without time zone` 中的值应该是以 `timezone` 为时区的本地时间。当然，也可以用 `AT TIME ZONE` 指定一个不同的时区。

# 特殊值

为了方便, Greenplum 支持几种特殊的日期/时间输入值格式, 参见 [Table 5](#). `infinity` 和 `-infinity` 值是用于系统内部的特殊表示, 会被直接显示; 但其他都只是些快捷记号, 会在读取的时候被转换为常规的日期/时间值.(特别地, `now` 和相关字符串会被转换为读取时的当前特定时间值.) 所有这些常量值在 SQL 命令中都需要用单引号引起.

**Table 5. 特殊日期/时间输入**

| 输入字符串                  | 有效类型                                                                 | 说明                                   |
|------------------------|----------------------------------------------------------------------|--------------------------------------|
| <code>epoch</code>     | <code>date</code> ,<br><code>timestamp</code>                        | 1970-01-01 00:00:00+00 (Unix 系统时间零点) |
| <code>infinity</code>  | <code>date</code> ,<br><code>timestamp</code>                        | 比任何时间戳都晚的时间                          |
| <code>-infinity</code> | <code>date</code> ,<br><code>timestamp</code>                        | 比任何时间戳都早的时间                          |
| <code>now</code>       | <code>date</code> ,<br><code>time</code> ,<br><code>timestamp</code> | 当前事务的开始时间                            |
| <code>today</code>     | <code>date</code> ,<br><code>timestamp</code>                        | 今天午夜                                 |
| <code>tomorrow</code>  | <code>date</code> ,<br><code>timestamp</code>                        | 明天午夜                                 |
| <code>yesterday</code> | <code>date</code> ,<br><code>timestamp</code>                        | 昨天午夜                                 |
| <code>allballs</code>  | <code>time</code>                                                    | 00:00:00.00 UTC                      |

下面这些 SQL-兼容函数也可以用来获取相应类型的当前时间:  
`CURRENT_DATE`, `CURRENT_TIME`, `CURRENT_TIMESTAMP`,  
`LOCALTIME`, `LOCALTIMESTAMP`. 后面四个可接受一个可选的亚秒级精度参数. (参见 PostgreSQL 文档的[当前日期/时间](#).) 注意有一些 SQL 函数在输入字符串中不被接受.

## 日期/时间输出

日期/时间类型的输出格式可以被设置为以下四种样式: ISO 8601, SQL (Ingres), 传统 POSTGRES (Unix `date` 格式), 或者 German. 默认是 ISO 格式. (SQL 标准需要使用 ISO 8601 格式. SQL 输出格式这个名字是由于偶然的历史原因.) [Table 6](#) 给出了各种输出格式的例子. `date` 和 `time` 类型的输出格式通常为对应例子中的日期或时间部分. 然而, POSTGRES 格式的纯日期输出用的是 ISO 格式.

Table 6. 日期/时间输出格式

| 样式规范     | 说明               | 举例                              |
|----------|------------------|---------------------------------|
| ISO      | ISO 8601, SQL 标准 | 1997-12-17 07:37:16-08          |
| SQL      | 传统样式             | 12/17/1997 07:37:16.00<br>PST   |
| Postgres | 原始样式             | Wed Dec 17 07:37:16 1997<br>PST |
| German   | 区域样式             | 17.12.1997 07:37:16.00<br>PST   |

Note: ISO 8601 规定使用大写字母 T 隔开日期和时间. Greenplum 支持这个输入格式,但是输出时使用空格而不是 T,上面您已经看到了.这是为了可读性, 同时与 RFC 3339 以及一些其他数据库系统保持一致.

在 SQL 和 POSTGRES 样式中, 如果指定 DMY 模式, 日会出现在月之前; 其他情况下月在日之前。(参见 [Table 2](#) 关于这些设置如何影响输入值解释.) [Table 7](#) 中的例子.

Table 7. 日期顺序约定

| <b>datestyle</b> 设置 | 输入顺序                      | 输出举例                            |
|---------------------|---------------------------|---------------------------------|
| SQL, DMY            | <i>day / month / year</i> | 17/12/1997 15:37:16.00<br>CET   |
| SQL, MDY            | <i>month / day / year</i> | 12/17/1997 07:37:16.00<br>PST   |
| Postgres,<br>DMY    | <i>day / month / year</i> | Wed 17 Dec 07:37:16<br>1997 PST |

用户可以使用以下方式选择日期/时间样式: `SET datestyle` 命令,  
`postgresql.conf` 配置文件中的 `DateStyle` 参数, 或者服务器或客  
户端的 `PGDATESTYLE` 环境变量.

格式化函数 `to_char` (参见 [数据类型格式化函数](#)) 是一种格式化日期/时间输出的更加灵活的方式.

时

时区和时区转换，不仅是地理位置问题，也受政治决定影响。从 1900

年以来，世界上的时区稍微标准了一点 但还是在持续的改变 尤其是夏令时规则. Greenplum 使用广泛采纳的 IANA (Olson) 时区数据库来定义历史时区规则. 对于未来时间, 假定给定时区的现行已知规则会无限期持续.

Greenplum 保持与 SQL 标准兼容的典型用法定义. 然而, SQL 标准偶尔也有一点混淆时间和日期类型和能力. 两个明显的问题是:

1. 虽然 `date` 类型不能带时区, 但是 `time` 类型是可以的. 不与日期和时间关联, 时区在真实世界意义不大, 因为偏移变化与夏令时边界相关.
2. 默认时区是与 UTC 之间差异的常量数值. 因而, 当进行跨 DST 边界的日期/时间数值计算时, 可能采用夏令时.

为了解决这些困难, 当使用时区的时候, 我们推荐用同时包含日期和时间的日期/时间类型. 我们不推荐使用 `time with time zone` 类型(虽然 Greenplum 支持, 但主要是为了一个老应用, 同时保持与 SQL 标准兼容). 对于任何只包含日期或时间的类型, Greenplum 假定是你的本地时区(的时间或日期).

所有时区感知的日期和时间内部都存储为 UTC 值. 当被显示到客户端时, 才通过 `TimeZone` 配置参数的值转换为指定时区的日期或时间.

Greenplum 允许你以三种不同的形式指定期区:

1. 时区全称, 例如 `America/New_York`. 在 `pg_timezone_names` 视图里列出了所有可被识别的时区名称. 为了这个目的, Greenplum 使用广泛采纳的 IANA 的时区数据, 因此同样的时区名称也可以被很多其他软件识别.
2. 另一个形式时区缩写, 例如 `PST`. 对于时区全称, 能够暗示夏令时规则, 而时区缩写这个规范只能表示与 UTC 直接的时区偏移. 可识别的缩写列在 `pg_timezone_abbrevs` 视图中. 你不能给时区缩写设置配置参数 `TimeZone` 或 `log_timezone`, 但你可以在时间/日期输入值中通过 `AT TIME ZONE` 操作符使用缩写.
3. 除了时区全称和时区缩写外, Greenplum 也接受 POSIX-风格的时区规范, 形如 `STD offset` 或者 `STD offset DST`, 这里 `STD` 是一个时区缩写, `offset` 是 UTC 往西偏移的小时数值, `DST` 是一个可选的夏令时时区缩写, 假定为比给定偏移提前一个小时. 举个例子, 如果 `EST5EDT` 还没有被识别为时区名称, 它能被接受, 相当于美国东岸时间(United States East Coast time). 在这种语法中, 时区缩写可以是一串字母, 或者包围在尖括号中的任意字符串(`<>`). 当存在夏令时时区缩写时, 假定使用与 IANA 时区数据库中相同的夏令时转换规则. 在标准 Greenplum 安装时, 与 `US/Eastern` 相同, 因此 POSIX-风格时区规范遵守 USA 夏令时规则. 如果需要, 你也可以通过替换文件来调整这个行为.

总之, 这就是时区全称与缩写的不同: 缩写表示与 UTC 之间的偏移, 而全称还暗示了夏令时规则, 因此有两种可能的 UTC 偏移. 例如, 2014-06-04 12:00 America/New\_York 表示纽约本地时间中午, 指定为东部夏令时间 Eastern Daylight Time (UTC-4). 因此 2014-06-04 12:00 EDT 指定了同样的时间常数. 但是 2014-06-04 12:00 EST 指定为东部标准时间 (UTC-5) 中午, 不管那天夏令时是否正常生效与否.

另外复杂情况是, 有些管辖范围使用同样的时区缩写表示不同的 UTC 偏移和不同的时间; 例如, 在 MSK 有些年头表示 UTC+3, 有些年头又表示 UTC+4. 对于这些缩写, Greenplum 根据日期来进行解释(对于未来时间, 根据最近的情况解释); 但是, 就像上面 EST 这个例子, 这不一定与那个日期的本地时间相同.

应该小心 POSIX-风格时区特征很容易导致错误输入被默默接受, 因为很难检查时区输入的合理性. 例如, SET TIMEZONE TO FOOBAR0 能够正常工作, 让系统有效使用相当特别的 UTC 缩写. 另一个要记住的问题是在 POSIX 时区名称中, 正数被用在(相对于)格林威治时间. 其他任何地方, Greenplum 遵守 ISO-8601 约定, 正时区偏移都是(相对于)格林威治时间.

所有情况下, 失去名称和缩小都是大小写无关的.

不论时区名称还是缩写, 都没有硬编码进服务程序; 我们是从配置文件 .TimeZone 配置参数可以在文件中设定, 或者其他任何标准设置配置参数的方式. 另外也有一个特殊方式来设置:

1. SQL 命令 SET TIME ZONE 可以设置当前会话的时区. 这是比 SET TIMEZONE TO 更符合 SQL 兼容规范的一种形式.
2. PGTZ 环境变量被 libpq 客户端用来, 在初始化与服务器的连接时, 发送 SET TIME ZONE 命令.

## 时间段输入

时间段值可以被写成下面的详细语法:

```
@ quantity unit quantity unit... direction
```

这里 *quantity* 是一个有符号数值; *unit* 是 microsecond, millisecond, second, minute, hour, day, week, month, year, decade, century, millennium, 或者这些单位的缩写或复数形式; *direction* 可以是 ago 或者为空. at 符号 (@) 是一个可选的无用符号. 不同单位的值会被隐式的加在一起(考虑符号和单位). ago 导致所有单

位的值都变号. 如果 `IntervalStyle` 设置为了 `postgres_verbose`, 这个语法也用于时间段输出.

天, 小时, 分, 和秒的数量都可以不用单位指定. 例如, '1 12:59:10' 与这个 '1 day 12 hours 59 min 10 sec' 是一样的. 另外, 年和月可以通过一个划线(减号)来指定; 例如 '200-10' 读出来与 '200 years 10 months' 一样. (这些短格式事实上是 SQL 标准仅允许的格式. 当 `IntervalStyle` 设置给 `sql_standard` 时, 也按这样的格式输出.)

时间段值也可以写成 ISO 8601 时间段的格式, 用标准中的 4.4.3.2 节内容 `format with designators`, 或者用 4.4.3.3 节说明的 `alternative format`. `format with designators` 看起来像这个样子:

```
P quantity unit quantity unit... T quantity unit...
```

这个时间段字符串必须以 `P` 开头, 可以包含一个 `T` 来隔开时间部分. 可以使用的缩写列在 [Table 8](#) 中. 单位可以忽略, 也可以按任意顺序指定, 但是时间单位必须出现在 `T` 后面. 特别地, `M` 的含义依赖于它出现在 `T` 之前还是之后.

**Table 8. ISO 8601 时间段单位缩写**

| 缩写 | 含义        |
|----|-----------|
| Y  | 年         |
| M  | 月 (在日期部分) |
| W  | 周         |
| D  | 天         |
| H  | 小时        |
| M  | 分 (在时间部分) |
| S  | 秒         |

另一种格式:

```
P years - months - days T hours : minutes : seconds
```

字符串必须以 `P` 开头, 用字母 `T` 隔开时间段的日期和时间部分. 数值按类似于 ISO 8601 日期格式书写.

当书写一个带有 `fields` 格式的时间段常量, 或者当把一个字符串赋给带有 `fields` 格式的时间段列时, 没有标志单位的数值根据 `fields` 解释. 例如 `INTERVAL '1' YEAR` 读成 1 年, 而 `INTERVAL '1'` 意味着 1 秒. 另外, 带有 `to the right` 字段值中 `fields` 允许的右边最不重要的字段会被简单的忽略. 例如, 写 `INTERVAL '1 day 2:03:04' HOUR TO`

MINUTE 结果或丢弃秒部分, 而不会丢掉天的部分.

根据 SQL 标准, 时间段中所有部分必须有同样的符号, 开头的负号会对所有部分起作用; 例如, 在时间段字面值 '-1 2:03:04' 中, 负号应用于天, 以及小时/分/秒部分. Greenplum 允许各部分有不同的符号, 并且传统认为文本中每个部分有独立的符号, 因此上面例子中小时/分/秒会被认为是正值. 如果 *IntervalStyle* 设置给 `sql_standard` 那么开头的负号就应用到所有部分 (只有当别的地方没有符号时). 要不然, 就用传统 Greenplum 解释. 为了避免模糊, 如果时间段值是负的, 建议每个部分都带上负号.

在详细日期格式中, 对于更紧凑的输入格式的几个部分值可以带小数; 例如 '1.5 week' 或者 '01:02:03.45'. 这些输入被转换为月, 天和秒来存储. 当这些导致结果的月或者天有小数时, 小数部分被加到低一级的部分上. 转换规则是 1 月 = 30 天, 1 天 = 24 小时. 例如, '1.5 month' 变成 1 月 15 天. 输出中秒从来不会显示小数部分.

**Table 9** 这里给出 时间段 输入的有效示例.

**Table 9. 时间段输入**

| 举例                                                 | 说明                                       |
|----------------------------------------------------|------------------------------------------|
| 1-2                                                | SQL 标准格式: 1 年 2 个月                       |
| 3 4:05:06                                          | SQL 标准格式: 3 天 4 小时 5 分 6 秒               |
| 1 year 2 months 3 days 4 hours 5 minutes 6 seconds | 传统 Postgres 格式: 1 年 2 月 3 天 4 小时 5 分 6 秒 |
| P1Y2M3DT4H5M6S                                     | ISO 8601 format with designators: 同上     |
| P0001-02-03T04:05:06                               | ISO 8601 alternative format: 同上          |

内部地 时间段 值被存为月, 天, 和秒. 这是因为一个月中的天数是变化的, 另外因为夏令时的原因, 一天可能有23, 24 或 25 小时. 月和天部分是整数, 秒可以包含小数. 因为时间段通常从常量字符串或 时间戳 相减得来, 大部分情况下这种存储挺好, 但有时也有一些意外: `SELECT EXTRACT(hours from '80 minutes'::interval);` `date_part ----- 1` `SELECT EXTRACT(days from '80 hours'::interval);` `date_part ----- 0` 当超过值的正常范围时, 可以用 `justify_days` 函数和 `justify_hours` 函数用来调整天和小时.

# 时间段输出

时间段类型的输出格式可以设置为以下四种格式之一:  
`sql_standard`, `postgres`, `postgres_verbose`, 或者 `iso_8601`,  
通过命令 `SET intervalstyle` 来设置. 默认是 `postgres` 格式.  
[Table 10](#) 里有每种输出格式的一些例子.

如果时间段值符合标准要求 (年-月 或者 天-时间, 没有混合正负数),  
`sql_standard` 样式产生的输出遵从 SQL 标准中对于时间段字面值  
的规范. 要不然输出看起来像一个标准的 年-月 字面值, 后面跟着一个  
天-时间 字面值字符串, 并且标记上明确的符号来消除混合符号带来的  
歧义.

当 `DateStyle` 参数设置为 `ISO` 时, `postgres` 样式输出与 PostgreSQL  
8.4 发布之前的输出一致.

当 `DateStyle` 参数设置为 非- `ISO` 时, `postgres_verbose` 样式的输出  
与 PostgreSQL 8.4 之前版本的输出一致.

`iso_8601` 样式与 ISO 8601 标准中 4.4.3.2 节关于 `format with  
designators` 的描述一致.

Table 10. 时间段输出样式举例

| 样式规范                          | 年-月<br>时间段      | 天-时间<br>时间段                    | 混合时间段                                                |
|-------------------------------|-----------------|--------------------------------|------------------------------------------------------|
| <code>sql_standard</code>     | 1-2             | 3 4:05:06                      | -1-2 +3 -4:05:06                                     |
| <code>postgres</code>         | 1 year 2 mons   | 3 days 04:05:06                | -1 year -2 mons<br>+3 days<br>-04:05:06              |
| <code>postgres_verbose</code> | @ 1 year 2 mons | @ 3 days 4 hours 5 mins 6 secs | @ 1 year 2 mons<br>-3 days 4 hours 5 mins 6 secs ago |
| <code>iso_8601</code>         | P1Y2M           | P3DT4H5M6S                     | P-1Y-2M3DT-4H-5M-6S                                  |

Parent topic: [Greenplum 数据库数据类型](#)

Greenplum数据库® 6.0文档

- 参考指南

- SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

- 系统目录参考

- gp\_toolkit管理模式

- gpperfmon 数据库

- Greenplum 数据库数据类型

日期/时间类型

## 伪类型

全文搜索类型

范围类型

字符集支持

- 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum 数据库支持一个特殊目的的数据类型集合，叫做 伪类型。一个伪类型不能被用作列数据类型，但是可以用来声明一个函数的参数或返回值。当一个函数的行为并不是简单获取或者返回一个特定 SQL 数据类型值的时候，这些伪类型非常有用。

过程语言实现的函数只能使用实现语言中支持的伪类型。过程与语言全都禁止使用伪类型作为参数类型，仅仅允许 *void* 和 *record* 作为返回类型。

一个使用 *record* 伪类型作为返回数据类型的函数返回一个待定的数据类型。*record* 表示可能的匿名复合类型数组。由于复合数据带有自己的类型标识，因此不需要额外的数组级信息。

伪类型 *void* 表示一个函数不返回值。

**Note:** Greenplum 数据库不支持触发器和 *trigger* 伪类型。

另外，*anyelement*, *anyarray*, *anynonnullarray*, 和 *anyenum* 几个伪类型被称为多态类型。一些过程语言也支持多态函数使用 *anyarray*, *anyelement*, *anyenum*, 和 *anynonnullarray* 函数。

*anytable* 伪类型是一个 Greenplum 数据库类型，表示一个二维表表达式 — 评估为一张二维表的表达式。Greenplum 数据库只允许这个类型作为用户定义函数的参数。参见 [二维表表达式](#) 了解更多关于 *anytable* 伪类型信息。

更多关于伪类型的介绍，查看 PostgreSQL 文档中关于 [伪数据类型](#)。

**Parent topic:** [Greenplum 数据库数据类型](#)

## 多态类型

四种特殊目的的数据类型：*anyelement*, *anyarray*, *anynonnullarray*, 和 *anyenum*, 被称作 多态 类型。任何使用这些类型的函数被称做多态函数。通过在运行时检查实际传递的数据，一个多态函数可以操作许多不同的数据类型。

多态参数和返回值互相关联，当查询调用一个多态函数时，真实类型才被决定。每个位置（或者参数或者返回值）声明为 *anyelement* 类型时，允许有任何特定的实际类型，但是，任何调用中，必须是同一种真实数据类型。每个位置声明为 *anyarray* 允许有任何类型的数组，但 ( , ) .

同样的, 在一个特定查询中 必须是同一种数据类型 如果有位置声明为 *anyarray* 同时别的位置声明为 *anyelement*, 在 *anyarray* 位置中的实际数组类型的元素必须与 *anyelement* 位置 中的类型相同. *anynonnullarray* 与 *anyelement* 类型同样对待, 但是包含附加的约束就是, 实际类型不能是数组类型. *anyenum* 被当作 *anyelement* 同样对待, 但是实际类型必须是 *enum* 类型.

当超过一个参数位置声明为多态类型, 最终效果是只允许仅有特定的实际参数组合. 举例说, 函数声明为 `equal( anyelement, anyelement )` 则这个函数接受两个输入值, 并且他们是同样的数据类型就可以.

当一个函数的返回值声明为多态类型时, 必须有一个参数位置也声明为多态类型, 并且实际参数的数据类型决定调用的实际返回值类型. 举例说, 如果还没有数组订阅机制, 你可以像这样定义一个函数实现订阅: `subscript( anyarray, integer ) returns anyelement`. 这里声明约束第一个参数必须是数组类型, 并且允许解析器根据第一个实际参数类型, 推断新的正确的返回值类型. 另一个例子是一个函数声明为 `myfunc( anyarray ) returns anyenum` 则只能接受一个 *enum* 类型数组(作为参数).

注意: *anynonnullarray* 和 *anyenum* 不表示单个类型变量; 可以用 *anyelement*, 只是带一点附加约束. 例如, 声明函数为 `myfunc( anyelement, anyenum )` 与声明为 `myfunc( anyenum, anyenum )` 是一样的: 两者的实际参数都必须是同样的 *enum* 类型.

一个可变参数函数(接收可变数目参数)中, 如果最后的参数声明为 VARIADIC *anyarray*, 就是多态的. 为了匹配和决定实际参数类型, 这样的函数与你生命合适数量的 *anynonnullarray* 参数的函数一样.

了解更多多态类型信息, 参见 PostgreSQL 文档关于 [多态参数和返回值](#).

## 二维表表达式

当用 *anytable* 伪类型声明了一个函数参数, 这就是一个二维表表达式. 一个二维表表达式的写法是把 SELECT 语句放在一个 TABLE( ) 函数中. 你可以通过添加这样的语句来指定表的分布规则: SCATTER RANDOMLY, 或者 SCATTER BY 子句以及列列表指定分布键.

SELECT 语句在函数被调用时执行, 结果集被分布到 segments 节点中, 因此每一个 segment 节点在一部分结果表上执行函数.

举例说,下面这个表表达式从一个叫做 `customer` 的表中选了三列,并将分布键设置为第一列:

```
TABLE(SELECT cust_key, name, address FROM customer SCATTER BY 1)
```

`SELECT` 语句可以包括连接多个基础表, `WHERE` 语句, 聚集, 以及任何有效的查询语法.

对于 C 或 C++ 语言实现的函数, `anytable` 类型是唯一允许的(多态)类型. 函数体可以用 Greenplum 数据库服务器编程接口 (SPI) 或者 Greenplum 伙伴连接器接口 (GPPC) API 访问.

Greenplum数据库® 6.0文档

- 参考指南

- SQL Command Reference

SQL 2008可选特性兼容性

Greenplum环境变量

保留标识符和SQL关键字

- 系统目录参考

- gp\_toolkit管理模式

- gpperfmon 数据库

- Greenplum 数据库数据类型

日期/时间类型

伪类型

## 全文搜索类型

范围类型

字符集支持

- 服务器配置参数

内置函数摘要

Greenplum MapReduce规范

Greenplum PL/pgSQL过程语言

Greenplum PL/R 语言扩展

Greenplum 数据库提供两种数据类型，这两种数据类型被设计来支持全文搜索，即搜索自然语言 文档 集合并定位与一个 查询 最匹配内容。 `tsvector` 类型以一种专为全文搜索优化的形式表示一个文档；类似地， `tsquery` 类型表示一个文本查询。 [使用全文搜索](#) 有这方面支持的详细解释，[文本搜索函数和操作符](#) 总结了相关函数和操作。

`tsvector` 和 `tsquery` 类型不能是 Greenplum 数据库表的分布键的一部分。

**Parent topic:** [Greenplum 数据库数据类型](#)

## tsvector

一个 `tsvector` 值是不同 词素(*lexemes*) 的排序列表，也就是由对不同形式的词进行合并，已经被 标准化的单词组成的列表 (更多细节，请参考 [使用全文搜索](#))。 排序和去重在输入过程中已经自动完成。请看以下例子：

```
SELECT 'a fat cat sat on a mat and ate a fat
rat'::tsvector;
 tsvector

'a' 'and' 'ate' 'cat' 'fat' 'mat' 'on' 'rat' 'sat'
```

要表示包含空格和标点符号的词素，可以用单引号把它们包围起来：

```
SELECT $$the lexeme ' ' contains spaces$$::tsvector;
 tsvector

' ' 'contains' 'lexeme' 'spaces' 'the'
```

(在这个例子中，我们使用了 `$$` 引起来的字符串字面值。下一个例子中，我们使用常规的单引号引起来的字符串，所以字面值中嵌入的单引号必须双写进行转义。) 单引号引起来的词素中嵌入的单引号(')和反斜杠(\)必须双写：

```
SELECT $$the lexeme 'Joe''s' contains a quote$$::tsvector;
 tsvector

'Joe''s' 'a' 'contains' 'lexeme' 'quote' 'the'
```

可选的, 整数 定位数值 可以被附加到词素:

```
SELECT 'a:1 fat:2 cat:3 sat:4 on:5 a:6 mat:7 and:8 ate:9
a:10 fat:11 rat:12'::tsvector;
 tsvector
```

```

'a':1,6,10 'and':8 'ate':9 'cat':3 'fat':2,11 'mat':7
'on':5 'rat':12 'sat':4
```

一个定位数值通常表示源单词在文档中的位置。定位信息能够在邻近排名时使用。定位数值的范围可以从 1 到 16383; 更大的数值自动归整到 16383。相同定位数值的同一词素会被丢弃。

包含定位数值的词素可以进一步使用一个 权重 进行标记, 权重可以是 A, B, C, 或 D 。 D 是默认权重因此在输出中不会显示:

```
SELECT 'a:1A fat:2B,4C cat:5D'::tsvector;
 tsvector
```

```

'a':1A 'cat':5 'fat':2B,4C
```

典型地, 权重被用来反映文档结构, 例如, 可以使用不同权重标记标题与正文中的单词。全文搜索排名函数能够对不同权重标记指派不同优先级。

理解 `tsvector` 类型本身并不进行任何标准化非常重要; 它假定给它的单词已经为应用进行了适当地标准化。例如,

```
select 'The Fat Rats'::tsvector;
 tsvector
```

```

'Fat' 'Rats' 'The'
```

对大多数英文全文搜索应用, 上面的单词被认为是非标准化的, 但是 `tsvector` 并不在意。原始文档文本应该总是通过 `to_tsvector` 函数来为搜索进行适当地标准化:

```
SELECT to_tsvector('english', 'The Fat Rats');
 to_tsvector
```

```

'fat':2 'rat':3
```

## tsquery

一个 tsquery 值为搜索存储词素，同时结合它们实现布尔操作 & (与), | (或), 以及 ! (非)。括号可以用来表示分组操作:

```
SELECT 'fat & rat'::tsquery;
 tsquery

'fat' & 'rat'

SELECT 'fat & (rat | cat)'::tsquery;
 tsquery

'fat' & ('rat' | 'cat')

SELECT 'fat & rat & ! cat'::tsquery;
 tsquery

'fat' & 'rat' & !'cat'
```

不使用括号时, ! (非) 具有最高优先级, 另外 & (与) 优先级高于 | (或)。

可选地, 一个 tsquery 中的词素可以带一个或多个权重标记字母, 用来限制他们只与合适权重的 tsvector 相匹配:

```
SELECT 'fat:ab & cat'::tsquery;
 tsquery

'fat':AB & 'cat'
```

Also, 一个 tsquery 中的词素也可以用星号(\*)指定前缀匹配:

```
SELECT 'super:*'::tsquery;
 tsquery

'super':*
```

这个查询会匹配一个以 "super" 开头的 tsvector 中的任何单词。注意前缀会被全文搜索配置首先处理, 也就是说这个比较会返回 true (真):

```
SELECT to_tsvector('postgraduate') @@ to_tsquery(
'postgres:*');
?column?

t
(1 row)
```

因为 postgres 中提取的词干是 postgr :

```
SELECT to_tsquery('postgres:*');
 to_tsquery

'postgr':*
(1 row)
```

所以就匹配到了 postgraduate 。

词素引用规则与前面 tsvector 中的规则一样; 同时, 也和 tsvector 一样, 任何需要标准化的单词必须在转换为 tsquery 类型之前完成。 to\_tsquery 函数可以方便地用来进行这类标准化操作:

```
SELECT to_tsquery('Fat:ab & Cats');
 to_tsquery

'fat':AB & 'cat'
```

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

- PXF介绍
- PXF管理手册
- 使用PXF访问hadoop
- 使用PXF访问Azure, Google云端存储, Minio和S3对象存储

#### About Accessing the S3 Object Store

读取和写入文本数据

读取Avro数据

读取JSON数据

读写Parquet数据

读写SequenceFile数据

将多行文本文件读入单个表行

使用S3 Select从S3读取CSV和Parquet数据

使用PXF访问SQL数据库(JDBC)

PXF故障排除

- PXF实用程序手册

Back to the Administrator Guide

#### In this topic:

- [用DDL覆盖S3服务器配置](#)
- [使用Amazon S3 Select服务](#)

PXF安装了与S3对象库的连接器。PXF通过此连接器支持以下其他运行时特性：

- 通过在 `CREATE EXTERNAL TABLE` 命令DDL中提供它们来覆盖服务器配置中指定的S3凭据。
- 使用Amazon S3 Select服务从S3读取某些CSV和Parquet数据。

## 用DDL覆盖S3服务器配置

如果您访问兼容S3的对象库，可以通过 `CREATE EXTERNAL TABLE` `LOCATION` 子句中的以下自定义选项直接指定S3访问ID和密钥，从而覆盖S3服务配置：

| 自定义选项                  | 值描述             |
|------------------------|-----------------|
| <code>accesskey</code> | AWS账户访问密钥ID     |
| <code>secretkey</code> | 与AWS访问密钥ID关联的密钥 |

例如：

```
CREATE EXTERNAL TABLE pxf_ext_tbl(name text, orders int)
 LOCATION ('pxf://S3_BUCKET/dir/file.txt?
PROFILE=s3:text&SERVER=s3srvcfg&accesskey=YOURKEY&secretkey=YOURSECRETKEY');
FORMAT 'TEXT' (delimiter=E',');
```

已这种方式提供的凭据在外部表定义中可见。不要在生产环境中使用这种传递凭据的方法。

PXF目前不支持以这种方式覆盖Azure, Google Cloud Storage和Minio服务器凭据。

有关PXF用于获取Greenplum数据库用户的配置属性设置的优先级规则的详细信息，请参考[配置属性优先级](#)。

# 使用Amazon S3 Select服务

请参阅[使用S3 Select从S3读取CSV和Parquet数据](#)，了解有关PXF如何使用Amazon S3 Select服务读取S3上存储的CSV和Parquet文件的特定信息。

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

□ PXF介绍

□ PXF管理手册

□ 使用PXF访问hadoop

读写文本数据

读取Avro数据

读取JSON数据

读写Parquet数据

读写SequenceFile数据

将多行文本文件读入单个表行

读取Hive表数据

读取HBase表数据

□ 使用PXF访问Azure, Google云端存储, Minio和S3对象存储

使用PXF访问SQL数据库(JDBC)

PXF故障排除

□ PXF实用程序手册

Back to the Administrator Guide

Greenplum database 5管理员指南

## In this topic:

- [前提条件](#)
- [读取多行文本和JSON文件](#)
  - [示例：将HDFS文本文件读入单个表行](#)

您可以使用PXF HDFS连接器以单个表行的形式读取HDFS中的一个或多个多行文本文件。当您想将多个文件读入同一Greenplum数据库外部表时，例如当每个JSON文件各自包含单独的记录时，这可能很有用。

PXF支持以这种方式仅读取文本和JSON文件。

**Note:** 如果要使用PXF读取包含多个记录的JSON文件，请参考[从HDFS读取JSON数据](#)主题。

## 前提条件

尝试从HDFS读取文件之前，请确保已满足PXF Hadoop [前提条件](#)。

## 读取多行文本和JSON文件

您可以将单行和多行文件读取到单个表行中，包括具有嵌入式换行符的文件。如果要读取多个JSON文件，则每个文件必须是完整的记录，并且每个文件必须包含相同的记录类型。

PXF将完整的文件数据读取到单个行和列中。创建外部表以读取多个文件时，必须确保要读取的所有文件都具有相同的类型（文本或JSON）。您还必须指定一个 `text` 或 `json` 列，具体取决于文件类型。

以下语法创建了Greenplum数据库可读的外部表，该表引用HDFS上的一个或多个文本或JSON文件：

```
CREATE EXTERNAL TABLE <table_name>
 (<column_name> text|json | LIKE <other_table>)
 LOCATION ('pxf://<path-to-files>?
PROFILE=hdfs:text:multi[&SERVER=
<server_name>]&FILE_AS_ROW=true')
 FORMAT 'CSV');
```

下表描述了此CREATE EXTERNAL TABLE命令中使用的关键字和值。

| 关键词                      | 值                                             |
|--------------------------|-----------------------------------------------|
| <path-to-files>          | HDFS数据存储中目录或文件的绝对路径。                          |
| PROFILE                  | PROFILE关键字必须指定<br>hdfs:text:multi。            |
| SERVER=<br><server_name> | PXF用于访问数据的命名服务器配置。可选的；如果未指定，PXF将使用default服务器。 |
| FILE_AS_ROW=true         | 指示PXF将每个文件读入单个表行的必需选项。                        |
| FORMAT                   | FORMAT必须指定为'CSV'。                             |

**Note:** 当您指定 FILE\_AS\_ROW=true 选项时，hdfs:text:multi 配置文件不支持其他格式选项。

例如，如果 /data/pxf\_examples/jdir 标识了包含多个JSON文件的HDFS目录，则以下语句将创建一个Greenplum数据库外部表，该表引用该目录中的所有文件：

```
CREATE EXTERNAL TABLE pxf_readjfiles(j1 json)
 LOCATION ('pxf://data/pxf_examples/jdir?
PROFILE=hdfs:text:multi&FILE_AS_ROW=true')
FORMAT 'CSV';
```

当您使用 SELECT 语句查询 pxf\_readjfiles 表时，PXF 将 jdir/ 中的每个JSON文件的内容作为外部表中的单独行返回。

读取JSON文件时，可以使用Greenplum数据库中提供的JSON函数来访问JSON记录中的各个数据字段。例如，如果上面的 pxf\_readjfiles 外部表读取包含此JSON记录的JSON文件：

```
{
 "root": [
 {
 "record_obj": {
 "created_at": "MonSep3004:04:53+00002013",
 "id_str": "384529256681725952",
 "user": {
 "id": 31424214,
 "location": "COLUMBUS"
 }
 }
 }
]
}
```

```

 "coordinates":null
 }
}
]
}
}

```

您可以使用 `json_array_elements()` 函数从表行中提取特定的JSON字段。例如，以下命令显示 `user->id` 字段：

```

SELECT json_array_elements(j1->'root')->'record_obj'-'>'user'-
>'id'
AS userid FROM pxf_readjfiles;

userid

31424214
(1 rows)

```

有关使用Greenplum数据库操作JSON数据的特定信息，请参考[使用JSON数据](#)。

## 示例：将HDFS文本文件读入单个表行

执行以下过程在HDFS目录中创建3个示例文本文件，并使用PXF `hdfs:text:multi` 配置文件和默认的PXF服务器在单个外部表查询中读取所有这些文本文件。

1. 为文本文件创建一个HDFS目录。例如：

```
$ hdfs dfs -mkdir -p /data/pxf_examples/tdir
```

2. 创建一个名为 `file1.txt` 的文本数据文件：

```
$ echo 'text file with only one line' > /tmp/file1.txt
```

3. 创建另一个名为 `file2.txt` 的文本数据文件：

```
$ echo 'Prague,Jan,101,4875.33
Rome,Mar,87,1557.39
Bangalore,May,317,8936.99
Beijing,Jul,411,11600.67' > /tmp/file2.txt
```

这个文件有多行。

#### 4. 创建一个名为 `/tmp/file3.txt` 的第三个文本文件：

```
$ echo '"4627 Star Rd.
San Francisco, CA 94107":Sept:2017
"113 Moon St.
San Diego, CA 92093":Jan:2018
"51 Belt Ct.
Denver, CO 90123":Dec:2016
"93114 Radial Rd.
Chicago, IL 60605":Jul:2017
"7301 Brookview Ave.
Columbus, OH 43213":Dec:2018' > /tmp/file3.txt
```

该文件包括嵌入式换行符。

#### 5. 保存文件并退出编辑器。

#### 6. 将文本文件复制到HDFS：

```
$ hdfs dfs -put /tmp/file1.txt /data/pxf_examples/tdir
$ hdfs dfs -put /tmp/file2.txt /data/pxf_examples/tdir
$ hdfs dfs -put /tmp/file3.txt /data/pxf_examples/tdir
```

#### 7. 登录到Greenplum数据库系统并启动 `psql` 子系统。

#### 8. 使用 `hdfs:text:multi` 配置文件来创建引用 `tdir` HDFS目录的外部表。例如：

```
CREATE EXTERNAL TABLE pxf_readfileasrow(c1 text)
LOCATION ('pxf://data/pxf_examples/tdir?
PROFILE=hdfs:text:multi&FILE_AS_ROW=true')
FORMAT 'CSV';
```

#### 9. 打开扩展显示并查询 `pxf_readfileasrow` 表：

```
postgres=# \x on
postgres=# SELECT * FROM pxf_readfileasrow;
```

```
- [RECORD 1]-----
c1 | Prague,Jan,101,4875.33
| Rome,Mar,87,1557.39
| Bangalore,May,317,8936.99
| Beijing,Jul,411,11600.67
-[RECORD 2]-----
c1 | text file with only one line
-[RECORD 3]-----
c1 | "4627 Star Rd.
| San Francisco, CA 94107":Sept:2017
```

```
| "113 Moon St.
| San Diego, CA 92093":Jan:2018
| "51 Belt Ct.
| Denver, CO 80123":Dec:2016
| "93114 Radial Rd.
| Chicago, IL 60605":Jul:2017
| "7301 Brookview Ave.
| Columbus, OH 43213":Dec:2018
```

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

- [PXF介绍](#)
- [PXF管理手册](#)
- [使用PXF访问hadoop](#)
- [使用PXF访问Azure, Google云端存储, Minio和S3对象存储](#)

[About Accessing the S3 Object Store](#)

读取和写入文本数据

读取Avro数据

读取JSON数据

读写Parquet数据

读写SequenceFile数据

将多行文本文件读入单个表行

使用S3 Select从S3读取CSV和Parquet数据

使用PXF访问SQL数据库(JDBC)

PXF故障排除

- [PXF实用程序手册](#)

[Back to the Administrator Guide](#)

### In this topic:

- [先决条件](#)
- [创建外部表](#)
- [示例](#)

PXF对象存储连接器支持将多行文本文件读取为单个表行。本节介绍如何使用PXF读取对象存储中的多行文本和JSON数据文件，包括如何创建引用存储中多个文件的外部表。

PXF支持以这种方式仅读取文本和JSON文件。

注意: 从对象存储访问多行文件与访问HDFS中的多行文件非常相似。本主题标识读取这些文件所需的特定于对象存储的信息。有关更多信息，请参考[PXF HDFS 文档](#)。

## 先决条件

在尝试从驻留在对象存储中的多个文件中读取数据之前，请确保已满足PXF对象存储的[先决条件](#)。

## 创建外部表

使用 `<objstore>:hdfs:multi` 配置文件将对象存储中的多个文件读入单个表行。PXF支持以下 `<objstore>` 配置文件前缀：

| 对象存储                 | 配置文件前缀 |
|----------------------|--------|
| Azure Blob Storage   | wasbs  |
| Azure Data Lake      | adl    |
| Google Cloud Storage | gs     |
| Minio                | s3     |
| S3                   | s3     |

以下语句创建了Greenplum数据库可读的外部表，该表引用对象存储中的一个或多个文本文件：

```
CREATE EXTERNAL TABLE <table_name>
 (<column_name> text|json | LIKE <other_table>)
 LOCATION ('pxf://<path-to-files>?PROFILE=
<objstore>:text:multi[&SERVER=<server_name>]&FILE_AS_ROW=true')
 FORMAT 'CSV');
```

下表描述了`CREATE EXTERNAL TABLE`命令中使用的特定关键字和值。

| 关键字              | 值                                                                      |
|------------------|------------------------------------------------------------------------|
| <path-to-files>  | 对象存储中目录或文件的绝对路径。                                                       |
| PROFILE=         | <code>PROFILE</code> 关键字必须标识特定的对象存储。例如， <code>s3: text: multi</code> 。 |
| SERVER=          | PXF用于访问数据的命名服务器配置。可选的；如果未指定，PXF将使用 <code>default</code> 服务器。           |
| FILE_AS_ROW=true | 指示PXF将每个文件读入单个表行的必需选项。                                                 |
| FORMAT           | <code>FORMAT</code> 必须指定 <code>'CSV'</code> 。                          |

如果要访问S3对象存储，则可以通过`CREATE EXTERNAL TABLE`命令中的自定义选项提供S3凭据，如使用[DDL覆盖S3服务器配置](#)中所述。

## 示例

有关示例，请参阅PXF HDFS文档中的[示例：将HDFS文本文件读入单个表行](#)。您必须对使用对象库运行示例进行的修改包括：

- 将文件复制到对象存储而不是HDFS。例如，要将文件复制到S3：

```
$ aws s3 cp /tmp/file1.txt s3://BUCKET/pxf_examples/tdir
$ aws s3 cp /tmp/file2.txt s3://BUCKET/pxf_examples/tdir
$ aws s3 cp /tmp/file3.txt s3://BUCKET/pxf_examples/tdir
```

- 使用上述的`CREATE EXTERNAL TABLE`语法和`LOCATION`关键字和设置。例如，如果您的服务器名称是`s3srvcfg`：

```
CREATE EXTERNAL TABLE pxf_readfileasrow_s3(c1 text)
 LOCATION('pxf://BUCKET/pxf_examples/tdir?
PROFILE=s3:text:multi&SERVER=s3srvcfg&FILE_AS_ROW=true')
 FORMAT 'CSV'
```



# 取CSV和Parquet数据

Greenplum数据库® 6.0文档

Greenplum平台扩展框架(PXF)

- PXF介绍
- PXF管理手册
- 使用PXF访问hadoop
- 使用PXF访问Azure, Google云端存储, Minio和S3对象存储

About Accessing the S3 Object Store

读取和写入文本数据

读取Avro数据

读取JSON数据

读写Parquet数据

读写SequenceFile数据

将多行文本文件读入单个表行

使用S3 Select从S3读取CSV和Parquet数据

使用PXF访问SQL数据库(JDBC)

PXF故障排除

- PXF实用程序手册

[Back to the Administrator Guide](#)

## In this topic:

- [启用PXF以使用S3 Select](#)
- [使用S3 Select读取Parquet数据](#)
  - [指定Parquet列的压缩类型](#)
  - [创建外部表](#)
- [使用S3 Select读取CSV文件](#)
  - [处理CSV文件头](#)
  - [指定CSV文件压缩类型](#)
  - [创建外部表](#)

PXF S3连接器支持使用Amazon S3 Select服务从S3读取某些CSV和Parquet格式的数据。 S3 Select提供了对Amazon S3中存储的数据的直接就地查询功能。

启用它后，PXF使用S3 Select过滤S3对象的内容以检索您请求的数据子集。这样通常可以减少传输到Greenplum数据库的数据量和查询时间。

可以将PXF S3连接器与S3选择一起使用以读取：

- `gzip` 或 `bzip2` 压缩的CSV文件
- 带有 `gzip` 或 `snappy` 压缩列的 `Parquet` 文件

数据必须是 `UTF-8` 编码的，并且可能是服务器端加密的。

当使用S3 Select时，PXF支持列投影以及 `AND` 和 `OR` 或 `NOT` 运算符的谓词下推。

使用Amazon S3 Select服务可能会增加数据访问和检索的成本。在启用PXF使用S3 Select服务之前，请务必考虑相关开销。

## 启用PXF以使用S3 Select

在访问S3对象存储库时，`s3_SELECT` 外部表自定义选项将控制PXF对S3 Select的使用。设置 `s3_SELECT` 选项时，可以提供以下值：

Greenplum database 5管理员

| S3-SELECT值 | 描述                            |
|------------|-------------------------------|
| OFF        | PXF不使用S3 Select。 默认值。         |
| ON         | PXF始终使用S3 Select。             |
| AUTO       | PXF将在有利于访问或性能的情况下使用S3 Select。 |

默认情况下，PXF不使用S3 Select（`s3_select = OFF`）。您可以使PXF始终使用S3 Select，或者仅在PXF确定它可能对性能有利时才使用S3 Select。例如，当`s3_select = AUTO`时，当外部表上的查询使用列投影或谓词下推时，或者当引用的CSV文件具有标题行时，PXF会自动使用S3 Select。

## 使用S3 Select读取Parquet数据

PXF支持从S3读取Parquet数据，如[在对象存储中读取和写入Parquet数据](#)中所述。如果您希望PXF在读取Parquet数据时使用S3 Select，则将`s3_select`自定义选项和值添加到`CREATE EXTERNAL TABLE` `LOCATION` URI中。

### 指定Parquet列的压缩类型

如果Parquet文件中的列是`gzip`或`snappy`压缩的，请在`LOCATION` URI中使用`COMPRESSION_CODEC`自定义选项来标识压缩编解码器别名。例如：

```
&COMPRESSION_CODEC=gzip
```

或，

```
&COMPRESSION_CODEC=snappy
```

## 创建外部表

使用以下语法在您希望PXF使用S3 Select服务访问的S3上引用一个Parquet文件的情况下，创建Greenplum数据库外部表：

```
CREATE EXTERNAL TABLE <table_name>
 (<column_name> <data_type> [, ...] | LIKE <other_table>)
 LOCATION ('pxf://<path-to-file>?PROFILE=s3:parquet[&SERVER=
<server_name>]&S3_SELECT=ON|AUTO[&<other-custom-option>=<value>
[...]]')
 FORMAT 'CSV';
```

当您启用PXF以使用S3 Select来访问S3上Parquet文件的外部表时，必须指定 `FORMAT 'CSV'`。

例如，使用以下命令让PXF使用S3在最佳时机访问S3上的Parquet文件：

```
CREATE EXTERNAL TABLE parquet_on_s3 (LIKE table1)
 LOCATION ('pxf://bucket/file.parquet?
PROFILE=s3:parquet&SERVER=s3srvcfg&S3_SELECT=AUTO')
 FORMAT 'CSV';
```

## 使用S3 Select读取CSV文件

PXF支持从S3读取CSV数据，如[在对象存储中读取和写入文本数据](#)中所述。如果您希望PXF在读取CSV数据时使用S3 Select，则可以将 `S3_SELECT` 自定义选项和值添加到 `CREATE EXTERNAL TABLE` `LOCATION` URI中。您也可以指定定界符，文件头和压缩自定义选项。

## 处理CSV文件头

当您启用PXF以使用S3 Select访问CSV格式的文件时，可以在 `LOCATION` URI中使用 `FILE_HEADER` 自定义选项来标识CSV文件是否具有标题行，如果有的话，还可以确定PXF如何处理标题 `FILE_HEADER` 选项采用以下值：

| FILE_HEADER值 | 描述             |
|--------------|----------------|
| NONE         | 该文件没有标题行；默认值。  |
| IGNORE       | 该文件具有标题行；忽略标题。 |
| USE          | 该文件具有标题行；读取标题。 |

默认的 `FILE_HEADER` 值为 `NONE`。您还可以指示PXF忽略或读取文件头。例如，要让PXF忽略标题，请将以下内容添加到

`CREATE EXTERNAL TABLE` `LOCATION` `URI中:`

```
&FILE_HEADER=IGNORE
```

仅当S3连接器使用Amazon S3 Select服务访问S3上的文件时，PXF可以读取带标题行的CSV文件。PXF不支持从任何其他外部数据存储中读取包含头行的CSV文件。

## 指定CSV文件压缩类型

如果CSV文件是`gzip`或`bzip2`压缩文件，请使用`LOCATION` URI中的`COMPRESSION_CODEC`自定义选项来标识压缩编解码器别名。例如：

```
&COMPRESSION_CODEC=gzip
```

或，

```
&COMPRESSION_CODEC=bzip2
```

## 创建外部表

使用以下语法创建Greenplum数据库外部表，该表引用您希望PXF使用S3 Select服务访问的S3上的CSV文件：

```
CREATE EXTERNAL TABLE <table_name>
(<column_name> <data_type> [, ...] | LIKE <other_table>)
LOCATION ('pxf://<path-to-file>
?PROFILE=s3:text[&SERVER=
<server_name>]&S3_SELECT=ON|AUTO[&FILE_HEADER=IGNORE|USE]
[&COMPRESSION_CODEC=gzip|bzip2][&<other-custom-option>=<value>
[...]]')
FORMAT 'CSV' [(<delimiter> '<delim_char>')];
```

例如，使用以下命令使PXF始终使用S3 Select来访问S3上的`gzip`压缩文件，其中字段分隔符是竖线（‘|’）字符，并且您想读取标题行：

```
CREATE EXTERNAL TABLE gzippedcsv_on_s3 (LIKE table2)
LOCATION ('pxf://bucket/file.csv.gz?
PROFILE=s3:text&SERVER=s3srvcfg&S3_SELECT=ON&FILE_HEADER=USE')
FORMAT 'CSV' (<delimiter> '|');
```

