Copyright © 2017 Pivotal Software, Inc. All rights reserved.

Pivotal Software, Inc. believes the information in this publication is accurate as of its publication date. The information is subject to change without notice. THE INFORMATION IN THIS PUBLICATION IS PROVIDED "AS IS." PIVOTAL SOFTWARE, INC. ("Pivotal") MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Use, copying, and distribution of any Pivotal software described in this publication requires an applicable software license.

All trademarks used herein are the property of Pivotal or their respective owners.

Revised October 2017 (5.1.0)
Chapter 9: Greenplum Database Security Configuration Guide........ 474

Securing the Database......................................................... 475
Greenplum Database Ports and Protocols............................ 476
Configuring Client Authentication........................................ 480
Allowing Connections to Greenplum Database......................... 480
Editing the pg_hba.conf File.................................................... 481
## Contents

<table>
<thead>
<tr>
<th>Chapter 13: Greenplum Database Utility Guide</th>
<th>594</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management Utility Reference</td>
<td>595</td>
</tr>
<tr>
<td>Backend Server Programs</td>
<td>596</td>
</tr>
<tr>
<td>analyzedb</td>
<td>597</td>
</tr>
<tr>
<td>gactivatestandby</td>
<td>601</td>
</tr>
<tr>
<td>gpaddmirrors</td>
<td>602</td>
</tr>
<tr>
<td>gpcheck</td>
<td>606</td>
</tr>
<tr>
<td>gpcheckcat</td>
<td>608</td>
</tr>
<tr>
<td>gpcheckperf</td>
<td>610</td>
</tr>
<tr>
<td>gpconfig</td>
<td>613</td>
</tr>
<tr>
<td>gpcrondump</td>
<td>617</td>
</tr>
<tr>
<td>gpdbrestore</td>
<td>630</td>
</tr>
<tr>
<td>gpdeletefilesystem</td>
<td>637</td>
</tr>
<tr>
<td>gpexpand</td>
<td>638</td>
</tr>
<tr>
<td>gpfdist</td>
<td>641</td>
</tr>
<tr>
<td>gpfilesystem</td>
<td>644</td>
</tr>
<tr>
<td>gpinistandby</td>
<td>647</td>
</tr>
<tr>
<td>gpinitsystem</td>
<td>650</td>
</tr>
<tr>
<td>gupload</td>
<td>657</td>
</tr>
<tr>
<td>gplogfilter</td>
<td>666</td>
</tr>
<tr>
<td>gpmapreduce</td>
<td>669</td>
</tr>
<tr>
<td>gpmdir</td>
<td>670</td>
</tr>
<tr>
<td>gpperfmon_install</td>
<td>674</td>
</tr>
<tr>
<td>gppkg</td>
<td>678</td>
</tr>
<tr>
<td>gprecover</td>
<td>680</td>
</tr>
<tr>
<td>gprecoverseg</td>
<td>684</td>
</tr>
<tr>
<td>gpreload</td>
<td>686</td>
</tr>
<tr>
<td>gpscp</td>
<td>687</td>
</tr>
<tr>
<td>gpseginstall</td>
<td>689</td>
</tr>
<tr>
<td>gpssh</td>
<td>692</td>
</tr>
<tr>
<td>gpssh-exkeys</td>
<td>694</td>
</tr>
<tr>
<td>gpstart</td>
<td>696</td>
</tr>
<tr>
<td>gpstate</td>
<td>699</td>
</tr>
<tr>
<td>gpstop</td>
<td>700</td>
</tr>
<tr>
<td>gpsy1</td>
<td>702</td>
</tr>
<tr>
<td>gptransfer</td>
<td>703</td>
</tr>
<tr>
<td>pgbouncer</td>
<td>715</td>
</tr>
<tr>
<td>Client Utility Reference</td>
<td>735</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tuning SQL Queries</th>
<th>580</th>
</tr>
</thead>
<tbody>
<tr>
<td>How to Generate Explain Plans</td>
<td>580</td>
</tr>
<tr>
<td>How to Read Explain Plans</td>
<td>580</td>
</tr>
<tr>
<td>Optimizing Greenplum Queries</td>
<td>582</td>
</tr>
<tr>
<td>High Availability</td>
<td>584</td>
</tr>
<tr>
<td>Disk Storage</td>
<td>584</td>
</tr>
<tr>
<td>Master Mirroring</td>
<td>584</td>
</tr>
<tr>
<td>Segment Mirroring</td>
<td>585</td>
</tr>
<tr>
<td>Dual Clusters</td>
<td>586</td>
</tr>
<tr>
<td>Backup and Restore</td>
<td>586</td>
</tr>
<tr>
<td>Detecting Failed Master and Segment Instances</td>
<td>587</td>
</tr>
<tr>
<td>Segment Mirroring Configuration</td>
<td>588</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Release Notes</th>
<th>7</th>
</tr>
</thead>
</table>

- Configuring the Greenplum Cluster: 574
- Creating and Installing Keytab Files: 575
- Configuring gphdfs for Kerberos: 576
- Testing Greenplum Database Access to HDFS: 577
- Troubleshooting HDFS with Kerberos: 578
- Optimizing Greenplum Queries: 580
- How to Generate Explain Plans: 580
- How to Read Explain Plans: 580
- Tuning SQL Queries: 582
- High Availability: 584
- Disk Storage: 584
- Master Mirroring: 584
- Segment Mirroring: 585
- Dual Clusters: 586
- Backup and Restore: 586
- Detecting Failed Master and Segment Instances: 587
- Segment Mirroring Configuration: 588

## Chapter 13: Greenplum Database Utility Guide

### Management Utility Reference

<table>
<thead>
<tr>
<th>Utility</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backend Server Programs</td>
<td>595</td>
</tr>
<tr>
<td>analyzedb</td>
<td>597</td>
</tr>
<tr>
<td>gactivatestandby</td>
<td>601</td>
</tr>
<tr>
<td>gpaddmirrors</td>
<td>602</td>
</tr>
<tr>
<td>gpcheck</td>
<td>606</td>
</tr>
<tr>
<td>gpcheckcat</td>
<td>608</td>
</tr>
<tr>
<td>gpcheckperf</td>
<td>610</td>
</tr>
<tr>
<td>gpconfig</td>
<td>613</td>
</tr>
<tr>
<td>gpcrondump</td>
<td>617</td>
</tr>
<tr>
<td>gpdbrestore</td>
<td>630</td>
</tr>
<tr>
<td>gpdeletefilesystem</td>
<td>637</td>
</tr>
<tr>
<td>gpexpand</td>
<td>638</td>
</tr>
<tr>
<td>gpfdist</td>
<td>641</td>
</tr>
<tr>
<td>gpfilesystem</td>
<td>644</td>
</tr>
<tr>
<td>gpinistandby</td>
<td>647</td>
</tr>
<tr>
<td>gpinitsystem</td>
<td>650</td>
</tr>
<tr>
<td>gupload</td>
<td>657</td>
</tr>
<tr>
<td>gplogfilter</td>
<td>666</td>
</tr>
<tr>
<td>gpmapreduce</td>
<td>669</td>
</tr>
<tr>
<td>gpmdir</td>
<td>670</td>
</tr>
<tr>
<td>gpperfmon_install</td>
<td>674</td>
</tr>
<tr>
<td>gppkg</td>
<td>678</td>
</tr>
<tr>
<td>gprecover</td>
<td>680</td>
</tr>
<tr>
<td>gprecoverseg</td>
<td>684</td>
</tr>
<tr>
<td>gpreload</td>
<td>686</td>
</tr>
<tr>
<td>gpscp</td>
<td>687</td>
</tr>
<tr>
<td>gpseginstall</td>
<td>689</td>
</tr>
<tr>
<td>gpssh</td>
<td>692</td>
</tr>
<tr>
<td>gpssh-exkeys</td>
<td>694</td>
</tr>
<tr>
<td>gpstart</td>
<td>696</td>
</tr>
<tr>
<td>gpstate</td>
<td>699</td>
</tr>
<tr>
<td>gpstop</td>
<td>700</td>
</tr>
<tr>
<td>gpsy1</td>
<td>702</td>
</tr>
<tr>
<td>gptransfer</td>
<td>703</td>
</tr>
<tr>
<td>pgbouncer</td>
<td>715</td>
</tr>
</tbody>
</table>

### Client Utility Reference

<table>
<thead>
<tr>
<th>Utility</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client Utility Reference</td>
<td>735</td>
</tr>
<tr>
<td>Command</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>ALTER SEQUENCE</td>
<td>872</td>
</tr>
<tr>
<td>ALTER TABLE</td>
<td>874</td>
</tr>
<tr>
<td>ALTER TABLESPACE</td>
<td>884</td>
</tr>
<tr>
<td>ALTER TYPE</td>
<td>885</td>
</tr>
<tr>
<td>ALTER USER</td>
<td>885</td>
</tr>
<tr>
<td>ALTER VIEW</td>
<td>886</td>
</tr>
<tr>
<td>ANALYZE</td>
<td>887</td>
</tr>
<tr>
<td>BEGIN</td>
<td>890</td>
</tr>
<tr>
<td>CHECKPOINT</td>
<td>892</td>
</tr>
<tr>
<td>CLOSE</td>
<td>892</td>
</tr>
<tr>
<td>CLUSTER</td>
<td>893</td>
</tr>
<tr>
<td>COMMENT</td>
<td>894</td>
</tr>
<tr>
<td>COMMIT</td>
<td>896</td>
</tr>
<tr>
<td>COPY</td>
<td>897</td>
</tr>
<tr>
<td>CREATE AGGREGATE</td>
<td>907</td>
</tr>
<tr>
<td>CREATE CAST</td>
<td>911</td>
</tr>
<tr>
<td>CREATE CONVERSION</td>
<td>914</td>
</tr>
<tr>
<td>CREATE DATABASE</td>
<td>915</td>
</tr>
<tr>
<td>CREATE DOMAIN</td>
<td>916</td>
</tr>
<tr>
<td>CREATE EXTENSION</td>
<td>918</td>
</tr>
<tr>
<td>CREATE EXTERNAL TABLE</td>
<td>919</td>
</tr>
<tr>
<td>CREATE FUNCTION</td>
<td>928</td>
</tr>
<tr>
<td>CREATE GROUP</td>
<td>934</td>
</tr>
<tr>
<td>CREATE INDEX</td>
<td>935</td>
</tr>
<tr>
<td>CREATE LANGUAGE</td>
<td>938</td>
</tr>
<tr>
<td>CREATE OPERATOR</td>
<td>940</td>
</tr>
<tr>
<td>CREATE OPERATOR CLASS</td>
<td>944</td>
</tr>
<tr>
<td>CREATE OPERATOR FAMILY</td>
<td>947</td>
</tr>
<tr>
<td>CREATE PROTOCOL</td>
<td>948</td>
</tr>
<tr>
<td>CREATE RESOURCE GROUP</td>
<td>949</td>
</tr>
<tr>
<td>CREATE RESOURCE QUEUE</td>
<td>951</td>
</tr>
<tr>
<td>CREATE ROLE</td>
<td>951</td>
</tr>
<tr>
<td>CREATE RULE</td>
<td>954</td>
</tr>
<tr>
<td>CREATE SCHEMA</td>
<td>959</td>
</tr>
<tr>
<td>CREATE SEQUENCE</td>
<td>960</td>
</tr>
<tr>
<td>CREATE TABLE</td>
<td>962</td>
</tr>
<tr>
<td>CREATE TABLE AS</td>
<td>964</td>
</tr>
<tr>
<td>CREATE TABLESPACE</td>
<td>976</td>
</tr>
<tr>
<td>CREATE TYPE</td>
<td>979</td>
</tr>
<tr>
<td>CREATE USER</td>
<td>980</td>
</tr>
<tr>
<td>CREATE VIEW</td>
<td>985</td>
</tr>
<tr>
<td>DEALLOCATE</td>
<td>986</td>
</tr>
<tr>
<td>DECLARE</td>
<td>988</td>
</tr>
<tr>
<td>DELETE</td>
<td>989</td>
</tr>
<tr>
<td>DISCARD</td>
<td>991</td>
</tr>
<tr>
<td>DO</td>
<td>993</td>
</tr>
<tr>
<td>DROP AGGREGATE</td>
<td>994</td>
</tr>
<tr>
<td>DROP CAST</td>
<td>996</td>
</tr>
<tr>
<td>DROP CONVERSION</td>
<td>996</td>
</tr>
<tr>
<td>DROP DATABASE</td>
<td>997</td>
</tr>
<tr>
<td>DROP DATABASE</td>
<td>998</td>
</tr>
<tr>
<td>DROP DOMAIN</td>
<td>998</td>
</tr>
<tr>
<td>DROP EXTENSION</td>
<td>999</td>
</tr>
<tr>
<td>DROP EXTERNAL TABLE</td>
<td>999</td>
</tr>
<tr>
<td>DROP FILESPACE</td>
<td>1000</td>
</tr>
<tr>
<td>DROP FUNCTION</td>
<td>1001</td>
</tr>
<tr>
<td>DROP FUNCTION</td>
<td>1002</td>
</tr>
<tr>
<td>DROP GROUP</td>
<td>1003</td>
</tr>
</tbody>
</table>
The gp_toolkit Administrative Schema......................................................................................... 1194
Checking for Tables that Need Routine Maintenance................................................................. 1194
Checking for Locks..................................................................................................................... 1195
Checking Append-Optimized Tables............................................................................................ 1197
Viewing Greenplum Database Server Log Files........................................................................ 1201
Checking Server Configuration Files.......................................................................................... 1204
Checking for Failed Segments................................................................................................... 1205
Checking Resource Queue Activity and Status.......................................................................... 1206
Checking Query Disk Spill Space Usage.................................................................................... 1208
Viewing Users and Groups (Roles)............................................................................................. 1210
Checking Database Object Sizes and Disk Space...................................................................... 1211
Checking for Uneven Data Distribution.................................................................................... 1215
The gpperfmon Database.............................................................................................................. 1216
database_*................................................................................................................................ 1218
diskspace_*................................................................................................................................ 1219
filerep_*...................................................................................................................................... 1219
interface_stats_*......................................................................................................................... 1224
log_alert_*.................................................................................................................................. 1225
queries_*.................................................................................................................................... 1227
segment_*................................................................................................................................... 1229
socket_stats_*............................................................................................................................ 1230
system_*..................................................................................................................................... 1231
dynamic_memory_info............................................................................................................... 1232
memory_info............................................................................................................................ 1233
Greenplum Database Data Types................................................................................................... 1235
Character Set Support.................................................................................................................. 1239
Setting the Character Set............................................................................................................ 1241
Character Set Conversion Between Server and Client............................................................... 1241
Server Configuration Parameters............................................................................................... 1244
Parameter Types and Values........................................................................................................ 1244
Setting Parameters.................................................................................................................... 1244
Parameter Categories.................................................................................................................. 1245
Configuration Parameters.......................................................................................................... 1255
Summary of Built-in Functions.................................................................................................... 1334
Greenplum Database Function Types........................................................................................... 1334
Built-in Functions and Operators................................................................................................ 1335
JSON Functions and Operators.................................................................................................... 1338
Window Functions....................................................................................................................... 1341
Advanced Aggregate Functions.................................................................................................... 1343
Greenplum MapReduce Specification.......................................................................................... 1345
Greenplum MapReduce Document Format................................................................................ 1345
Greenplum MapReduce Document Schema................................................................................ 1346
Example Greenplum MapReduce Document............................................................................... 1353
Greenplum PL/pgSQL Procedural Language................................................................................ 1359
About Greenplum Database PL/pgSQL....................................................................................... 1359
PL/pgSQL Plan Caching................................................................................................................ 1361
PL/pgSQL Examples.................................................................................................................... 1361
References.................................................................................................................................. 1365
Greenplum PostGIS Extension..................................................................................................... 1366
About PostGIS............................................................................................................................. 1366
Enabling and Removing PostGIS Support.................................................................................... 1367
Usage.......................................................................................................................................... 1368
PostGIS Extension Support and Limitations............................................................................... 1368
PostGIS Support Scripts............................................................................................................. 1370
Greenplum PL/R Language Extension........................................................................................ 1372
About Greenplum Database PL/R............................................................................................... 1372
Chapter 17: Greenplum Database UNIX Client Documentation

Chapter 18: Greenplum Database Windows Client Documentation
Welcome to Pivotal Greenplum 5.1.0

Pivotal Greenplum Database is a massively parallel processing (MPP) database server that supports next generation data warehousing and large-scale analytics processing. By automatically partitioning data and running parallel queries, it allows a cluster of servers to operate as a single database supercomputer performing tens or hundreds times faster than a traditional database. It supports SQL, MapReduce parallel processing, and data volumes ranging from hundreds of gigabytes, to hundreds of terabytes.

**Note:** This document contains pertinent release information about Pivotal Greenplum Database 5.1.0. For previous versions of the release notes for Greenplum Database, go to Pivotal Greenplum Database Documentation. For information about Greenplum Database end of life, see Pivotal Greenplum Database end of life policy.

Pivotal Greenplum 5.x software is available for download from Pivotal Network.

**Important:** Pivotal Greenplum 5.1.0 is not yet certified for running on DCA systems. Contact your DCA representative for information about the availability of Greenplum Database 5.1.0 support on the DCA.

**Important:** Significant Pivotal Greenplum performance degradation has been observed when enabling resource group-based workload management (an experimental feature) on RedHat 6.x, CentOS 6.x, and SuSE 11 systems. This issue is caused by a Linux cgroup kernel bug. This kernel bug has been fixed in CentOS 7.x and Red Hat 7.x systems. The performance problem is mitigated in Red Hat 6 systems that use kernel version 2.6.32-696 or higher. SuSE 11 does not have a kernel version that resolves this issue. See known issue 149789783.

Pivotal Greenplum 5.x is based on the open source Greenplum Database project code.

**Important:** Pivotal Support does not provide support for open source versions of Greenplum Database. Only Pivotal Greenplum Database is supported by Pivotal Support.

Greenplum Database 5.1.0 is a minor release that includes product enhancements and changes, and resolves some known issues.
New Features

Pivotal Greenplum 5.1.0 includes these new features.

- GPORCA Enhanced Short-Running Query Performance
- GPORCA Performance Enhancements
- GPORCA Supports Indexes on Leaf Child Partitions
- COPY Table Data to or from a Program
- gptransfer Supports SHA-256 Data Validation
- gprecoverseg Performance Enhancement
- PXF Extension Framework
- New Parameters

**GPORCA Enhanced Short-Running Query Performance**

In Greenplum Database 5.1.0, GPORCA reduces optimization time by retrieving column statistics only for the columns required for cardinality estimation. Eliminating the overhead of retrieving and generating the statistics for columns that are not used in cardinality estimation improves the performance of short running queries.

In previous releases, when only column width information was required by GPORCA, the column statistics was also retrieved.

**GPORCA Performance Enhancements**

Greenplum Database 5.1.0 includes these GPORCA performance enhancements.

- GPORCA has reduced the maximum number of join combinations that are evaluated during query optimization. This reduces the time evaluating join combinations without significantly reducing query performance.

  In previous releases, for queries that contain a large number of joins, GPORCA spent a significant amount of time evaluating all possible multiple join combinations to determine the most efficient join tree to use in the query plan.

- For some queries that contain a correlated subquery and the subquery contains a window function, GPORCA de-correlates the query and produces an efficient plan that uses joins.

  In previous releases, GPORCA generated a suboptimal query plan with correlated execution.

**GPORCA Supports Indexes on Leaf Child Partitions**

In Greenplum Database 5.1.0, if an index exists on a leaf child partition, GPORCA considers the index when generating a query plan for queries against the leaf child partition. In previous releases, GPORCA did not consider the index when generating plans even though using the index would benefit the query that directly accesses the leaf child partition.

**COPY Table Data to or from a Program**

In Greenplum Database 5.1.0, the COPY command supports the PROGRAM 'command' clause to copy table data to or from a program. The command must be specified from the viewpoint of the Greenplum Database master host system, and must be executable by the Greenplum Database administrator user (gpadm). For the COPY FROM command, the input is read from standard output of the command, and for the COPY TO command, the output is written to the standard input of the command.

For more information about the command, see the COPY command.
**gptransfer Supports SHA-256 Data Validation**

In Greenplum Database 5.1.0, the Greenplum Database gptransfer utility supports validating transferred table data with SHA-256 checksums when copying database objects between source and destination systems. Specify `--validate value sha256` to compare SHA-256 checksums values between source and destination table data.

On a Greenplum Database system with FIPS enabled, the option `md5` is not supported. Use the option `count` or `sha256` to validate table data.

**Note:** Validating table data with SHA-256 requires the Greenplum Database pgcrypto extension. The extension is included with Pivotal Greenplum 5.x. When copying data from a supported Pivotal Greenplum 4.3.x system, the extension package must be installed on the 4.3.x system. You do not need to run `pgcrypto.sql` to install the pgcrypto functions in a Greenplum 4.3.x database.

For more information about the utility, see `gptransfer`.

**gprecoverseg Performance Enhancement**

In Greenplum Database 5.1.0, the performance of the Greenplum Database gprecoverseg utility has been improved when performing directory deletes as part of a full recovery (`gprecoverseg -F` command). gprecoverseg now uses `rsync` instead of an `rm -rf` operation against directories, which improves delete performance with a large number of files.

**PXF Extension Framework**

The PXF Extension Framework (PXF) is integrated with Greenplum Database 5.1.0. PXF enables you to access external HDFS and Hive data stores from a Greenplum Database external table, essentially providing an ANSI-compliant SQL interface to virtually any dataset. The Greenplum Database PXF Extension Framework is based on PXF from Apache HAWQ (incubating).

For information about configuring and using the PXF Extension Framework in your Greenplum Database deployment, refer to Accessing HDFS and Hive Data with PXF in the Greenplum Database Administrator Guide.

**New Parameters**

Greenplum Database 5.1.0 includes these new server configuration parameters.

- `gp_interconnect_debug_retry_interval`
- `gp_log_interconnect`
- `optimizer_join_arity_for_associativity_commutativity`

For information about Greenplum Database server configuration parameters, see the Greenplum Database Reference Guide.

**gp_interconnect_debug_retry_interval**

Specifies the interval, in seconds, to log Greenplum Database interconnect debugging messages when the server configuration parameter `gp_log_interconnect` is set to `DEBUG`. The default is 10 seconds.

The log messages contain information about the interconnect communication between Greenplum Database segment instance worker processes. The information can be helpful when debugging network issues between segment instances.
gp_log_interconnect

Controls the amount of information that is written to the log file about communication between Greenplum Database segment instance worker processes. The default value is terse. The log information is written to both the master and segment instance logs.

Increasing the amount of logging could affect performance and increase disk space usage.

optimizer_join_arity_for_associativity_commutativity

The value is an optimization hint to limit the number of join associativity and join commutativity transformations explored during query optimization. The limit controls the alternative plans that GPORCA considers during query optimization. For example, the default value of 7 is an optimization hint for GPORCA to stop exploring join associativity and join commutativity transformations when an n-ary join operator has more than 7 children during optimization.

For a query with a large number of joins, specifying a lower value improves query performance by limiting the number of alternate query plans that GPORCA evaluates. However, setting the value too low might cause GPORCA to generate a query plan that performs sub-optimally.

This parameter can be set for a database system or a session.
Experimental Features

Because Pivotal Greenplum Database is based on the open source Greenplum Database project code, it includes several experimental features to allow interested developers to experiment with their use on development systems. Feedback will help drive development of these features, and they may become supported in future versions of the product.

**Warning:** Experimental features are not recommended or supported for production deployments. These features may change in or be removed from future versions of the product based on further testing and feedback. Moreover, any features that may be visible in the open source code but that are not described in the product documentation should be considered experimental and unsupported for production use.

Key experimental features in Greenplum Database 5.1.0 include:

- Recursive WITH Queries (Common Table Expressions). See [WITH Queries (Common Table Expressions)](https://www.pivotalgreenplum.com) in the Pivotal Greenplum Database Documentation.

**Important:** Significant Pivotal Greenplum performance degradation has been observed when enabling resource group-based workload management (an experimental feature) on RedHat 6.x, CentOS 6.x, and SuSE 11 systems. This issue is caused by a Linux cgroup kernel bug. This kernel bug has been fixed in CentOS 7.x and Red Hat 7.x systems. The performance problem is mitigated in Red Hat 6 systems that use kernel version 2.6.32-696 or higher. SuSE 11 does not have a kernel version that resolves this issue. See known issue 149789783.

See [Known Issues and Limitations](https://www.pivotalgreenplum.com) for information about issues associated with resource groups.

- The pgAdmin 4 tool is compatible with Pivotal Greenplum Database 5.1.0, but is not officially supported by Pivotal. You can use pgAdmin 4 to query Greenplum Database tables and to view Greenplum Database table DDL. Append-optimized tables are expected to work with pgAdmin 4.

Some functionality is either unavailable or not fully implemented when used with Greenplum Database:

- Partitioned tables can cause pgAdmin 4 performance to slow down.
- External tables are not supported.
- The graphical **EXPLAIN** feature of the Query tool is not supported.

See [https://www.pgadmin.org/](https://www.pgadmin.org/) for information about installing and using the tool.
Changed Features

Greenplum Database 5.1.0 includes these feature changes.

- The Greenplum Database utilities gpcrondump and gpdbrestore have changed.
  - If you specify a gpcrondump option to back up schemas, `-s`, `-S`, `-schema-file`, `--schema-exclude-file`, procedural languages that are installed in the database are also backed up even though they are not schema specific. External items such as shared libraries that are used by a language are not backed up.
  - If you specify the gpdbrestore option `-S` to restore a schema, procedural languages that are in a database backup are also restored even though they are not schema specific.

  **Note:** When restoring procedural languages, gprestore logs non-fatal messages if the languages that are being restored already exist in the target database.

In previous releases, functions, that are schema specific, were backed up and restored for a schema specific operation. However, only the PL/Java language was backed up and restored.

- The meta-command `\copy` for the Greenplum Database psql utility has changed. When copying data with `\copy`, enclosing the source or destination with single quotes (`'`) indicates that the source or destination is a file. For example, this `\copy` command copies data to the file `stdout`.

```
\copy testbl to 'stdout'
```

Without the single quotes, the output is sent to command output.

For information about the psql utility, see the [Greenplum Database Utility Guide](#).

- MADlib 1.12 is available with Greenplum Database 5.1.0. When using MADlib 1.12, the server configuration parameter `optimizer_control` can be set to either `on` (the default) or `off`.

  When using MADlib 1.11 with Greenplum Database, some MADlib functions would not work if the parameter was set to `off`.

  For information about the MADlib extension for analytics, see the [Greenplum Database Reference Guide](#).

- The Greenplum Database PL/R extension package version has been updated to 2.3.1.

  After installing the PL/R extension package, the directory is `$GPHOME/ext/R-3.3.3`. The directory name matches the version of R that is installed, 3.3.3. In the previous package, the installation directory was `$GPHOME/ext/R-3.3.1`

  For information about the PL/R extension package, see the [Greenplum Database Reference Guide](#).

- The Greenplum Database Client and Loader Tools for AIX 7 have been updated for Greenplum Database 5.1.0.

- A PostGIS extension package for Red Hat Enterprise Linux 6.x is available on [Pivotal Network](#).
### Deprecated Features

Pivotal Greenplum 5.1.0 deprecates these unused catalog tables. They will be removed in a future release.

- `gp_configuration`
- `gp_db_interfaces`
- `gp_interfaces`
Differences Compared to Open Source Greenplum Database

Pivotal Greenplum 5.x includes all of the functionality in the open source Greenplum Database project and adds:

- Product packaging and installation script.
- Support for QuickLZ compression. QuickLZ compression is not provided in the open source version of Greenplum Database due to licensing restrictions.
- Support for managing Greenplum Database using Pivotal Greenplum Command Center.
- Support for monitoring and managing queries with Pivotal Greenplum Workload Manager.
- Support for full text search and text analysis using Pivotal GPText.
Supported Platforms

Pivotal Greenplum 5.1.0 runs on the following platforms:

- Red Hat Enterprise Linux 64-bit 7.x (See the following Note)
- Red Hat Enterprise Linux 64-bit 6.x
- SuSE Linux Enterprise Server 64-bit 11 SP4 (See the following Note)
- CentOS 64-bit 7.x
- CentOS 64-bit 6.x

**Important:** Pivotal Greenplum 5.1.0 is not yet certified for running on DCA systems. Contact your DCA representative for information about the availability of Greenplum Database 5.1.0 support on the DCA.

**Important:** Significant Pivotal Greenplum performance degradation has been observed when enabling resource group-based workload management (an experimental feature) on RedHat 6.x, CentOS 6.x, and SuSE 11 systems. This issue is caused by a Linux cgroup kernel bug. This kernel bug has been fixed in CentOS 7.x and Red Hat 7.x systems. The performance problem is mitigated in Red Hat 6 systems that use kernel version 2.6.32-696 or higher. SuSE 11 does not have a kernel version that resolves this issue. See known issue 149789783.

**Note:** For Greenplum Database that is installed on Red Hat Enterprise Linux 7.x or CentOS 7.x prior to 7.3, an operating system issue might cause Greenplum Database that is running large workloads to hang in the workload. The Greenplum Database issue is caused by Linux kernel bugs. RHEL 7.3 and CentOS 7.3 resolves the issue.

**Note:** Greenplum Database on SuSE Linux Enterprise systems does not support the PL/Perl procedural language or the gpmapreduce tool.

Pivotal Greenplum 5.1.0 supports these Java versions:

- 8.xxx
- 7.xxx

Greenplum Database 5.10 software that runs on Linux systems uses OpenSSL 1.0.2l (with FIPS 2.0.16), cURL 7.54, OpenLDAP 2.4.44, and Python 2.7.12.

Greenplum Database client software that runs on Windows and AIX systems uses OpenSSL 0.9.8zg.

The Greenplum Database s3 external table protocol supports these data sources:

- Amazon Simple Storage Service (Amazon S3)
- Dell EMC Elastic Cloud Storage (ECS), an Amazon S3 compatible service

Pivotal Greenplum 5.1.0 supports Data Domain Boost on Red Hat Enterprise Linux.

This table lists the versions of Data Domain Boost SDK and DDOS supported by Pivotal Greenplum 5.x.

<table>
<thead>
<tr>
<th>Pivotal Greenplum</th>
<th>Data Domain Boost</th>
<th>DDOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1.0</td>
<td>3.3, 3.0.0.3</td>
<td>6.0 (all versions)</td>
</tr>
<tr>
<td>5.0.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** In addition to the DDOS versions listed in the previous table, Pivotal Greenplum 5.0.0 and later supports all minor patch releases (fourth digit releases) later than the certified version.

**Note:** Pivotal Greenplum 5.1.0 does not support the ODBC driver for Cognos Analytics V11.
Connecting to IBM Cognos software with an ODBC driver is not supported. Greenplum Database supports connecting to IBM Cognos software with the DataDirect JDBC driver for Pivotal Greenplum. This driver is available as a download from Pivotal Network.

**Veritas NetBackup**

Pivotal Greenplum 5.1.0 supports backup with Veritas NetBackup version 7.7.3. See Backing Up Databases with Veritas NetBackup.

**Supported Platform Notes**

The following notes describe platform support for Pivotal Greenplum. Please send any questions or comments to Pivotal Support at https://support.pivotal.io.

- The only file system supported for running Greenplum Database is the XFS file system. All other file systems are explicitly not supported by Pivotal.
- Greenplum Database is supported on all 1U and 2U commodity servers with local storage. Special purpose hardware that is not commodity may be supported at the full discretion of Pivotal Product Management based on the general similarity of the hardware to commodity servers.
- Greenplum Database is supported on network or shared storage if the shared storage is presented as a block device to the servers running Greenplum Database and the XFS file system is mounted on the block device. Network file systems are not supported. When using network or shared storage, Greenplum Database mirroring must be used in the same way as with local storage, and no modifications may be made to the mirroring scheme or the recovery scheme of the segments. Other features of the shared storage such as de-duplication and/or replication are not directly supported by Pivotal Greenplum Database, but may be used with support of the storage vendor as long as they do not interfere with the expected operation of Greenplum Database at the discretion of Pivotal.
- Greenplum Database is supported when running on virtualized systems, as long as the storage is presented as block devices and the XFS file system is mounted for the storage of the segment directories.
- A minimum of 10-gigabit network is required for a system configuration to be supported by Pivotal.
- Greenplum Database is supported on Amazon Web Services (AWS) servers using either Amazon instance store (Amazon uses the volume names ephemeral[0-20]) or Amazon Elastic Block Store (Amazon EBS) storage. If using Amazon EBS storage the storage should be RAID of Amazon EBS volumes and mounted with the XFS file system for it to be a supported configuration.
- For Red Hat Enterprise Linux 7.2 or CentOS 7.2, the default systemd setting RemoveIPC=yes removes IPC connections when non-system users logout. This causes the Greenplum Database utility gpinit system to fail with semaphore errors. To avoid this issue, see "Setting the Greenplum Recommended OS Parameters" in the Greenplum Database Installation Guide.
Pivotal Greenplum Tools and Extensions Compatibility

- **Client Tools**
- **Extensions**
- **Pivotal Greenplum Data Connectors**
- **Pivotal GPText Compatibility**
- **Pivotal Greenplum Command Center**

**Client Tools**

Greenplum releases a number of client tool packages on various platforms that can be used to connect to Greenplum Database and the Greenplum Command Center management tool. The following table describes the compatibility of these packages with this Greenplum Database release.

Tool packages are available from *Pivotal Network*.

**Table 2: Pivotal Greenplum 5.0.0 Tools Compatibility**

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description of Contents</th>
<th>Tool Version</th>
<th>Server Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pivotal Greenplum Clients</td>
<td>Greenplum Database Command-Line Interface (psql)</td>
<td>5.0</td>
<td>5.0.0</td>
</tr>
<tr>
<td>Pivotal Greenplum Loaders</td>
<td>Greenplum Database Parallel Data Loading Tools (gpfdist, gpload)</td>
<td>5.0</td>
<td>5.0.0</td>
</tr>
<tr>
<td>Pivotal Greenplum Command Center</td>
<td>Greenplum Database management tool</td>
<td>3.2.2 and later</td>
<td>5.0.0</td>
</tr>
<tr>
<td>Pivotal Greenplum Workload Manager</td>
<td>Greenplum Database query monitoring and management tool</td>
<td>1.8.0</td>
<td>5.0.0</td>
</tr>
</tbody>
</table>

The Greenplum Database Client Tools and Load Tools are supported on the following platforms:

- AIX 7.2 (64-bit) (Client and Load Tools only)
- Red Hat Enterprise Linux x86_64 6.x (RHEL 6)
- SuSE Linux Enterprise Server x86_64 SLES 11
- Windows 10 (32-bit and 64-bit)
- Windows 8 (32-bit and 64-bit)
- Windows Server 2012 (32-bit and 64-bit)
- Windows Server 2012 R2 (32-bit and 64-bit)
- Windows Server 2008 R2 (32-bit and 64-bit)
Extensions

Table 3: Pivotal Greenplum 5.0.0 Extensions Compatibility

<table>
<thead>
<tr>
<th>Pivotal Greenplum Extension</th>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MADlib machine learning for Greenplum Database 5.0.x(^1)</td>
<td>MADlib 1.12, 1.11</td>
</tr>
<tr>
<td>PL/Java for Greenplum Database 5.0.x</td>
<td>PL/Java 1.4.2, 1.4.0</td>
</tr>
<tr>
<td>PL/R for Greenplum Database 5.0.x</td>
<td>2.3.0, 2.2.0</td>
</tr>
<tr>
<td>PostGIS Spatial and Geographic Objects for Greenplum Database 5.0.x</td>
<td>2.1.5</td>
</tr>
<tr>
<td>Python Data Science Module Package for Greenplum Database 5.0.x(^2)</td>
<td>1.0.0, 1.1.0</td>
</tr>
<tr>
<td>R Data Science Library Package for Greenplum Database 5.0.x(^3)</td>
<td>1.0.0</td>
</tr>
</tbody>
</table>

**Note:**
\(^1\) For information about MADlib support and upgrade information, see the [MADlib FAQ](#).
\(^2\) For information about the Python package, including the modules provided, see the [Python Data Science Module Package](#) in the [Greenplum Database Documentation](#).
\(^3\) For information about the R package, including the libraries provided, see the [R Data Science Library Package](#) in the [Greenplum Database Documentation](#).

These Greenplum Database extensions are installed with Pivotal Greenplum Database:

- Fuzzy String Match Extension
- PL/Python Extension
- pgcrypto Extension

**Pivotal Greenplum Data Connectors**

- PXF Extension Framework - The PXF Extension Framework, integrated with Greenplum Database 5.1.0, provides access to HDFS and Hive external data stores. Refer to Accessing HDFS and Hive Data with PXF in the [Greenplum Database Administrator Guide](#) for PXF configuration and usage information.
- Greenplum-Spark Connector - The Pivotal Greenplum-Spark Connector supports high speed, parallel data transfer from Greenplum Database to an Apache Spark cluster. The Greenplum-Spark Connector is available as a separate download from [Pivotal Network](#). Refer to the [Greenplum-Spark Connector documentation](#) for compatibility and usage information.

**Pivotal GPTText Compatibility**

Pivotal Greenplum Database 5.1.0 is compatible with Pivotal GPTText version 2.1.3 and later.

**Pivotal Greenplum Command Center**

See the [Greenplum Command Center documentation site](#) for GPCC and Greenplum Workload Manager compatibility information.
Hadoop Distribution Compatibility

Greenplum Database provides access to HDFS with gphdfs and PXF.

**gphdfs Hadoop Distribution Compatibility**

The supported Hadoop distributions for gphdfs are listed below:

Table 4: Supported gphdfs Hadoop Distributions

<table>
<thead>
<tr>
<th>Hadoop Distribution</th>
<th>Version</th>
<th>gp_hadoop_target_version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloudera</td>
<td>CDH 5.2, 5.3, 5.4.x - 5.8.x</td>
<td>cdh5</td>
</tr>
<tr>
<td></td>
<td>CDH 5.0, 5.1</td>
<td>cdh4.1</td>
</tr>
<tr>
<td>Hortonworks Data Platform</td>
<td>HDP 2.1, 2.2, 2.3, 2.4, 2.5</td>
<td>hdp2</td>
</tr>
<tr>
<td>MapR</td>
<td>MapR 4.x, MapR 5.x</td>
<td>gpmr-1.2</td>
</tr>
<tr>
<td>Apache Hadoop</td>
<td>2.x</td>
<td>hadoop2</td>
</tr>
</tbody>
</table>

**Note:** MapR requires the MapR client. For MapR 5.x, only TEXT and CSV are supported in the FORMAT clause of the CREATE EXTERNAL TABLE command.

**PXF Hadoop Distribution Compatibility**

The PXF Extension Framework supports Cloudera, Hortonworks Data Platform, and generic Apache Hadoop distributions.
Upgrading to Greenplum Database 5.1.0

The upgrade path supported for this release is Greenplum Database 5.0.0 to Greenplum Database 5.1.0. Upgrading a Greenplum Database 4.3.x release to Pivotal Greenplum 5.x is not supported. See Migrating Data to Pivotal Greenplum 5.x.

Prerequisites

Before starting the upgrade process, Pivotal recommends performing the following checks.

- Verify the health of the Greenplum Database host hardware, and that you verify that the hosts meet the requirements for running Greenplum Database. The Greenplum Database gpcheckperf utility can assist you in confirming the host requirements.

  Note: If you need to run the gpcheckcat utility, Pivotal recommends running it a few weeks before the upgrade and that you run gpcheckcat during a maintenance period. If necessary, you can resolve any issues found by the utility before the scheduled upgrade.

  The utility is in $GPHOME/bin. Pivotal recommends that Greenplum Database be in restricted mode when you run gpcheckcat utility. See the Greenplum Database Utility Guide for information about the gpcheckcat utility.

  If gpcheckcat reports catalog inconsistencies, you can run gpcheckcat with the -g option to generate SQL scripts to fix the inconsistencies.

  After you run the SQL scripts, run gpcheckcat again. You might need to repeat the process of running gpcheckcat and creating SQL scripts to ensure that there are no inconsistencies. Pivotal recommends that the SQL scripts generated by gpcheckcat be run on a quiescent system. The utility might report false alerts if there is activity on the system.

  Important: If the gpcheckcat utility reports errors, but does not generate a SQL script to fix the errors, contact Pivotal support. Information for contacting Pivotal Support is at https://support.pivotal.io.

- During the migration process from Greenplum Database 5.0.0, a backup is made of some files and directories in $MASTER_DATA_DIRECTORY. Pivotal recommends that files and directories that are not used by Greenplum Database be backed up, if necessary, and removed from the $MASTER_DATA_DIRECTORY before migration. For information about the Greenplum Database migration utilities, see the Greenplum Database Documentation.

For information about supported versions of Greenplum Database extensions, see Pivotal Greenplum Tools and Extensions Compatibility.

If you are utilizing Data Domain Boost, you have to re-enter your DD Boost credentials after upgrading from Greenplum Database 4.2.x.x to 4.3.x.x as follows:

```
gpcrondump --ddboost-host ddboost_hostname --ddboost-user ddboost_user
--ddboost-backupdir backup_directory
```

Note: If you do not reenter your login credentials after an upgrade, your backup will never start because the Greenplum Database cannot connect to the Data Domain system. You will receive an error advising you to check your login credentials.

Upgrading from 5.0.0 to 5.1.0

An upgrade from 5.0.0 to 5.1.0 involves stopping Greenplum Database, updating the Greenplum Database software binaries, upgrading and restarting Greenplum Database. If you are using Greenplum Database extension packages there are additional requirements. See Prerequisites in the previous section.
Note: If the Greenplum Command Center database `gpperfmon` is installed in your Greenplum Database system, the migration process changes the distribution key of the Greenplum Database `log_alert_*` tables to the `logtime` column. The redistribution of the table data might take some time the first time you start Greenplum Database after migration. The change occurs only the first time you start Greenplum Database after a migration.

1. Log in to your Greenplum Database master host as the Greenplum administrative user:

   ```
   $ su - gpadmin
   ```

2. Perform a smart shutdown of your current Greenplum Database 5.x system (there can be no active connections to the database). This example uses the `-a` option to disable confirmation prompts:

   ```
   $ gpstop -a
   ```

3. Run the binary installer for 5.1.0 on the Greenplum Database master host.

   When prompted, choose an installation location in the same base directory as your current installation. For example:

   ```
   /usr/local/greenplum-db-5.1.0
   ```

   If you install Greenplum Database with the rpm (as root), the installation directory is `/usr/local/greenplum-db-5.1.0`.

   For the rpm installation, update the permissions for the new installation. For example, run this command as root to change user and group of the installed files to `gpadmin`.

   ```
   # chown -R gpadmin:gpadmin /usr/local/greenplum*
   ```

4. If your Greenplum Database deployment uses LDAP authentication, manually edit the `/usr/local/greenplum-db/greenplum_path.sh` file to add the line:

   ```
   export LDAPCONF=/etc/openldap/ldap.conf
   ```

5. Edit the environment of the Greenplum Database superuser (`gpadmin`) and make sure you are sourcing the `greenplum_path.sh` file for the new installation. For example change the following line in `.bashrc` or your chosen profile file:

   ```
   source /usr/local/greenplum-db-5.0.0/greenplum_path.sh
   ```

   to:

   ```
   source /usr/local/greenplum-db-5.1.0/greenplum_path.sh
   ```

   Or if you are sourcing a symbolic link (`/usr/local/greenplum-db`) in your profile files, update the link to point to the newly installed version. For example:

   ```
   $ rm /usr/local/greenplum-db
   $ ln -s /usr/local/greenplum-db-5.1.0 /usr/local/greenplum-db
   ```

6. Source the environment file you just edited. For example:

   ```
   $ source ~/.bashrc
   ```

7. Run the `gpseginstall` utility to install the 5.1.0 binaries on all the segment hosts specified in the `hostfile`. For example:

   ```
   $ gpseginstall -f hostfile
   ```
8. Use the Greenplum Database `gppkg` utility to install Greenplum Database extensions. If you were previously using any Greenplum Database extensions such as pgcrypto, PL/R, PL/Java, PL/Perl, and PostGIS, download the corresponding packages from Pivotal Network, and install using this utility. See the Greenplum Database Documentation for `gppkg` usage details.

9. After all segment hosts have been upgraded, you can log in as the `gpadmin` user and restart your Greenplum Database system:

```bash
# su - gpadmin
$ gpstart
```

10. If you are utilizing Data Domain Boost, you have to re-enter your DD Boost credentials after upgrading from Greenplum Database 5.0.0 to 5.1.0 as follows:

```bash
gpcrondump --ddboost-host ddboost_hostname --ddboost-user ddboost_user
--ddboost-backupdir backup_directory
```

**Note:** If you do not reenter your login credentials after an upgrade, your backup will never start because the Greenplum Database cannot connect to the Data Domain system. You will receive an error advising you to check your login credentials.

**Troubleshooting a Failed Upgrade**

If you experience issues during the migration process and have active entitlements for Greenplum Database that were purchased through Pivotal, contact Pivotal Support. Information for contacting Pivotal Support is at [https://support.pivotal.io](https://support.pivotal.io).

Be prepared to provide the following information:

- A completed Upgrade Procedure.
- Log output from `gpcheckcat` (located in `~/gpAdminLogs`)
Migrating Data to Pivotal Greenplum 5.x

Upgrading a Pivotal Greenplum Database 4.x system directly to Pivotal Greenplum Database 5.x is not supported.

You can migrate existing data to Greenplum Database 5.x using standard backup and restore procedures (gpcrondump and gpdbrestore) or by using gptransfer if both clusters will run side-by-side.

Follow these general guidelines for migrating data:

• Make sure that you have a complete backup of all data in the Greenplum Database 4.3.x cluster, and that you can successfully restore the Greenplum Database 4.3.x cluster if necessary.

• You must install and initialize a new Greenplum Database 5.x cluster using the version 5.x gpinitsystem utility.

  Note: Unless you modify file locations manually, gpdbrestore only supports restoring data to a cluster that has an identical number of hosts and an identical number of segments per host, with each segment having the same content_id as the segment in the original cluster. If you initialize the Greenplum Database 5.x cluster using a configuration that is different from the version 4.3 cluster, then follow the steps outlined in Restoring to a Different Greenplum System Configuration to manually update the file locations.

• If you intend to install Greenplum Database 5.x on the same hardware as your 4.3.x system, you will need enough disk space to accommodate over 5 times the original data set (2 full copies of the primary and mirror data sets, plus the original backup data in ASCII format). Keep in mind that the ASCII backup data will require more disk space than the original data, which may be stored in compressed binary format. Offline backup solutions such as Dell EMC Data Domain or Veritas NetBackup can reduce the required disk space on each host.

• Use the version 5.x gpdbrestore utility to load the 4.3.x backup data into the new cluster.

• If the Greenplum Database 5.x cluster resides on separate hardware from the 4.3.x cluster, you can optionally use the version 5.x gptransfer utility to migrate the 4.3.x data. You must initiate the gptransfer operation from the version 5.x cluster, pulling the older data into the newer system.

On a Greenplum Database system with FIPS enabled, validating table data with MD5 (specifying the option --validate=md5) is not available. Use the option sha256 to validate table data.

Validating table data with SHA-256 (specifying the option --validate=sha256) requires the Greenplum Database pgcrypto extension. The extension is included with Pivotal Greenplum 5.x. The extension package must be installed on supported Pivotal Greenplum 4.3.x systems. Support for pgcrypto functions in a Greenplum 4.3.x database is not required.

• After migrating data you may need to modify SQL scripts, administration scripts, and user-defined functions as necessary to account for changes in Greenplum Database version 5.x. Look for Upgrade Action Required entries in the Pivotal Greenplum 5.0.0 Release Notes for features that may necessitate post-migration tasks.
Resolved Issues

The listed issues that are resolved in Pivotal Greenplum Database 5.1.0

For issues resolved in prior 5.x releases, refer to the corresponding release notes. Release notes are available from the Pivotal Greenplum page on Pivotal Network or on the Pivotal Greenplum Database documentation site at Release Notes.

29069 - Query Optimizer
GPORCA query optimization performance was poor for queries that contain an IN operator with a large number of values and the values require an implicit CAST.

This issue has been resolved. Query optimization performance has been improved for the specified type of queries.

29061 - gpcheckcat
In some cases, the Greenplum Database gpcheckcat utility returned an error stating that a value was out of range for type integer when the utility did not handle large OID values correctly (relfile nodes exceeding $2^{31}$, more than 2 billion).

This issue has been resolved. Now the utility handles large OID values correctly.

29042 - Query Planner
The legacy query optimizer generated a Greenplum Database PANIC when executing some aggregation queries. The queries contain column aliases with the same name as the table columns, the queries contain subqueries that reference the column alias, and grouping is applied on the column alias. In some cases, the legacy optimizer caused a PANIC when it attempted to generate a plan that contained a inconsistent target list for the aggregation plan.

This issue has been resolved. The legacy optimizer generates the correct query plan for the specified type of aggregation queries.

29039 - Query Optimizer
GPORCA generated a PANIC when a query attempted to use an index on a table column after the table was altered by dropping columns that are listed earlier in the table definition. For example, the index is defined on column 10, and columns 3 and 4 were deleted from the table. GPORCA did not correctly determine the index associated with the column.

This issue has been resolved. Now GPORCA correctly determines the column index in the specified situation.

29030 - Query Optimizer
For queries that contain a large number of joins, GPORCA spent a significant amount of time evaluating all possible multiple join combinations to determine the most efficient join tree to use in the query plan.

This issue has been resolved. The maximum number of join combinations that are evaluated has been reduced. See GPORCA Performance Enhancements.

29025 - Backup and Restore
Some restore operations performed by the Greenplum Database utility gpdbrestore that used DD Boost failed because of issues parsing DD Boost path information in the backup report file.

This issue has been resolved. Now the utility correctly parses DD Boost path information.

29009 - gprecoverseg
Running the Greenplum Database `gprecoverseg` with the option `-r` to rebalance segments did not return errors if the operation generated errors on some segments and completed successfully on other segments.

This issue has been resolved. Now if any errors occur during a rebalance operation, the user is notified and is referred to the logs for details.

29005 - Query Planner

For some `DELETE` or `UPDATE` commands that contain subqueries that reference the same table, the Greenplum Database legacy query optimizer generated a plan that did not correctly redistribute some rows to the appropriate segments. This caused a Greenplum Database PANIC when attempting to update or delete a row on an incorrect segment.

This issue has been resolved. Now the plan generated by the legacy optimizer correctly redistributes rows in the specified situation.

26993 - `gptransfer`

The Greenplum Database utility `gptransfer` failed intermittently when transferring large amounts of data. When exporting data, Greenplum Database did not properly check whether a write operation had completed. This might cause intermittent failures when exporting large amounts of data in a single operation.

This issue has been resolved. Checking the completion of write operations during data export has been improved.

151498144 - `gpcrondump`

When backing up and restoring database objects with the Greenplum Database utilities `gpcrondump` and `gpdbrestore` and specifying a schema level option such as the `gpcrondump` option `-s` or `-S`, functions (which are schema specific) were backed up and restored. However, only the PL/Java language was backed up and restored.

This issue has been resolved. Now for schema level back up and restore operations, procedural languages are included. See Changed Features.

151344210 - Query Planner

For an aggregation query, the Greenplum Database legacy query optimizer might return incorrect results when the aggregation is over a column, a `CAST` is defined on the column, and the original column (without the `CAST`) is in a `GROUP BY` clause. The legacy optimizer did not handle the `CAST` correctly in the query.

This issue has been resolved. Now the legacy optimizer handles the `CAST` correctly for the specified queries.

151341000 - Backup and Restore

In some cases for a Greenplum Database system with 10 or more segment instances, the Greenplum Database `gpdbrestore` utility did not perform a restore operation correctly. When restoring from a version 4.3.11.3 or earlier backup to version 4.3.12.0 or later, the utility did not handle the backup filenames correctly. The format of backup filenames changed in 4.3.12.0.

This issue has been resolved. Now the utility handles the backup filenames correctly in the specified situation.

151016845 - Query Optimizer

For queries involving catalog tables, GPORCA falls back to Greenplum Database legacy optimizer. The fallback event was set to the log severity level `LOG` and was captured in the Greenplum Database logs. In some cases, the messages bloated the Greenplum Database log files. For example, when a workload executed a large number of queries against catalog tables.
The issue has been resolved. Now the log level for the fallback event is DEBUG1. This level does not log the fallback events by default. The log level can be set to log the events for debugging purposes.

150988530 - Query Optimizer
In some cases, GPORCA generated incorrect results for a query when the query predicate contains a [NOT] EXISTS clause that contains both an aggregate function and a GROUP BY clause where a grouping column is an outer reference that is not in the aggregate function. GPORCA did not process the predicate correctly.

This issue has now been resolved. Now GPORCA handles the specified type of predicate correctly.

150906152 - Query Optimizer
For some queries that contain a correlated subquery and the subquery contains a window function, GPORCA generated a query plan that returned incorrect results. The plan incorrectly performed the join after the window function was applied.

This issue has been resolved. Now the plan generated by GPORCA correctly applies the window function before the join.

149513137 - gpstart
For Greenplum Database 5.0.0 systems installed on Linux hosts, the Greenplum system did not start when the host operating system had FIPS mode enabled. This issue was caused by the use of the Python MD5 hash algorithm by Greenplum Database.

This issue has been resolved. Greenplum Database 5.1.0 uses the Python SHA-256 hash algorithm.
# Known Issues and Limitations

Pivotal Greenplum 5.x has these limitations:

- Upgrading a Greenplum Database 4.3.x release to Pivotal Greenplum 5.x is not supported. See [Migrating Data to Pivotal Greenplum 5.x](#).
- Pivotal Greenplum 5.x is not yet certified for running on DCA systems. Contact your DCA representative for information about the availability of Greenplum Database 5.x support on the DCA.
- These features are considered work-in-progress and are experimental features. Pivotal does not support using experimental features in a production environment.
  - Greenplum Database resource groups
  - Recursive WITH Queries (Common Table Expressions)
  - Greenplum Database 4.3.x packages are not compatible with Pivotal Greenplum 5.x.

The following table lists key known issues in Pivotal Greenplum 5.x.

## Table 5: Key Known Issues in Pivotal Greenplum 5.x

<table>
<thead>
<tr>
<th>Issue</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
</table>
| 151468673 | Client and Loader Tools | For the Greenplum Database 5.0.0 Client and Loader tools for RHEL and SLES platforms, the default installation path is not valid. The default installation path contains a `:` that is not handled properly.  
*Workaround:* When installing a Client or Loader tool, specify a installation path, for example `/usr/local/greenplum-loaders-5.0.0`. Do not use the default installation directory. |
| 151135629 | COPY command           | When the `ON SEGMENT` clause is specified, the `COPY` command does not support specifying a `SELECT` statement in the `COPY TO` clause. For example, this command is not supported.  
```
COPY (SELECT * FROM testtbl) TO '/tmp/mytst<SEGID>' ON SEGMENT
```
| 29064   | Storage: DDL           | The `money` data type accepts out-of-range values as negative values, and no error message is displayed.  
*Workaround:* Use only in-range values for the `money` data type (32-bit for Greenplum Database 4.x, or 64-bit for Greenplum Database 5.x). Or, use an alternative data type such as `numeric` or `decimal`. |
<table>
<thead>
<tr>
<th>Issue</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3290</td>
<td>JSON</td>
<td>The <code>to_json()</code> function is not implemented as a callable function. Attempting to call the function results in an error. For example:</td>
</tr>
</tbody>
</table>

    tutorial=# select to_json('Fred said "Hi. "'::text);
    ERROR: function to_json(text) does not exist
    LINE 1: select to_json('Fred said "Hi."'::text);
    ^
    HINT: No function matches the given name and argument types. You might need to add explicit type casts.

**Workaround:** Greenplum Database invokes `to_json()` internally when casting to the `json` data type, so perform a cast instead. For example: SELECT '{"foo":"bar"}'::json; Greenplum Database also provides the `array_to_json()` and `row_to_json()` functions.
<table>
<thead>
<tr>
<th>Issue</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>148119917</td>
<td>Resource Groups (Experimental)</td>
<td>Testing of the experimental resource groups feature has found that a kernel panic can occur when using the default kernel in RHEL/ Centos system. The problem occurs due to a problem in the kernel cgroups implementation, and results in a kernel panic backtrace similar to:</td>
</tr>
</tbody>
</table>

```
[81375.325947] BUG: unable to handle kernel NULL pointer dereference at 0000000000000010
[81375.325986] IP: [fffffff812f94b1] rb_next+0x1/0x50 [81375.326014] PGD 0 [81375.326025] 
Oops: 0000 [#1] SMP [81375.326041] Modules
linked in: veth ipt масquerade_ipv4 iptable_nat nf_conntrack_ipv4 nf_defrag_ipv4 nf_nat_ipv4 xt_addrtype
  iptable_filter xt_conntrack nf_nat nf_conntrack bridge stp l1c intel_powerclamp
  coretemp
  intel_rapl dm_thin_pool dm_persistent_data
dm_bio_prison dm_bufio kvm_intel kvm crc32_pclmul
  ghash_clmulni_intel aesni_intel lrw
gf128mul glue_helper ablk_helper cryptd itcO_wdt
iTCO_vendor_support ses enclosure ipmi_ssi
  lpc ich sg sb_ecac md_core edac_core
  mei_me ipmi_si mei wmi ipmi_msghandler
  shpcp acpi_power_meter acpi_pad ip_tables xfs
  libcrc32c sd_mod crc_t10dif crrt10dif_gen
  generic mgqag200 syscopyarea sysfillrect
crrt10dif_pclmul
  sysimbgblt crrt10dif_common crc32c_intel
  drm_kms_helper ixgbe ttm mdio igb libhaci
drm ptp
  pps_core libdata dca i2c_algo_bit [81375.326369] i2c_core megaraid_sas dm_mirror
  dm_region_hash dm_log dm_mod [81375.326396] 
CPU: 17 PID: 0 Comm: swapper/I7 Not tainted
  3.10.0-327.el7.x86_64 #1 [81375.326422] Hardware name: Cisco Systems Inc
2.0.8b.0.080620151546 08/06/2015 [81375.326459] task:
  ffff88140ecec500 ti: ffff88140ed10000 task. 
ti: ffff88140ed10000 [81375.326485] RIP:
  0010:[<ffffff812f94b1>]
  [<ffffff812f94b1>] rb_next+0x1/0x50 [81375.326514] RSP:
  0018:ffff88140ed13e10 EFLAGS: 00010046
  [81375.326534] RAX: 0000000000000000 RBX:
  0000000000000000 RCX: 0000000000000000
  [81375.326559] RDX: ffff88282f1d4800 RSI:
  ffff88280bc0f140 RDI: 0000000000000000
  [81375.326584] RBP: ffff88140ed13e58 R8B:
  0000000000000000 R9: 0000000000000000
  [81375.326609] R10: 0000000000000000 R11:
  0000000000000000 R12: ffff88280be7000
  [81375.326634] R13: 0000000000000000 R14:
  0000000000000000 R15: 0000000000b6f979
  [81375.326659] FS: 0000000000000000(0000) |
```
<table>
<thead>
<tr>
<th>Issue</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Stack: [81375.326792] fff88140ed13e58 fff88282f1d4780 fff88282f1d4780 [81375.326826] fff88140ecec500 [81375.326895] Call Trace: [81375.326912]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[&lt;ffffffff810bf539&gt;] ? pick_next_task_fair +0x129/0x1d0 [81375.326940] [&lt;ffffffff8163a10a&gt;] __schedule+0x12a/0x900 [81375.326961] [&lt;ffffffff8163b9e9&gt;] schedule_preempt_disabled +0x29/0x70 [81375.326987] [&lt;ffffffff810d6244&gt;] cpu_startup_entry+0x184/0x290 [81375.327011] [&lt;ffffffff810475fa&gt;] start_secondary +0x1ba/0x230 [81375.327032] Code: e5 48 85 c0 75 07 eb 19 66 90 48 89 d0 48 8b 50 10 48 85 d2 75 f4 48 8b 50 08 48 85 d2 75 e6 5d c3 31 c0 5d c3 0f 1f 44 00 00 55 &lt;48&gt; 8b 17 48 89 e5 48 39 d7 74 3b 48 8b 47 08 48 85 c0 75 0e eb [81375.327157] RIP [&lt;ffffffff812f94b1&gt;] rb_next+0x1/0x50 [81375.327179] RSP &lt;fff88140ed13e10&gt; [81375.327192] CR2: 00000000000000010</td>
</tr>
</tbody>
</table>

**Workaround:** Upgrade to the latest-available kernel for your Red Hat or CentOS release to avoid the above system panic.
<table>
<thead>
<tr>
<th>Issue</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>149789783</td>
<td>Resource Groups</td>
<td>Significant Pivotal Greenplum performance degradation has been observed when enabling resource group-based workload management (an experimental feature) on RedHat 6.x, CentOS 6.x, and SuSE 11 systems. This issue is caused by a Linux cgrou...</td>
</tr>
</tbody>
</table>

When resource groups are enabled, there can be a delay of 1 second or longer when starting a transaction or a query. The delay is caused by a Linux cgrou kernel bug where a synchronization mechanism called synchronize_sched is abused when a process is attached to a cgroup. See [http://www.spinics.net/lists/cgroups/msg05708.html](http://www.spinics.net/lists/cgroups/msg05708.html) and [https://lkml.org/lkml/2013/1/14/97](https://lkml.org/lkml/2013/1/14/97) for more information.

The issue causes single attachment operations to take longer and also causes all concurrent attachments to be executed in sequence. For example, one process attachment could take about 0.01 second. When concurrently attaching 100 processes, the fastest process attachment takes 0.01 second and the slowest takes about 1 second. Pivotal Greenplum performs process attachments when a transaction or queries are started. So the performance degradation is dependent on concurrent started transactions or queries, and not related to concurrent running queries. Also Pivotal Greenplum has optimizations to bypass the rewriting when a QE is reused by multiple queries in the same session.

**Workaround:** If using Red Hat 6, update your kernel to version 2.6.32-696 or higher to reduce the performance impact. The bug does not affect CentOS 7.x and Red Hat 7.x systems. There is currently no kernel version that fixes this issue for SuSE 11 systems. Do not enable resource groups on affected Linux systems if the performance degradation is not acceptable.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>149970753</td>
<td>Resource Groups</td>
<td>The gpcrondump and gpdbrestore utilities do not backup or restore configuration information for the experimental Resource Groups feature.</td>
</tr>
</tbody>
</table>

**Workaround:** If you need to preserve your resource groups configuration, store the associated SQL commands in a separate file and manually apply the commands after restoring from a backup.
<table>
<thead>
<tr>
<th>Issue</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>150906510</td>
<td>Backup and Restore</td>
<td>Greenplum Database 4.3.15.0 and later backups contain the following line in the backup files:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SET gp_strict_xml_parse = false;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>However, Greenplum Database 5.0.0 does not have a parameter named gp_strict_xml_parse. When you restore the 4.3 backup set to the 5.0.0 cluster, you may see the warning:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[WARNING]:-gpdbrestore finished but ERRORS were found, please check the restore report file for details</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Also, the report file may contain the error:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ERROR: unrecognized configuration parameter &quot;gp_strict_xml_parse&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>These warnings and errors do not affect the restoration procedure, and can be ignored.</td>
</tr>
</tbody>
</table>
Chapter 5

Greenplum Database Installation Guide

Information about installing and configuring Greenplum Database software and configuring Greenplum Database host machines.
Introduction to Greenplum

High-level overview of the Greenplum Database system architecture.

Greenplum Database stores and processes large amounts of data by distributing the load across several servers or hosts. A logical database in Greenplum is an array of individual PostgreSQL databases working together to present a single database image. The master is the entry point to the Greenplum Database system. It is the database instance to which users connect and submit SQL statements. The master coordinates the workload across the other database instances in the system, called segments, which handle data processing and storage. The segments communicate with each other and the master over the interconnect, the networking layer of Greenplum Database.

Greenplum Database is a software-only solution; the hardware and database software are not coupled. Greenplum Database runs on a variety of commodity server platforms from Greenplum-certified hardware vendors. Performance depends on the hardware on which it is installed. Because the database is distributed across multiple machines in a Greenplum Database system, proper selection and configuration of hardware is vital to achieving the best possible performance.

This chapter describes the major components of a Greenplum Database system and the hardware considerations and concepts associated with each component: The Greenplum Master, The Segments and The Interconnect. Additionally, a system may have optional ETL Hosts for Data Loading and Greenplum Performance Monitoring for monitoring query workload and performance.

The Greenplum Master

The master is the entry point to the Greenplum Database system. It is the database server process that accepts client connections and processes the SQL commands that system users issue. Users connect to Greenplum Database through the master using a PostgreSQL-compatible client program such as psql or ODBC.

The master maintains the system catalog (a set of system tables that contain metadata about the Greenplum Database system itself), however the master does not contain any user data. Data resides only on the segments. The master authenticates client connections, processes incoming SQL commands, distributes the work load between segments, coordinates the results returned by each segment, and presents the final results to the client program.

Because the master does not contain any user data, it has very little disk load. The master needs a fast, dedicated CPU for data loading, connection handling, and query planning because extra space is often necessary for landing load files and backup files, especially in production environments. Customers may decide to also run ETL and reporting tools on the master, which requires more disk space and processing power.

Master Redundancy

You may optionally deploy a backup or mirror of the master instance. A backup master host serves as a warm standby if the primary master host becomes nonoperational. You can deploy the standby master on a designated redundant master host or on one of the segment hosts.

The standby master is kept up to date by a transaction log replication process, which runs on the standby master host and synchronizes the data between the primary and standby master hosts. If the primary master fails, the log replication process shuts down, and an administrator can activate the standby master in its place. When an the standby master is active, the replicated logs are used to reconstruct the state of the master host at the time of the last successfully committed transaction.
Since the master does not contain any user data, only the system catalog tables need to be synchronized between the primary and backup copies. When these tables are updated, changes automatically copy over to the standby master so it is always synchronized with the primary.

**Figure 1: Master Mirroring in Greenplum Database**

**The Segments**

In Greenplum Database, the *segments* are where data is stored and where most query processing occurs. User-defined tables and their indexes are distributed across the available segments in the Greenplum Database system; each segment contains a distinct portion of the data. Segment instances are the database server processes that serve segments. Users do not interact directly with the segments in a Greenplum Database system, but do so through the master.

In the reference Greenplum Database hardware configurations, the number of segment instances per segment host is determined by the number of effective CPUs or CPU core. For example, if your segment hosts have two dual-core processors, you may have two or four primary segments per host. If your segment hosts have three quad-core processors, you may have three, six or twelve segments per host. Performance testing will help decide the best number of segments for a chosen hardware platform.

**Segment Redundancy**

When you deploy your Greenplum Database system, you have the option to configure *mirror* segments. Mirror segments allow database queries to fail over to a backup segment if the primary segment becomes unavailable. Mirroring is a requirement for Pivotal-supported production Greenplum Database systems.

A mirror segment must always reside on a different host than its primary segment. Mirror segments can be arranged across the hosts in the system in one of two standard configurations, or in a custom configuration you design. The default configuration, called *group* mirroring, places the mirror segments for all primary segments on a host on one other host. Another option, called *spread* mirroring, spreads mirrors for each host's primary segments over the remaining hosts. Spread mirroring requires that there be more hosts in the system than there are primary segments on the host. On hosts with multiple network interfaces, the primary and mirror segments are distributed equally among the interfaces. **Figure 2: Data Mirroring in Greenplum Database** shows how table data is distributed across the segments when the default group mirroring option is configured.

**Figure 2: Data Mirroring in Greenplum Database**

**Segment Failover and Recovery**

When mirroring is enabled in a Greenplum Database system, the system automatically fails over to the mirror copy if a primary copy becomes unavailable. A Greenplum Database system can remain operational if a segment instance or host goes down only if all portions of data are available on the remaining active segments.

If the master cannot connect to a segment instance, it marks that segment instance as *invalid* in the Greenplum Database system catalog. The segment instance remains invalid and out of operation until an administrator brings that segment back online. An administrator can recover a failed segment while the system is up and running. The recovery process copies over only the changes that were missed while the segment was nonoperational.

If you do not have mirroring enabled and a segment becomes invalid, the system automatically shuts down. An administrator must recover all failed segments before operations can continue.
Example Segment Host Hardware Stack

Regardless of the hardware platform you choose, a production Greenplum Database processing node (a segment host) is typically configured as described in this section.

The segment hosts do the majority of database processing, so the segment host servers are configured in order to achieve the best performance possible from your Greenplum Database system. Greenplum Database's performance will be as fast as the slowest segment server in the array. Therefore, it is important to ensure that the underlying hardware and operating systems that are running Greenplum Database are all running at their optimal performance level. It is also advised that all segment hosts in a Greenplum Database array have identical hardware resources and configurations.

Segment hosts should also be dedicated to Greenplum Database operations only. To get the best query performance, you do not want Greenplum Database competing with other applications for machine or network resources.

The following diagram shows an example Greenplum Database segment host hardware stack. The number of effective CPUs on a host is the basis for determining how many primary Greenplum Database segment instances to deploy per segment host. This example shows a host with two effective CPUs (one dual-core CPU). Note that there is one primary segment instance (or primary/mirror pair if using mirroring) per CPU core.

Figure 3: Example Greenplum Database Segment Host Configuration

Example Segment Disk Layout

Each CPU is typically mapped to a logical disk. A logical disk consists of one primary file system (and optionally a mirror file system) accessing a pool of physical disks through an I/O channel or disk controller. The logical disk and file system are provided by the operating system. Most operating systems provide the ability for a logical disk drive to use groups of physical disks arranged in RAID arrays.

Figure 4: Logical Disk Layout in Greenplum Database

Depending on the hardware platform you choose, different RAID configurations offer different performance and capacity levels. Greenplum supports and certifies a number of reference hardware platforms and operating systems. Check with your sales account representative for the recommended configuration on your chosen platform.

The Interconnect

The *interconnect* is the networking layer of Greenplum Database. When a user connects to a database and issues a query, processes are created on each of the segments to handle the work of that query. The *interconnect* refers to the inter-process communication between the segments, as well as the network infrastructure on which this communication relies. The interconnect uses a standard 10 Gigabit Ethernet switching fabric.

By default, Greeplum Database interconnect uses UDP (User Datagram Protocol) with flow control for interconnect traffic to send messages over the network. The Greenplum software does the additional packet verification and checking not performed by UDP, so the reliability is equivalent to TCP (Transmission Control Protocol), and the performance and scalability exceeds that of TCP. For information about the types of interconnect supported by Greeplum Database, see server configuration parameter `gp_interconnect_type` in the *Greenplum Database Reference Guide*. 

Interconnect Redundancy

A highly available interconnect can be achieved by deploying dual 10 Gigabit Ethernet switches on your network, and redundant 10 Gigabit connections to the Greenplum Database master and segment host servers.

Network Interface Configuration

A segment host typically has multiple network interfaces designated to Greenplum interconnect traffic. The master host typically has additional external network interfaces in addition to the interfaces used for interconnect traffic.

Depending on the number of interfaces available, you will want to distribute interconnect network traffic across the number of available interfaces. This is done by assigning segment instances to a particular network interface and ensuring that the primary segments are evenly balanced over the number of available interfaces.

This is done by creating separate host address names for each network interface. For example, if a host has four network interfaces, then it would have four corresponding host addresses, each of which maps to one or more primary segment instances. The `/etc/hosts` file should be configured to contain not only the host name of each machine, but also all interface host addresses for all of the Greenplum Database hosts (master, standby master, segments, and ETL hosts).

With this configuration, the operating system automatically selects the best path to the destination. Greenplum Database automatically balances the network destinations to maximize parallelism.

Figure 5: Example Network Interface Architecture

Switch Configuration

When using multiple 10 Gigabit Ethernet switches within your Greenplum Database array, evenly divide the number of subnets between each switch. In this example configuration, if we had two switches, NICs 1 and 2 on each host would use switch 1 and NICs 3 and 4 on each host would use switch 2. For the master host, the host name bound to NIC 1 (and therefore using switch 1) is the effective master host name for the array. Therefore, if deploying a warm standby master for redundancy purposes, the standby master should map to a NIC that uses a different switch than the primary master.

Figure 6: Example Switch Configuration

ETL Hosts for Data Loading

Greenplum supports fast, parallel data loading with its external tables feature. By using external tables in conjunction with Greenplum Database's parallel file server (`gpfdist`), administrators can achieve maximum parallelism and load bandwidth from their Greenplum Database system. Many production systems deploy designated ETL servers for data loading purposes. These machines run the Greenplum parallel file server (`gpfdist`), but not Greenplum Database instances.

One advantage of using the `gpfdist` file server program is that it ensures that all of the segments in your Greenplum Database system are fully utilized when reading from external table data files.

The `gpfdist` program can serve data to the segment instances at an average rate of about 350 MB/s for delimited text formatted files and 200 MB/s for CSV formatted files. Therefore, you should consider the following options when running `gpfdist` in order to maximize the network bandwidth of your ETL systems:
• If your ETL machine is configured with multiple network interface cards (NICs) as described in *Network Interface Configuration*, run one instance of `gpfdist` on your ETL host and then define your external table definition so that the host name of each NIC is declared in the `LOCATION` clause (see `CREATE EXTERNAL TABLE` in the *Greenplum Database Reference Guide*). This allows network traffic between your Greenplum segment hosts and your ETL host to use all NICs simultaneously.

**Figure 7: External Table Using Single gpfdist Instance with Multiple NICs**

• Run multiple `gpfdist` instances on your ETL host and divide your external data files equally between each instance. For example, if you have an ETL system with two network interface cards (NICs), then you could run two `gpfdist` instances on that machine to maximize your load performance. You would then divide the external table data files evenly between the two `gpfdist` programs.

**Figure 8: External Tables Using Multiple gpfdist Instances with Multiple NICs**

**Greenplum Performance Monitoring**

Greenplum Database includes a dedicated system monitoring and management database, named `gpperfmon`, that administrators can install and enable. When this database is enabled, data collection agents on each segment host collect query status and system metrics. At regular intervals (typically every 15 seconds), an agent on the Greenplum master requests the data from the segment agents and updates the `gpperfmon` database. Users can query the `gpperfmon` database to see the stored query and system metrics. For more information see the "gpperfmon Database Reference" in the *Greenplum Database Reference Guide*.

Greenplum Command Center is an optional web-based performance monitoring and management tool for Greenplum Database, based on the `gpperfmon` database. Administrators can install Command Center separately from Greenplum Database.

**Figure 9: Greenplum Performance Monitoring Architecture**
Estimating Storage Capacity

To estimate how much data your Greenplum Database system can accommodate, use these measurements as guidelines. Also keep in mind that you may want to have extra space for landing backup files and data load files on each segment host.

Calculating Usable Disk Capacity

To calculate how much data a Greenplum Database system can hold, you have to calculate the usable disk capacity per segment host and then multiply that by the number of segment hosts in your Greenplum Database array. Start with the raw capacity of the physical disks on a segment host that are available for data storage \((raw\_capacity)\), which is:

\[
disk\_size * number\_of\_disks
\]

Account for file system formatting overhead (roughly 10 percent) and the RAID level you are using. For example, if using RAID-10, the calculation would be:

\[
(raw\_capacity * 0.9) / 2 = formatted\_disk\_space
\]

For optimal performance, do not completely fill your disks to capacity, but run at 70% or lower. So with this in mind, calculate the usable disk space as follows:

\[
formatted\_disk\_space * 0.7 = usable\_disk\_space
\]

Once you have formatted RAID disk arrays and accounted for the maximum recommended capacity \((usable\_disk\_space)\), you will need to calculate how much storage is actually available for user data \((U)\). If using Greenplum Database mirrors for data redundancy, this would then double the size of your user data \((2 * U)\). Greenplum Database also requires some space be reserved as a working area for active queries. The work space should be approximately one third the size of your user data \((work\_space = U/3)\):

With mirrors: \((2 * U) + U/3 = usable\_disk\_space\)

Without mirrors: \(U + U/3 = usable\_disk\_space\)

Guidelines for temporary file space and user data space assume a typical analytic workload. Highly concurrent workloads or workloads with queries that require very large amounts of temporary space can benefit from reserving a larger working area. Typically, overall system throughput can be increased while decreasing work area usage through proper workload management. Additionally, temporary space and user space can be isolated from each other by specifying that they reside on different tablespaces.

In the *Greenplum Database Administrator Guide*, see these topics:

- "Managing Workload and Resources" for information about workload management
- "Creating and Managing Tablespaces" for information about moving the location of temporary files
- "Monitoring System State" for information about monitoring Greenplum Database disk space usage

Calculating User Data Size

As with all databases, the size of your raw data will be slightly larger once it is loaded into the database. On average, raw data will be about 1.4 times larger on disk after it is loaded into the database, but could be smaller or larger depending on the data types you are using, table storage type, in-database compression, and so on.
• **Page Overhead** - When your data is loaded into Greenplum Database, it is divided into pages of 32KB each. Each page has 20 bytes of page overhead.

• **Row Overhead** - In a regular 'heap' storage table, each row of data has 24 bytes of row overhead. An 'append-optimized' storage table has only 4 bytes of row overhead.

• **Attribute Overhead** - For the data values itself, the size associated with each attribute value is dependent upon the data type chosen. As a general rule, you want to use the smallest data type possible to store your data (assuming you know the possible values a column will have).

• **Indexes** - In Greenplum Database, indexes are distributed across the segment hosts as is table data. The default index type in Greenplum Database is B-tree. Because index size depends on the number of unique values in the index and the data to be inserted, precalculating the exact size of an index is impossible. However, you can roughly estimate the size of an index using these formulas.

\[
\text{B-tree: unique_values} \times (\text{data_type_size} + 24 \text{ bytes})
\]

\[
\text{Bitmap: (unique_values} \times \text{number_of_rows} \times 1 \text{ bit} \times \text{compression_ratio} / 8) + (\text{unique_values} \times 32)
\]

**Calculating Space Requirements for Metadata and Logs**

On each segment host, you will also want to account for space for Greenplum Database log files and metadata:

• **System Metadata** — For each Greenplum Database segment instance (primary or mirror) or master instance running on a host, estimate approximately 20 MB for the system catalogs and metadata.

• **Write Ahead Log** — For each Greenplum Database segment (primary or mirror) or master instance running on a host, allocate space for the write ahead log (WAL). The WAL is divided into segment files of 64 MB each. At most, the number of WAL files will be:

\[
2 \times \text{checkpoint_segments} + 1
\]

You can use this to estimate space requirements for WAL. The default `checkpoint_segments` setting for a Greenplum Database instance is 8, meaning 1088 MB WAL space allocated for each segment or master instance on a host.

• **Greenplum Database Log Files** — Each segment instance and the master instance generates database log files, which will grow over time. Sufficient space should be allocated for these log files, and some type of log rotation facility should be used to ensure that log files do not grow too large.

• **Command Center Data** — The data collection agents utilized by Command Center run on the same set of hosts as your Greenplum Database instance and utilize the system resources of those hosts. The resource consumption of the data collection agent processes on these hosts is minimal and should not significantly impact database performance. Historical data collected by the collection agents is stored in its own Command Center database (named `gpperfmon`) within your Greenplum Database system. Collected data is distributed just like regular database data, so you will need to account for disk space in the data directory locations of your Greenplum segment instances. The amount of space required depends on the amount of historical data you would like to keep. Historical data is not automatically truncated. Database administrators must set up a truncation policy to maintain the size of the Command Center database.
Configuring Your Systems and Installing Greenplum

Describes how to prepare your operating system environment for Greenplum, and install the Greenplum Database software binaries on all of the hosts that will comprise your Greenplum Database system.

Perform the following tasks in order:

1. Make sure your systems meet the System Requirements
2. Setting the Greenplum Recommended OS Parameters
3. (master only) Installing the Binary Distribution
4. Installing and Configuring Greenplum on all Hosts
5. (Optional) Installing Oracle Compatibility Functions
6. (Optional) Installing Optional Modules
7. (Optional) Installing Greenplum Database Extensions
8. Creating the Data Storage Areas
9. Synchronizing System Clocks
10. Next Steps

Unless noted, these tasks should be performed for all hosts in your Greenplum Database array (master, standby master and segments).

You can install and configure Greenplum Database on virtual servers provided by the Amazon Elastic Compute Cloud (Amazon EC2) web service. For information about using Greenplum Database in an Amazon EC2 environment, see Amazon EC2 Configuration.

Important: When data loss is not acceptable for a Pivotal Greenplum Database cluster, master and segment mirroring must be enabled in order for the cluster to be supported by Pivotal. Without mirroring, system and data availability is not guaranteed, Pivotal will make best efforts to restore a cluster in this case. For information about master and segment mirroring, see About Redundancy and Failover in the Greenplum Database Administrator Guide.

Note: For information about upgrading Pivotal Greenplum Database from a previous version, see the Greenplum Database Release Notes for the release that you are installing.

System Requirements

The following table lists minimum recommended specifications for servers intended to support Greenplum Database on Linux systems in a production environment. All servers in your Greenplum Database system must have the same hardware and software configuration. Greenplum also provides hardware build guides for its certified hardware platforms. It is recommended that you work with a Greenplum Systems Engineer to review your anticipated environment to ensure an appropriate hardware configuration for Greenplum Database.

Table 6: System Prerequisites for Greenplum Database 5.0

<table>
<thead>
<tr>
<th>Operating System</th>
<th>SUSE Linux Enterprise Server 64-bit 11 SP4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CentOS 64-bit 6.x or 7.x</td>
</tr>
<tr>
<td></td>
<td>Red Hat Enterprise Linux (RHEL) 64-bit 6.x or 7.x</td>
</tr>
</tbody>
</table>

Note: See the Greenplum Database Release Notes for current supported platform information.
### File Systems
- xfs required for data storage on SUSE Linux and Red Hat (ext3 supported for root file system)

### Minimum CPU
- Pentium Pro compatible (P3/Athlon and above)

### Minimum Memory
- 16 GB RAM per server

### Disk Requirements
- 150MB per host for Greenplum installation
- Approximately 300MB per segment instance for meta data
- Appropriate free space for data with disks at no more than 70% capacity
- High-speed, local storage

### Network Requirements
- 10 Gigabit Ethernet within the array
- Dedicated, non-blocking switch
- NIC bonding is recommended when multiple interfaces are present

### Software and Utilities
- zlib compression libraries
- bash shell
- GNU tars
- GNU zip
- GNU sed (used by Greenplum Database gpinitsystem)
- perl

**Important:** SSL is supported only on the Greenplum Database master host system. It is not supported on the segment host systems.

**Important:** For all Greenplum Database host systems, the SELinux must be disabled. You should also disable firewall software such as `iptables` (on systems such as RHEL 6.x and CentOS 6.x) or `firewalld` (on systems such as RHEL 7.x and CentOS 7.x). You can enable firewall software if it is required for security purposes. For information about enabling and configuring `iptables`, see [Enabling iptables](#). For information about enabling and configuring `firewalld`, see the list of instructions here:

- This command checks the status of SELinux when run as root:

  ```bash
  # sestatus
  SELinuxstatus: disabled
  ```

  You can disable SELinux by editing the `/etc/selinux/config` file. As root, change the value of the SELINUX parameter in the `config` file and reboot the system:

  ```bash
  SELinux=disabled
  ```

  For information about disabling firewall software, see the documentation for the firewall or your operating system. For information about disabling SELinux, see the SELinux documentation.

- This command checks the status of `iptables` when run as root:

  ```bash
  # /sbin/chkconfig --list iptables
  ```
This is the output if `iptables` is disabled.

```
iptables 0:off 1:off 2:off 3:off 4:off 5:off 6:off
```

One method of disabling `iptables` is to become root, run this command, and then reboot the system:

```
/sbin/chkconfig iptables off
```

- This command checks the status of `firewalld` when run as root:

```
# systemctl status firewalld
```

This is the output if `firewalld` is disabled.

```
* firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; vendor preset: enabled)
   Active: inactive (dead)
```

These commands disable `firewalld` when run as root:

```
# systemctl stop firewalld.service
# systemctl disable firewalld.service
```

## Setting the Greenplum Recommended OS Parameters

Greenplum requires the certain Linux operating system (OS) parameters be set on all hosts in your Greenplum Database system (masters and segments).

In general, the following categories of system parameters need to be altered:

- **Shared Memory** - A Greenplum Database instance will not work unless the shared memory segment for your kernel is properly sized. Most default OS installations have the shared memory values set too low for Greenplum Database. On Linux systems, you must also disable the OOM (out of memory) killer. For information about Greenplum Database shared memory requirements, see the Greenplum Database server configuration parameter `shared_buffers` in the Greenplum Database Reference Guide.

- **Network** - On high-volume Greenplum Database systems, certain network-related tuning parameters must be set to optimize network connections made by the Greenplum interconnect.

- **User Limits** - User limits control the resources available to processes started by a user’s shell. Greenplum Database requires a higher limit on the allowed number of file descriptors that a single process can have open. The default settings may cause some Greenplum Database queries to fail because they will run out of file descriptors needed to process the query.

### Linux System Settings

- **Edit the `/etc/hosts` file and make sure that it includes the host names and all interface address names for every machine participating in your Greenplum Database system.**

- **Set the following parameters in the `/etc/sysctl.conf` file and reboot:**

  ```
  kernel.shmmmax = 500000000
  kernel.shmni = 4096
  kernel.shmmax = 4000000000
  kernel.sem = 250 512000 100 2048
  kernel.sysrq = 1
  kernel.core_uses_pid = 1
  kernel.msgmnb = 65536
  ```
kernel.msgmax = 65536
kernel.msgmni = 2048
net.ipv4.tcp_syncookies = 1
net.ipv4.conf.default.accept_source_route = 0
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_max_syn_backlog = 4096
net.ipv4.conf.all.arp_filter = 1
net.ipv4.ip_local_port_range = 10000 65535
net.core.netdev_max_backlog = 10000
net.core.rmem_max = 2097152
net.core.wmem_max = 2097152
vm.overcommit_memory = 2

**Note:** When `vm.overcommit_memory` is 2, you specify a value for `vm.overcommit_ratio`. For information about calculating the value for `vm.overcommit_ratio`, see the Greenplum Database server configuration parameter `gp_vmem_protect_limit` in the *Greenplum Database Reference Guide*.

To avoid port conflicts between Greenplum Database and other applications, when initializing Greenplum Database, do not specify Greenplum Database ports in the range specified by the operating system parameter `net.ipv4.ip_local_port_range`. For example, if `net.ipv4.ip_local_port_range = 10000 65535`, you could set the Greenplum Database base port numbers to these values.

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

For information about port ranges that are used by Greenplum Database, see `gpinitdb`.

- Set the following parameters in the `/etc/security/limits.conf` file:

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

  For Red Hat Enterprise Linux (RHEL) 6.x and CentOS 6.x, parameter values in the `/etc/security/limits.d/90-nproc.conf` file override the values in the `limits.conf` file. If a parameter value is set in both `conf` files, ensure that the parameter is set properly in the `90-nproc.conf` file. The Linux module `pam_limits` sets user limits by reading the values from the `limits.conf` file and then from the `90-nproc.conf` file. For information about PAM and user limits, see the documentation on PAM and `pam_limits`.

- XFS is the preferred file system on Linux platforms for data storage. The following XFS mount options are recommended:

  `rw,nodev,noatime,nobarrier,inode64`

  See the manual page (man) for the `mount` command for more information about using that command (`man mount` opens the man page).

  The XFS options can also be set in the `/etc/fstab` file. This example entry from an `fstab` file specifies the XFS options.

  `/dev/data /data xfs nodev,noatime,nobarrier,inode64 0 0`

- Each disk device file should have a read-ahead (`blockdev`) value of 16384.
To verify the read-ahead value of a disk device:

```
# /sbin/blockdev --getra devname
```

For example:

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

To set blockdev (read-ahead) on a device:

```
# /sbin/blockdev --setra bytes devname
```

For example:

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

See the manual page (man) for the `blockdev` command for more information about using that command (`man blockdev` opens the man page).

- The Linux disk I/O scheduler for disk access supports different policies, such as CFQ, AS, and deadline.

The `deadline` scheduler option is recommended. To specify a scheduler until the next system reboot, run the following:

```
# echo schedulername > /sys/block/devname/queue/scheduler
```

For example:

```
# echo deadline > /sys/block/sbd/queue/scheduler
```

You can specify the I/O scheduler at boot time with the `elevator` kernel parameter. Add the parameter `elevator=deadline` to the `kernel` command in the file `/boot/grub/grub.conf`, the GRUB boot loader configuration file. This is an example `kernel` command from a `grub.conf` file on RHEL 6.x or CentOS 6.x. The command is on multiple lines for readability.

```
kernel /vmlinuz-2.6.18-274.3.1.el5 ro root=LABEL=/
    elevator=deadline
    crashkernel=128M@16M
core=ttys0,115200 panic=30 transparent_hugepage=never
    initrd /initrd-2.6.18-274.3.1.el5.img
```

To specify the I/O scheduler at boot time on systems that use `grub2` such as RHEL 7.x or CentOS 7.x, use the system utility `grubby`. This command adds the parameter when run as root.

```
# grubby --update-kernel=ALL --args="elevator=deadline"
```

After adding the parameter, reboot the system.

This `grubby` command displays kernel parameter settings.

```
# grubby --info=ALL
```

For more information about the `grubby` utility, see your operating system documentation. If the `grubby` command does not update the kernels, see the `Note` at the end of the section.

- Disable Transparent Huge Pages (THP). RHEL 6.0 or higher enables THP by default. THP degrades Greenplum Database performance. One way to disable THP on RHEL 6.x is by adding the parameter `transparent_hugepage=never` to the `kernel` command in the file `/boot/grub/grub.conf`, the
GRUB boot loader configuration file. This is an example kernel command from a grub.conf file. The command is on multiple lines for readability:

```
kernel /vmlinuz-2.6.18-274.3.1.el5 ro root=LABEL=/
elevator=deadline crashkernel=128M@16M quiet console=tty1
console=ttyS1,115200 panic=30 transparent_hugepage=never
initrd /initrd-2.6.18-274.3.1.el5.img
```

On systems that use grub2 such as RHEL 7.x or CentOS 7.x, use the system utility grubby. This command adds the parameter when run as root.

```
# grubby --update-kernel=ALL --args="transparent_hugepage=never"
```

After adding the parameter, reboot the system.

This cat command checks the state of THP. The output indicates that THP is disabled.

```
$ cat /sys/kernel/mm/*/transparent_hugepage/enabled
always [never]
```

For more information about Transparent Huge Pages or the grubby utility, see your operating system documentation. If the grubby command does not update the kernels, see the Note at the end of the section.

- Disable IPC object removal for RHEL 7.2 or CentOS 7.2. The default systemd setting RemoveIPC=yes removes IPC connections when non-system users log out. This causes the Greenplum Database utility gpinsitsystem to fail with semaphore errors. Perform one of the following to avoid this issue.

  - Create the gpadmin user as a system account. For the useradd command, the -r option creates a user as a system user and the -m option creates a home directory for the user.

    **Note:** When you run the gpseginstall utility as the root user to install Greenplum Database on host systems, the utility creates the gpadmin user as a system account on the hosts. See Installing and Configuring Greenplum on all Hosts

  - Disable RemoveIPC. Set this parameter in /etc/systemd/logind.conf on the Greenplum Database host systems.

    RemoveIPC=no

    The setting takes effect after restarting the systemd-login service or rebooting the system. To restart the service, run this command as the root user.

    service systemd-logind restart

- On some SUSE Linux Enterprise Server platforms, the Greenplum Database utility gpssh fails with the error message out of pty devices. A workaround is to add Greenplum Database users, for example gpadmin, to the tty group. On SUSE systems, tty is required to run gpssh

  **Note:** If the grubby command does not update the kernels of a RHEL 7.x or CentOS 7.x system, you can manually update all kernels on the system. For example, to add the parameter transparent_hugepage=never to all kernels on a system.

  1. Add the parameter to the GRUB_CMDLINE_LINUX line in the file parameter in /etc/default/grub.

```
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*\$,,g' /etc/system-release)"
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
```
2. As root, run the grub2-mkconfig command to update the kernels.

   # grub2-mkconfig -o /boot/grub2/grub.cfg

3. Reboot the system.

**Installing the Greenplum Database Software**

To configure your systems for Greenplum Database, the master host machine required the utilities found in `$GPHOME/bin` of the Greenplum Database software installation. Pivotal distributes the Greenplum Database software both as a downloadable RPM file and as a binary installer. You can use either distribution method to install the software on the master host machine.

If you do not have root access on the master host machine, run the binary installer as the `gpadmin` user and install the software into a directory in which you have write permission.

**Installing the RPM Distribution**

1. Log in as root on the machine that will become the Greenplum Database master host.
2. Download or copy the RPM distribution file to the master host machine. The RPM distribution filename has the format `greenplum-db-<version>-<platform>.rpm` where `<platform>` is either `RHEL6-x86_64` (Red Hat 64-bit) or `SuSE11-x86_64` (SuSe Linux 64 bit).
3. Install the local RPM file:

   ```
   # rpm -Uvh ./greenplum-db-<version>-<platform>.rpm
   Preparing...                ###########################################
   [100%]
   1:greenplum-db           ###########################################
   [100%]
   ```

   The RPM installation copies the Greenplum Database software into a version-specific directory, `/usr/local/greenplum-db-<version>`.

4. Change the ownership and group of the installed files to `gpadmin`:

   ```
   # chown -R gpadmin /usr/local/greenplum*
   # chgrp -R gpadmin /usr/local/greenplum*
   ```

5. To perform additional required system configuration tasks and to install Greenplum Database on other hosts, go to the next task *Installing and Configuring Greenplum on all Hosts.*

**Installing the Binary Distribution**

1. Log in as root on the machine that will become the Greenplum Database master host.

   If you do not have root access on the master host machine, run the binary installer as the `gpadmin` user and install the software into a directory in which you have write permission.

2. Download or copy the Binary Installation distribution file to the master host machine. The Binary Installer distribution filename has the format `greenplum-db-<version>-<platform>.zip` where `<platform>` is either `RHEL6-x86_64` (Red Hat 64-bit) or `SuSE11-x86_64` (SuSe Linux 64 bit).
3. Unzip the installer file:

   ```
   # unzip greenplum-db-<version>-<platform>.zip
   ```
4. Launch the installer using `bash`:

```
# /bin/bash greenplum-db-<version>-<platform>.bin
```

5. The installer prompts you to accept the Greenplum Database license agreement. Type `yes` to accept the license agreement.

6. The installer prompts you to provide an installation path. Press `ENTER` to accept the default install path (`/usr/local/greenplum-db-<version>`), or enter an absolute path to a custom install location. You must have write permission to the location you specify.

7. The installer installs the Greenplum Database software and creates a `greenplum-db` symbolic link one directory level above the version-specific installation directory. The symbolic link is used to facilitate patch maintenance and upgrades between versions. The installed location is referred to as `$GPHOME`.

8. If you installed as `root`, change the ownership and group of the installed files to `gpadmin`:

```
# chown -R gpadmin /usr/local/greenplum*
# chgrp -R gpadmin /usr/local/greenplum*
```

9. To perform additional required system configuration tasks and to install Greenplum Database on other hosts, go to the next task, `Installing and Configuring Greenplum on all Hosts`.

### About Your Greenplum Database Installation

- **greenplum_path.sh** — This file contains the environment variables for Greenplum Database. See Setting Greenplum Environment Variables.
- **bin** — This directory contains the Greenplum Database management utilities. This directory also contains the PostgreSQL client and server programs, most of which are also used in Greenplum Database.
- **docs/cli_help** — This directory contains help files for Greenplum Database command-line utilities.
- **docs/cli_help/gpconfigs** — This directory contains sample `gpinitdb` configuration files and host files that can be modified and used when installing and initializing a Greenplum Database system.
- **docs/javadoc** — This directory contains javadocs for the gNet extension (gphdfs protocol). The jar files for the gNet extension are installed in the `$GPHOME/lib/hadoop` directory.
- **etc** — Sample configuration file for OpenSSL and a sample configuration file to be used with the `gpcheck` management utility.
- **ext** — Bundled programs (such as Python) used by some Greenplum Database utilities.
- **include** — The C header files for Greenplum Database.
- **lib** — Greenplum Database and PostgreSQL library files.
- **sbin** — Supporting/Internal scripts and programs.
- **share** — Shared files for Greenplum Database.

### Installing and Configuring Greenplum on all Hosts

When run as `root`, `gpseginstall` copies the Greenplum Database installation from the current host and installs it on a list of specified hosts, creates the Greenplum system user (`gpadmin`), sets the system user's password (default is `changeme`), sets the ownership of the Greenplum Database installation directory, and exchanges ssh keys between all specified host address names (both as `root` and as the specified system user).

### About gpadmin

When a Greenplum Database system is first initialized, the system contains one predefined superuser role (also referred to as the system user), `gpadmin`. This is the user who owns and administers the Greenplum Database.
Note: If you are setting up a single node system, you can still use `gpseginstall` to perform the required system configuration tasks on the current host. In this case, the `hostfile_exkeys` should have only the current host name.

To install and configure Greenplum Database on all specified hosts

1. Log in to the master host as `root`:

   $ su -

2. Source the path file from your master host's Greenplum Database installation directory:

   # source /usr/local/greenplum-db/greenplum_path.sh

3. Create a file called `hostfile_exkeys` that has the machine configured host names and host addresses (interface names) for each host in your Greenplum system (master, standby master and segments). Make sure there are no blank lines or extra spaces. For example, if you have a master, standby master and three segments with two unbonded network interfaces per host, your file would look something like this:

   - 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

   Check the `/etc/hosts` file on your systems for the correct host names to use for your environment.

   The Greenplum Database segment host naming convention is `sdwN` where `sdw` is a prefix and `N` is an integer. For example, segment host names would be `sdw1`, `sdw2` and so on. NIC bonding is recommended for hosts with multiple interfaces, but when the interfaces are not bonded, the convention is to append a dash (–) and number to the host name. For example, `sdw1-1` and `sdw1-2` are the two interface names for host `sdw1`.

4. Run the `gpseginstall` utility referencing the `hostfile_exkeys` file you just created. This example runs the utility as `root`. The utility creates the Greenplum system user `gpadmin` on all hosts and sets the password as `changeme` for that user on all hosts.

   # gpseginstall -f hostfile_exkeys

   Use the `-u` and `-p` options to specify a different system user and password. See `gpseginstall` for option information and running the utility as a non-root user.

   Recommended security best practices:
   - Do not use the default password option for production environments.
   - Change the password immediately after installation.
Confirming Your Installation

To make sure the Greenplum software was installed and configured correctly, run the following confirmation steps from your Greenplum master host. If necessary, correct any problems before continuing on to the next task.

1. Log in to the master host as gpadmin:

   ```bash
   $ su - gpadmin
   ```

2. Source the path file from Greenplum Database installation directory:

   ```bash
   # source /usr/local/greenplum-db/greenplum_path.sh
   ```

3. Use the gpssh utility to see if you can login to all hosts without a password prompt, and to confirm that the Greenplum software was installed on all hosts. Use the hostfile_exkeys file you used for installation. For example:

   ```bash
   $ gpssh -f hostfile_exkeys -e ls -l $GPHOME
   ```

   If the installation was successful, you should be able to log in to all hosts without a password prompt. All hosts should show that they have the same contents in their installation directories, and that the directories are owned by the gpadmin user.

   If you are prompted for a password, run the following command to redo the ssh key exchange:

   ```bash
   $ gpssh-exkeys -f hostfile_exkeys
   ```

Installing Oracle Compatibility Functions

Optional. Many Oracle Compatibility SQL functions are available in Greenplum Database. These functions target PostgreSQL.

Before using any Oracle Compatibility Functions, you need to run the installation script `$GPHOME/share/postgresql/contrib/orafunc.sql` once for each database. For example, to install the functions in database testdb, use the command

```bash
$ psql -d testdb -f $GPHOME/share/postgresql/contrib/orafunc.sql
```

To uninstall Oracle Compatibility Functions, use the script:

```bash
$GPHOME/share/postgresql/contrib/uninstall_orafunc.sql
```

Note: The following functions are available by default and can be accessed without running the Oracle Compatibility installer: `sinh, tanh, cosh` and `decode`.

For more information about Greenplum's Oracle compatibility functions, see "Oracle Compatibility Functions" in the Greenplum Database Utility Guide.

Installing Optional Modules

dblink

The PostgreSQL dblink module provides simple connections to other databases either on the same database host, or on a remote host. Greenplum Database provides dblink support for database users to perform short ad hoc queries in other databases. It is not intended as a replacement for external tables or administrative tools such as gptransfer.
Before you can use dblink functions, run the installation script `$GPHOME/share/postgresql/contrib/dblink.sql` in each database where you want the ability to query other databases:

$$psql -d testdb -f $GPHOME/share/postgresql/contrib/dblink.sql$$

See `dblink Functions` for basic information about using dblink to query other databases. See `dblink` in the PostgreSQL documentation for more information about individual functions.

**pgcrypto**

Greenplum Database includes an optional package of encryption/decryption functions called `pgcrypto`. The `pgcrypto` functions allow database administrators to store certain columns of data in encrypted form. This adds an extra layer of protection for sensitive data, as data stored in Greenplum Database in encrypted form cannot be read by anyone who does not have the encryption key, nor can it be read directly from the disks.

**Note:** The `pgcrypto` functions run inside the database server, which means that all the data and passwords move between `pgcrypto` and the client application in clear-text. For optimal security, consider also using SSL connections between the client and the Greenplum master server.

Before you can use `pgcrypto` functions, run the installation script `$GPHOME/share/postgresql/contrib/pgcrypto.sql` in each database where you want the ability to query other databases:

$$psql -d testdb -f $GPHOME/share/postgresql/contrib/pgcrypto.sql$$

See `pgcrypto` in the PostgreSQL documentation for more information about individual functions.

### Installing Greenplum Database Extensions

**Optional.** Use the Greenplum package manager (`gppkg`) to install Greenplum Database extensions such as PL/Java, PL/R, PostGIS, and MADlib, along with their dependencies, across an entire cluster. The package manager also integrates with existing scripts so that any packages are automatically installed on any new hosts introduced into the system following cluster expansion or segment host recovery.

See `gppkg` for more information, including usage.

Extension packages can be downloaded from the Greenplum Database page on Pivotal Network. The extension documentation in the Greenplum Database Reference Guide contains information about installing extension packages and using extensions.

- Greenplum PL/R Language Extension
- Greenplum PL/Java Language Extension
- Greenplum MADlib Extension for Analytics
- Greenplum PostGIS Extension

**Important:** If you intend to use an extension package with Greenplum Database 5.0.0 you must install and use a Greenplum Database extension package (`gppkg` files and contrib modules) that is built for Greenplum Database 5.0.0. Any custom modules that were used with earlier versions must be rebuilt for use with Greenplum Database 5.0.0.

### Creating the Data Storage Areas

Every Greenplum Database master and segment instance has a designated storage area on disk that is called the **data directory location**. This is the file system location where the directories that store segment instance data will be created. The master host needs a data storage location for the master data directory. Each segment host needs a data directory storage location for its primary segments, and another for its mirror segments.
Creating a Data Storage Area on the Master Host

A data storage area is required on the Greenplum Database master host to store Greenplum Database system data such as catalog data and other system metadata.

To create the data directory location on the master

The data directory location on the master is different than those on the segments. The master does not store any user data, only the system catalog tables and system metadata are stored on the master instance, therefore you do not need to designate as much storage space as on the segments.

1. Create or choose a directory that will serve as your master data storage area. This directory should have sufficient disk space for your data and be owned by the gpadmin user and group. For example, run the following commands as root:

   ```bash
   # mkdir /data/master
   ``

2. Change ownership of this directory to the gpadmin user. For example:

   ```bash
   # chown gpadmin /data/master
   ``

3. Using gpssh, create the master data directory location on your standby master as well. For example:

   ```bash
   # source /usr/local/greenplum-db-5.0.x.x/greenplum_path.sh
   # gpssh -h smdw -e 'mkdir /data/master'
   # gpssh -h smdw -e 'chown gpadmin /data/master'
   ```

Creating Data Storage Areas on Segment Hosts

Data storage areas are required on the Greenplum Database segment hosts for primary segments. Separate storage areas are required for mirror segments.

To create the data directory locations on all segment hosts

1. On the master host, log in as root:

   ```bash
   # su
   ``

2. Create a file called `hostfile_gpssh_segonly`. This file should have only one machine configured host name for each segment host. For example, if you have three segment hosts:

   ```
   sdw1
   sdw2
   sdw3
   ```

3. Using gpssh, create the primary and mirror data directory locations on all segment hosts at once using the `hostfile_gpssh_segonly` file you just created. For example:

   ```bash
   # source /usr/local/greenplum-db-5.0.x.x/greenplum_path.sh
   # gpssh -f hostfile_gpssh_segonly -e 'mkdir /data/primary'
   # gpssh -f hostfile_gpssh_segonly -e 'mkdir /data/mirror'
   # gpssh -f hostfile_gpssh_segonly -e 'chown gpadmin /data/primary'
   # gpssh -f hostfile_gpssh_segonly -e 'chown gpadmin /data/mirror'
   ```

Synchronizing System Clocks

You should use NTP (Network Time Protocol) to synchronize the system clocks on all hosts that comprise your Greenplum Database system. See [www.ntp.org](http://www.ntp.org) for more information about NTP.
NTP on the segment hosts should be configured to use the master host as the primary time source, and the standby master as the secondary time source. On the master and standby master hosts, configure NTP to point to your preferred time server.

**To configure NTP**

1. On the master host, log in as root and edit the `/etc/ntp.conf` file. Set the `server` parameter to point to your data center's NTP time server. For example (if `10.6.220.20` was the IP address of your data center's NTP server):

   ```
   server 10.6.220.20
   ```

2. On each segment host, log in as root and edit the `/etc/ntp.conf` file. Set the first `server` parameter to point to the master host, and the second `server` parameter to point to the standby master host. For example:

   ```
   server mdw prefer
   server smdw
   ```

3. On the standby master host, log in as root and edit the `/etc/ntp.conf` file. Set the first `server` parameter to point to the primary master host, and the second `server` parameter to point to your data center's NTP time server. For example:

   ```
   server mdw prefer
   server 10.6.220.20
   ```

4. On the master host, use the NTP daemon synchronize the system clocks on all Greenplum hosts. For example, using `gpssh`:

   ```
   # gpssh -f hostfile_gpssh_allhosts -v -e 'ntpd'
   ```

**Enabling iptables**

On Linux systems, you can configure and enable the `iptables` firewall to work with Greenplum Database.

*Note:* Greenplum Database performance might be impacted when `iptables` is enabled. You should test the performance of your application with `iptables` enabled to ensure that performance is acceptable.

For more information about `iptables` see the `iptables` and firewall documentation for your operating system.

**How to Enable iptables**

1. As gpadmin, the Greenplum Database administrator, run this command on the Greenplum Database master host to stop Greenplum Database:

   ```
   $ gpstop -a
   ```

2. On the Greenplum Database hosts:

   a. Update the file `/etc/sysconfig/iptables` based on the *Example iptables Rules*.
   b. As root user, run these commands to enable `iptables`:

   ```
   # chkconfig iptables on
   # service iptables start
   ```
3. As gpadmin, run this command on the Greenplum Database master host to start Greenplum Database:

```
$ gpstart -a
```

**Warning:** After enabling `iptables`, this error in the `/var/log/messages` file indicates that the setting for the `iptables` table is too low and needs to be increased.

```
ip_conntrack: table full, dropping packet.
```

As root user, run this command to view the `iptables` table value:

```
# sysctl net.ipv4.netfilter.ip_conntrack_max
```

The following is the recommended setting to ensure that the Greenplum Database workload does not overflow the `iptables` table. The value might need to be adjusted for your hosts:

```
net.ipv4.netfilter.ip_conntrack_max=6553600
```

You can update `/etc/sysctl.conf` file with the value. For setting values in the file, see Setting the Greenplum Recommended OS Parameters.

To set the value until the next reboots run this command as root.

```
# sysctl net.ipv4.netfilter.ip_conntrack_max=6553600
```

### Example iptables Rules

When `iptables` is enabled, `iptables` manages the IP communication on the host system based on configuration settings (rules). The example rules are used to configure `iptables` for Greenplum Database master host, standby master host, and segment hosts.

- **Example Master and Standby Master iptables Rules**
- **Example Segment Host iptables Rules**

The two sets of rules account for the different types of communication Greenplum Database expects on the master (primary and standby) and segment hosts. The rules should be added to the `/etc/sysconfig/iptables` file of the Greenplum Database hosts. For Greenplum Database, `iptables` rules should allow the following communication:

- For customer facing communication with the Greenplum Database master, allow at least `postgres` and `28080` (eth1 interface in the example).
- For Greenplum Database system interconnect, allow communication using `tcp`, `udp`, and `icmp` protocols (`eth4` and `eth5` interfaces in the example).

The network interfaces that you specify in the `iptables` settings are the interfaces for the Greenplum Database hosts that you list in the `hostfile_gpinitsystem` file. You specify the file when you run the `gpinitsystem` command to initialize a Greenplum Database system. See Initializing a Greenplum Database System for information about the `hostfile_gpinitsystem` file and the `gpinitsystem` command.

- For the administration network on a Greenplum DCA, allow communication using `ssh`, `snmp`, `ntp`, and `icmp` protocols. (`eth0` interface in the example).

In the `iptables` file, each append rule command (lines starting with `-A`) is a single line.

The example rules should be adjusted for your configuration. For example:

- The append command, the `-A` lines and connection parameter `-i` should match the connectors for your hosts.
- the CIDR network mask information for the source parameter `-s` should match the IP addresses for your network.
Example Master and Standby Master iptables Rules

Example iptables rules with comments for the /etc/sysconfig/iptables file on the Greenplum Database master host and standby master host.

```
*filter
# Following 3 are default rules. If the packet passes through
# the rule set it gets these rule.
# Drop all inbound packets by default.
# Drop all forwarded (routed) packets.
# Let anything outbound go through.
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
# Accept anything on the loopback interface.
-A INPUT -i lo -j ACCEPT
# If a connection has already been established allow the
# remote host packets for the connection to pass through.
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
# These rules let all tcp and udp through on the standard
# interconnect IP addresses and on the interconnect interfaces.
# NOTE: gpsyncmaster uses random tcp ports in the range 1025 to 65535
# and Greenplum Database uses random udp ports in the range 1025 to 65535.
-A INPUT -i eth4 -p udp -s 192.0.2.0/22 -j ACCEPT
-A INPUT -i eth5 -p udp -s 198.51.100.0/22 -j ACCEPT
-A INPUT -i eth4 -p tcp -s 192.0.2.0/22 --syn --m state --state NEW
-A INPUT -i eth5 -p tcp -s 198.51.100.0/22 --syn --m state --state NEW
# Allow snmp connections on the admin network on Greenplum DCA.
-A INPUT -i eth0 -p udp --dport snmp -s 203.0.113.0/21 -j ACCEPT
-A INPUT -i eth0 -p udp --dport snmp -s 203.0.113.0/21 --syn --m state --state NEW
# Allow udp/tcp ntp connections on the admin network on Greenplum DCA.
-A INPUT -i eth0 -p udp --dport ntp -s 203.0.113.0/21 -j ACCEPT
-A INPUT -i eth0 -p tcp --dport ntp -s 203.0.113.0/21 --syn --m state --state NEW
# Allow ssh on all networks (This rule can be more strict).
-A INPUT -p tcp --dport ssh -j ACCEPT --syn --m state --state NEW
# Allow Greenplum Database on all networks.
-A INPUT -p tcp --dport postgres -j ACCEPT --syn --m state --state NEW
# Allow Greenplum Command Center on the customer facing network.
-A INPUT -i eth1 -p tcp --dport 28080 -j ACCEPT --syn --m state --state NEW
# Allow ping and any other icmp traffic on the interconnect networks.
-A INPUT -i eth4 -p icmp -s 192.0.2.0/22 -j ACCEPT
-A INPUT -i eth5 -p icmp -s 198.51.100.0/22 -j ACCEPT
# Allow ping only on the admin network on Greenplum DCA.
-A INPUT -i eth0 -p icmp --icmp-type echo-request -s 203.0.113.0/21 -j ACCEPT
# Log an error if a packet passes through the rules to the default
# INPUT rule (a DROP).
-A INPUT -m limit --limit 5/min -j LOG --log-prefix "iptables denied: " --log-level 7
COMMIT
```

Example Segment Host iptables Rules

Example iptables rules for the /etc/sysconfig/iptables file on the Greenplum Database segment hosts. The rules for segment hosts are similar to the master rules with fewer interfaces and fewer udp and tcp services.

```
*filter
```
Amazon EC2 Configuration (Amazon Web Services)

You can install and configure Greenplum Database on virtual servers provided by the Amazon Elastic Compute Cloud (Amazon EC2) web service. Amazon EC2 is a service provided by Amazon Web Services (AWS). The following overview information describes how to install Greenplum Database in an Amazon EC2 environment.

About Amazon EC2

You can use Amazon EC2 to launch as many virtual servers as you need, configure security and networking, and manage storage. An EC2 instance is a virtual server in the AWS cloud virtual computing environment.

EC2 instances are managed by AWS. AWS isolates your EC2 instances from other users in a virtual private cloud (VPC) and lets you control access to the instances. You can configure instance features such as operating system, network connectivity (network ports and protocols, IP address access), access to the Internet, and size and type of disk storage.

When you launch an instance, you use a preconfigured template for your instance, known as an Amazon Machine Image (AMI). The AMI packages the bits you need for your server (including the operating system and additional software). You can use images supplied by Amazon or use customized images. You launch instances in an Availability Zone of an AWS region. An Availability Zone is a distinct location within a region that are engineered to be insulated from failures in other Availability Zones.

For information about Amazon EC2, see http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

Launching EC2 Instances with the EC2 Console

You can launch instances, configure, start, stop, and terminate (delete) virtual servers, with the Amazon EC2 Console. When you launch instances, you select these features.

Amazon Machine Image (AMI)

An AMI is a template that contains the software configuration (operating system, application server, and applications).

For Greenplum Database - Select an AMI that runs a supported operating system. See the Greenplum Database Release Notes for the release that you are installing.
**Note:** You create and launch a customized AMI, see *About Amazon Machine Image (AMI)*

**EC2 Instance Type**

A predefined set performance characteristics. Instance types comprise varying combinations of CPU, memory, default storage, and networking capacity. You can modify storage options when you add storage.

**For Greenplum Database** - The instance type must be an EBS-Optimized instance type when using Amazon EBS storage for Greenplum Database. See *Configure storage* for information about Greenplum Database storage requirements. For information about EBS-Optimized instances, see the Amazon documentation about *EBS-Optimized instances*.

For sufficient network performance, the instance type must also support EC2 enhanced networking. For information about EC2 enhanced networking, see the Amazon documentation about *Enhanced Networking on Linux Instances*.

The instances should be in a single VPC and subnet. Instances are always assigned a VPC internal IP address and can be assigned a public IP address for external and Internet access.

The internal IP address is used for Greenplum Database communication between hosts. You can also use the internal IP address to access an instance from another instance within the EC2 VPC. For information about configuring launched instances to communicate with each other, see *Working with EC2 Instances*.

A public IP address for the instance and an Internet gateway configured for the EC2 VPC are required for accessing the instance from an external source and for the instance to access the Internet. Internet access is required when installing Linux packages. When you launch a set of instances, you can enable or disable the automatic assignment of public IP addresses when the instances are started.

If automatic assignment of public IP addresses is enabled, instances are always assigned a public IP address when the instance starts. If automatic assignment of public IP addresses is disabled, you can associate a public IP address with the EC2 elastic IP, and temporarily associate public IP addresses to an instance to connect and configure the instance.

To control whether a public IP is assigned when launching an instance, see the Amazon documentation about *Subnet Public IP Addressing*.

**EC2 Instance Details**

Information about starting, running, and stopping EC2 instances, such as such as number of instances of the same AMI, network information, and EC2 VPC and subnet membership.

**Configure storage**

Adjust and add storage. For example, you can change the size of root volume and add volumes.

**For Greenplum Database** - Greenplum Database supports either EC2 instance store or Amazon EBS storage in a production environment.

- **EC2 instance store** provides temporary block-level storage. This storage is located on disks that are physically attached to the host computer. With instance store, powering off the instance causes data loss. Soft reboots preserve instance store data. However, EC2 instance store can provide higher and more consistent I/O performance.

- **EBS storage** provides block level storage volumes with long-term persistence. EBS storage must be mounted with the XFS file system for it to be a supported configuration. All other file systems are explicitly not supported by Pivotal.

There are several classes of EBS. For Greenplum Database, select the EBS volume type *gp2* for root and swap partitions and *st1* for data. See the Amazon documentation about *Block Device Mapping*. 
For more information about the Amazon storage types, see Notes.

Create Tag
An optional label that consists of a case-sensitive key-value pair that is used for organizing searching a large number of EC2 resources.

Security Group
A set of firewall rules that control the network traffic for instances.

For external access to an instance with ssh, create a rule that enables ssh for inbound network traffic.

Working with EC2 Instances

After the EC2 instances have started, you connect to and configure the instances. The Instances page of the EC2 Console lists the running instances and network information. If the instance does not have a public IP address, you can create an Elastic IP and associate it with the instance. See About Amazon Elastic IP Addresses.

To access EC2 instances, AWS uses public-key cryptography to secure the login information for your instance. A Linux instance has no password; you use a key pair to log in to your instance securely. You specify the name of the key pair when you launch your instance, then provide the private key when you log in using SSH. See the Amazon documentation about EC2 Key Pairs.

A key pair consists of a public key that AWS stores, and a private key file that you store. Together, they allow you to connect to your instance securely.

This example logs into an into EC2 instance from an external location with the private key file my-test.pem and user ec2-user. The user ec2-user is the default user for some Linux AMI templates. This example assumes that the instance is configured with a public IP address 192.0.2.82 and that the pem file is the private key file that is used to access the instance.

```
ssh -i my-test.pem ec2-user@192.0.2.82
```

You can also copy the private key file to your home .ssh directory as the id_rsa file. This example, creates and configures the id_rsa file.

```
cp my-test.pem ~/.ssh/id_rsa
chmod 400 .ssh/id_rsa
```

You can also copy the id_rsa to your EC2 instances. This scp command copies the file to the .ssh directory of the ec2-user.

```
scp ~/.ssh/id_rsa ec2-user@192.0.2.86:~/.ssh/id_rsa
```

Note: gpssh-extkey is not used with Greenplum Database hosts that are EC2 instances. You must copy the private key file to the .ssh directory in the user home directory for each instance.

This example logs into an into EC2 instance using the id_rsa file.

```
ssh ec2-user@192.0.2.82
```

After the key file is installed on all Greenplum Database hosts you can use Greenplum Database utilities such as gpseginstall, gpssh, and gpscp that access multiple Greenplum Database hosts.

Before installing Greenplum Database, you configure the EC2 instances as you would a local host server machines. Configure the host operating system, configure host network information (for example, update the /etc/hosts file), set operating system parameters, and install operating system packages. For information about how to prepare your operating system environment for Greenplum Database, see Configuring Your Systems and Installing Greenplum.
These example commands use yum to install the Linux packages zlib, sed, unzip, and vim.

```
sudo yum install -y zlib
sudo yum install -y sed
sudo yum install -y unzip
sudo yum install -y vim
```

This example uploads Greenplum Database install file to an EC2 instance to the ec2-user home directory.

```
scp greenplum-db-5.0.0-build-1-RHEL6-x86_64.zip ec2-user@192.0.2.82:~/.
```

These example commands log into the instance and run the Greenplum Database install file that is in ec2-user home directory as the as ec2-user.

```
ssh ec2-user@192.0.2.82
unzip greenplum-db-5.0.0-build-1-RHEL6-x86_64.zip
./greenplum-db-5.0.0-build-1-RHEL6-x86_64.bin
```

This example command runs the gpseginstall utility that specifies the user as ec2-user. This example assumes the file my-hosts contains the instances that are used as Greenplum Database segment hosts and that the instances have been prepared for Greenplum Database.

```
gpseginstall -u ec2-user -f my-hosts
```

Note: During the Greenplum Database installation process, you might see ssh messages to confirm the authenticity of host connections. Enter yes to confirm the authenticity.

### About Amazon Machine Image (AMI)

An Amazon Machine Image (AMI) is a template that contains a software configuration (for example, an operating system, an application server, and applications). From an AMI, you launch an instance, which is a copy of the AMI running as a virtual server in the cloud. You can launch multiple instances of an AMI.

After you launch an instance, it acts like a traditional host, and you can interact with it as you would any computer. You have complete control of your instances; you can use sudo to run commands that require root privileges.

You can create a customized Amazon EBS-backed Linux AMI from an instance that you've launched from an existing Amazon EBS-backed Linux AMI. After you've customized the instance to suit your needs, create and register a new AMI, which you can use to launch new instances with these customizations.

For information about AMI, see the Amazon documentation about AMIs.

### About Amazon Elastic IP Addresses

An EC2 Elastic IP address is a public IP address that you can allocate (create) for your account. You can associate it to and disassociate it from instances as you require, and it's allocated to your account until you choose to release it.

Your default VPC is configured with an Internet gateway. When you allocate an EC2 Elastic IP address, AWS configures the VPC to allow internet access to the IP address using the gateway.

To enable an instance in your VPC to communicate with the Internet, it must have a public IP address or an EC2 Elastic IP address that's associated with a private IP address on your instance.

To ensure that your instances can communicate with the Internet, you must also attach an Internet gateway to your EC2 VPC. For information about VPC Internet Gateways, see the Amazon documentation about Internet gateways.
For information about EC2 Elastic IP addresses and how to use them, see see the Amazon documentation about Elastic IP Addresses.

**Notes**

- The Greenplum Database utility `gpssh-extkey` is not used with Greenplum Database hosts that are EC2 instances. You must copy the private key file to the `.ssh` directory in the user home directory for each instance.
- When you use Amazon EBS storage for Greenplum Database storage, the storage must be mounted with the XFS file system for it to be a supported configuration.

For information about EBS storage, see the Amazon documentation about Amazon EBS. Also, see the Amazon EC2 documentation for configuring the Amazon EBS volumes and managing storage and file systems used with EC2 instances.
- For an EC2 instance with instance store, the virtual devices for instance store volumes are `ephemeralN` (n is between 0 and 23). On an instance running CentOS the instance store block device names appear as `/dev/xvdletter`. A RAID0 configuration is recommended with ephemeral disks because there are more disks than segments on each host.

An example of an EC2 instance type that is configured with instance store and that showed acceptable performance is the `d2.8xlarge` instance type configured with four RAID0 volumes of 6 disks each.

For information about EC2 instance store, see the Amazon documentation about EC2 Instance Store.
- These are default ports in a Greenplum Database environment. These ports need to be open in the security group to allow access from a source external to a VPC.

<table>
<thead>
<tr>
<th>Port</th>
<th>Used by this application</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>ssh - connect to host with ssh</td>
</tr>
<tr>
<td>5432</td>
<td>Greenplum Database (master)</td>
</tr>
<tr>
<td>28080</td>
<td>Greenplum Command Center</td>
</tr>
</tbody>
</table>

- For a non-default VPC you can configure the VPC with an internet gateway for the VPC and allocate Elastic IP address for the VPC. AWS will automatically configure the Elastic IP for internet access. For information about EC2 internet gateways, see the Amazon documentation about Internet Gateways.
- A placement group is a logical grouping of instances within a single Availability Zone. Using placement groups enables applications to participate in a low-latency, 10 Gbps network. Placement groups are recommended for applications that benefit from low network latency, high network throughput, or both.

Placement Groups provide the ability for EC2 instances to separated from other instances. However, configuring instances are in different placement groups can improve performance but might create a configuration where an instance in a placement group cannot be replaced.

See the Amazon documentation about Placement Groups.

- Amazon EC2 provides enhanced networking capabilities using single root I/O virtualization (SR-IOV) on some instance types. Enabling enhanced networking on your instance results in higher performance (packets per second), lower latency, and lower jitter.

To enable enhanced networking on your Red Hat and CentOS RHEL/CentOS instance, you must ensure that the kernel has the `ixgbevf` module version 2.14.2 or higher is installed and that the `sriovNetSupport` attribute is set.

For information about EC2 enhanced networking, see the Amazon documentation about Enhanced Networking on Linux Instances.

**References**

References to AWS and EC2 features and related information.

- AWS - [https://aws.amazon.com/](https://aws.amazon.com/).
• Amazon EC2 and VPC command line interface (CLI) - http://docs.aws.amazon.com/AWSEC2/latest/CommandLineReference/Welcome.html.
• Amazon S3 (secure, durable, scalable object storage) - https://aws.amazon.com/s3/.

With AWS CloudFormation, you can create templates to simplify provisioning and management of related AWS resources.

**Next Steps**

After you have configured the operating system environment and installed the Greenplum Database software on all of the hosts in the system, the next steps are:

• Validating Your Systems
• Initializing a Greenplum Database System
Installing the Data Science Packages

Information about installing the Greenplum Database Python and R Data Science Packages.

Python Data Science Module Package

Greenplum Database provides a collection of data science-related Python modules that can be used with the Greenplum Database PL/Python language. You can download these modules in .gppkg format from Pivotal Network.

This section contains the following information:

- Python Data Science Modules
- Installing the Python Data Science Module Package
- Uninstalling the Python Data Science Module Package

For information about the Greenplum Database PL/Python Language, see *Greenplum PL/Python Language Extension*.

Python Data Science Modules

Modules provided in the Python Data Science package include:

Table 7: Data Science Modules

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Description/Used For</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beautiful Soup</td>
<td>Navigating HTML and XML</td>
</tr>
<tr>
<td>Gensim</td>
<td>Topic modeling and document indexing</td>
</tr>
<tr>
<td>Keras (RHEL/CentOS 7 only)</td>
<td>Deep learning</td>
</tr>
<tr>
<td>Lifelines</td>
<td>Survival analysis</td>
</tr>
<tr>
<td>lxml</td>
<td>XML and HTML processing</td>
</tr>
<tr>
<td>NLTK</td>
<td>Natural language toolkit</td>
</tr>
<tr>
<td>NumPy</td>
<td>Scientific computing</td>
</tr>
<tr>
<td>Pandas</td>
<td>Data analysis</td>
</tr>
<tr>
<td>Pattern-en</td>
<td>Part-of-speech tagging</td>
</tr>
<tr>
<td>pyLDAvis</td>
<td>Interactive topic model visualization</td>
</tr>
<tr>
<td>PyMC3</td>
<td>Statistical modeling and probabilistic machine learning</td>
</tr>
<tr>
<td>scikit-learn</td>
<td>Machine learning data mining and analysis</td>
</tr>
<tr>
<td>SciPy</td>
<td>Scientific computing</td>
</tr>
<tr>
<td>spaCy</td>
<td>Large scale natural language processing</td>
</tr>
<tr>
<td>StatsModels</td>
<td>Statistical modeling</td>
</tr>
<tr>
<td>Tensorflow (RHEL/CentOS 7 only)</td>
<td>Numerical computation using data flow graphs</td>
</tr>
<tr>
<td>XGBoost</td>
<td>Gradient boosting, classifying, ranking</td>
</tr>
</tbody>
</table>
Installing the Python Data Science Module Package

Before you install the Python Data Science Module package, make sure that your Greenplum Database is running, you have sourced `greenplum_path.sh`, and that the `MASTER_DATA_DIRECTORY` and `$GPHOME` environment variables are set.

**Note:** The PyMC3 module depends on Tk. If you want to use PyMC3, you must install the `tk` OS package on every node in your cluster. For example:

```
$ yum install tk
```

1. Locate the Python Data Science module package that you built or downloaded.
   The file name format of the package is `DataSciencePython-<version>-relhel<N>-x86_64.gppkg`.
2. Copy the package to the Greenplum Database master host.
3. Use the `gppkg` command to install the package. For example:

```
$ gppkg -i DataSciencePython-<version>-relhel<N>-x86_64.gppkg
```

`gppkg` installs the Python Data Science modules on all nodes in your Greenplum Database cluster. The command also updates the `PYTHONPATH`, `PATH`, and `LD_LIBRARY_PATH` environment variables in your `greenplum_path.sh` file.
4. Restart Greenplum Database. You must re-source `greenplum_path.sh` before restarting your Greenplum cluster:

```
$ source /usr/local/greenplum-db/greenplum_path.sh
$ gpstop -r
```

The Greenplum Database Python Data Science Modules are installed in the following directory:

```
$GPHOME/ext/DataSciencePython/lib/python2.7/site-packages/
```

Uninstalling the Python Data Science Module Package

Use the `gppkg` utility to uninstall the Python Data Science Module package. You must include the version number in the package name you provide to `gppkg`.

To determine your Python Data Science Module package version number and remove this package:

```
$ gppkg -q --all | grep DataSciencePython
DataSciencePython-<version>
$ gppkg -r DataSciencePython-<version>
```

The command removes the Python Data Science modules from your Greenplum Database cluster. It also updates the `PYTHONPATH`, `PATH`, and `LD_LIBRARY_PATH` environment variables in your `greenplum_path.sh` file to their pre-installation values.

Re-source `greenplum_path.sh` and restart Greenplum Database after you remove the Python Data Science Module package:

```
$ . /usr/local/greenplum-db/greenplum_path.sh
$ gpstop -r
```

**Note:** When you uninstall the Python Data Science Module package from your Greenplum Database cluster, any UDFs that you have created that import Python modules installed with this package will return an error.
**R Data Science Library Package**

R packages are modules that contain R functions and data sets. Greenplum Database provides a collection of data science-related R libraries that can be used with the Greenplum Database PL/R language. You can download these libraries in .gppkg format from Pivotal Network.

This chapter contains the following information:

- **R Data Science Libraries**
- **Installing the R Data Science Library Package**
- **Uninstalling the R Data Science Library Package**

For information about the Greenplum Database PL/R Language, see *Greenplum PL/R Language Extension*.

**R Data Science Libraries**

Libraries provided in the R Data Science package include:

<table>
<thead>
<tr>
<th>abind</th>
<th>glmnet</th>
<th>quantreg</th>
</tr>
</thead>
<tbody>
<tr>
<td>adabag</td>
<td>gplots</td>
<td>R2jags</td>
</tr>
<tr>
<td>arm</td>
<td>gtable</td>
<td>R6</td>
</tr>
<tr>
<td>assertthat</td>
<td>gtools</td>
<td>randomForest</td>
</tr>
<tr>
<td>BH</td>
<td>hms</td>
<td>RColorBrewer</td>
</tr>
<tr>
<td>bitops</td>
<td>hybridHclust</td>
<td>Rcpp</td>
</tr>
<tr>
<td>car</td>
<td>igraph</td>
<td>RcppEigen</td>
</tr>
<tr>
<td>caret</td>
<td>labeling</td>
<td>readr</td>
</tr>
<tr>
<td>caTools</td>
<td>lattice</td>
<td>reshape2</td>
</tr>
<tr>
<td>coda</td>
<td>lazyeval</td>
<td>rjags</td>
</tr>
<tr>
<td>colorspace</td>
<td>lattice</td>
<td>RobustRankAggreg</td>
</tr>
<tr>
<td>compHclust</td>
<td>MASS</td>
<td>ROCR</td>
</tr>
<tr>
<td>curl</td>
<td>Matrix</td>
<td>rpart</td>
</tr>
<tr>
<td>data.table</td>
<td>MCMCPack</td>
<td>RPostgreSQL</td>
</tr>
<tr>
<td>DBI</td>
<td>MTS</td>
<td>sandwich</td>
</tr>
<tr>
<td>dichromat</td>
<td>minqa</td>
<td>scales</td>
</tr>
<tr>
<td>digest</td>
<td>MTS</td>
<td>SparseM</td>
</tr>
<tr>
<td>dplyr</td>
<td>MTS</td>
<td>stringi</td>
</tr>
<tr>
<td>e1071</td>
<td>minqa</td>
<td>stringr</td>
</tr>
<tr>
<td>flashClust</td>
<td>minqa</td>
<td>survival</td>
</tr>
<tr>
<td>forecast</td>
<td>MTS</td>
<td>tibble</td>
</tr>
<tr>
<td>foreign</td>
<td>MTS</td>
<td>tseries</td>
</tr>
<tr>
<td>gdata</td>
<td>MTS</td>
<td>zoo</td>
</tr>
<tr>
<td>ggplot2</td>
<td>MTS</td>
<td></td>
</tr>
</tbody>
</table>
Installing the R Data Science Library Package

Before you install the R Data Science Library package, make sure that your Greenplum Database is running, you have sourced `greenplum_path.sh`, and that the `$MASTER_DATA_DIRECTORY` and `$GPHOME` environment variables are set.

1. **Locate the R Data Science library package that you built or downloaded.**
   
   The file name format of the package is `DataScienceR-<version>-relhel<N>-x86_64.gppkg`.

2. **Copy the package to the Greenplum Database master host.**

3. **Use the `gppkg` command to install the package. For example:**

   ```bash
   $ gppkg -i DataScienceR-<version>-relhel<N>-x86_64.gppkg
   
gppkg installs the R Data Science libraries on all nodes in your Greenplum Database cluster. The command also sets the `R_LIBS_USER` environment variable and updates the `PATH` and `LD_LIBRARY_PATH` environment variables in your `greenplum_path.sh` file.

4. **Restart Greenplum Database. You must re-source `greenplum_path.sh` before restarting your Greenplum cluster:**

   ```bash
   $ source /usr/local/greenplum-db/greenplum_path.sh
   $ gpstop -r
   
   The Greenplum Database R Data Science Modules are installed in the following directory:
   
   `$GPHOME/ext/DataScienceR/library`

   **Note:** `rjags` libraries are installed in the `$GPHOME/ext/DataScienceR/extlib/lib` directory. If you want to use `rjags` and your `$GPHOME` is not `/usr/local/greenplum-db`, you must perform additional configuration steps to create a symbolic link from `$GPHOME` to `/usr/local/greenplum-db` on each node in your Greenplum Database cluster. For example:

   ```bash
   $ gpssh -f all_hosts -e 'ln -s $GPHOME /usr/local/greenplum-db'
   $ gpssh -f all_hosts -e 'chown -h gpadmin /usr/local/greenplum-db'
   
   Uninstalling the R Data Science Library Package

   Use the `gppkg` utility to uninstall the R Data Science Library package. You must include the version number in the package name you provide to `gppkg`.

   To determine your R Data Science Library package version number and remove this package:

   ```bash
   $ gppkg -q --all | grep DataScienceR
   DataScienceR-<version>
   $ gppkg -r DataScienceR-<version>
   
   The command removes the R Data Science libraries from your Greenplum Database cluster. It also removes the `R_LIBS_USER` environment variable and updates the `PATH` and `LD_LIBRARY_PATH` environment variables in your `greenplum_path.sh` file to their pre-installation values.

   Re-source `greenplum_path.sh` and restart Greenplum Database after you remove the R Data Science Library package:

   ```bash
   $ . /usr/local/greenplum-db/greenplum_path.sh
   $ gpstop -r
**Note:** When you uninstall the R Data Science Library package from your Greenplum Database cluster, any UDFs that you have created that use R libraries installed with this package will return an error.
Validating Your Systems

Validate your operating system settings, hardware, and network.
Greenplum provides the following utilities to validate the configuration and performance of your systems:

- gpcheck
- gpcheckperf

These utilities can be found in $GPHOME/bin of your Greenplum installation.
The following tests should be run prior to initializing your Greenplum Database system.

Validating OS Settings

Greenplum provides a utility called gpcheck that can be used to verify that all hosts in your array have the recommended OS settings for running a production Greenplum Database system. To run gpcheck:

1. Log in on the master host as the gpadmin user.
2. Source the greenplum_path.sh path file from your Greenplum installation. For example:
   
   ```bash
   $ source /usr/local/greenplum-db/greenplum_path.sh
   ```
3. Create a file called hostfile_gpcheck that has the machine-configured host names of each Greenplum host (master, standby master and segments), one host name per line. Make sure there are no blank lines or extra spaces. This file should just have a single host name per host. For example:
   
   ```
   mdw
   smdw
   sdw1
   sdw2
   sdw3
   ```
4. Run the gpcheck utility using the host file you just created. For example:
   
   ```bash
   $ gpcheck -f hostfile_gpcheck -m mdw -s smdw
   ```
5. After gpcheck finishes verifying OS parameters on all hosts (masters and segments), you might be prompted to modify certain OS parameters before initializing your Greenplum Database system.

Validating Hardware Performance

Greenplum provides a management utility called gpcheckperf, which can be used to identify hardware and system-level issues on the machines in your Greenplum Database array. gpcheckperf starts a session on the specified hosts and runs the following performance tests:

- Network Performance (gpnetbench*)
- Disk I/O Performance (dd test)
- Memory Bandwidth (stream test)

Before using gpcheckperf, you must have a trusted host setup between the hosts involved in the performance test. You can use the utility gpssh-exkeys to update the known host files and exchange public keys between hosts if you have not done so already. Note that gpcheckperf calls to gpssh and gpscp, so these Greenplum utilities must be in your $PATH.

Validating Network Performance

To test network performance, run gpcheckperf with one of the network test run options: parallel pair test (-r N), serial pair test (-r n), or full matrix test (-r M). The utility runs a network benchmark program...
that transfers a 5 second stream of data from the current host to each remote host included in the test. By default, the data is transferred in parallel to each remote host and the minimum, maximum, average and median network transfer rates are reported in megabytes (MB) per second. If the summary transfer rate is slower than expected (less than 100 MB/s), you can run the network test serially using the \(-r \, n\) option to obtain per-host results. To run a full-matrix bandwidth test, you can specify \(-r \, M\) which will cause every host to send and receive data from every other host specified. This test is best used to validate if the switch fabric can tolerate a full-matrix workload.

Most systems in a Greenplum Database array are configured with multiple network interface cards (NICs), each NIC on its own subnet. When testing network performance, it is important to test each subnet individually. For example, considering the following network configuration of two NICs per host:

<table>
<thead>
<tr>
<th>Greenplum Host</th>
<th>Subnet1 NICs</th>
<th>Subnet2 NICs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment 1</td>
<td>sdw1-1</td>
<td>sdw1-2</td>
</tr>
<tr>
<td>Segment 2</td>
<td>sdw2-1</td>
<td>sdw2-2</td>
</tr>
<tr>
<td>Segment 3</td>
<td>sdw3-1</td>
<td>sdw3-2</td>
</tr>
</tbody>
</table>

You would create four distinct host files for use with the `gpcheckperf` network test:

<table>
<thead>
<tr>
<th>Table 8: Example Network Interface Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostfile_gpchecknet_ic1</td>
</tr>
<tr>
<td>sdw1-1</td>
</tr>
<tr>
<td>sdw2-1</td>
</tr>
<tr>
<td>sdw3-1</td>
</tr>
</tbody>
</table>

You would then run `gpcheckperf` once per subnet. For example (if testing an even number of hosts, run in parallel pairs test mode):

```
$ gpcheckperf -f hostfile_gpchecknet_ic1 -r N -d /tmp > subnet1.out
$ gpcheckperf -f hostfile_gpchecknet_ic2 -r N -d /tmp > subnet2.out
```

If you have an odd number of hosts to test, you can run in serial test mode (\(-r \, n\)).

### Validating Disk I/O and Memory Bandwidth

To test disk and memory bandwidth performance, run `gpcheckperf` with the disk and stream test run options (\(-r \, ds\)). The disk test uses the `dd` command (a standard UNIX utility) to test the sequential throughput performance of a logical disk or file system. The memory test uses the STREAM benchmark program to measure sustainable memory bandwidth. Results are reported in MB per second (MB/s).

#### To run the disk and stream tests

1. Log in on the master host as the `gpadmin` user.
2. Source the `greenplum_path.sh` path file from your Greenplum installation. For example:
   ```
   $ source /usr/local/greenplum-db/greenplum_path.sh
   ```
3. Create a host file named `hostfile_gpcheckperf` that has one host name per segment host. Do not include the master host. For example:
   ```
   sdw1
   ```
4. Run the `gpcheckperf` utility using the `hostfile_gpcheckperf` file you just created. Use the `-d` option to specify the file systems you want to test on each host (you must have write access to these directories). You will want to test all primary and mirror segment data directory locations. For example:

```
$ gpcheckperf -f hostfile_gpcheckperf -r ds -D \
   -d /data1/primary -d /data2/primary \ 
   -d /data1/mirror -d /data2/mirror
```

5. The utility may take a while to perform the tests as it is copying very large files between the hosts. When it is finished you will see the summary results for the Disk Write, Disk Read, and Stream tests.
Configuring Localization Settings

Describes the available localization features of Greenplum Database.

Greenplum Database supports localization with two approaches:

- Using the locale features of the operating system to provide locale-specific collation order, number formatting, and so on.
- Providing a number of different character sets defined in the Greenplum Database server, including multiple-byte character sets, to support storing text in all kinds of languages, and providing character set translation between client and server.

About Locale Support in Greenplum Database

Locale support refers to an application respecting cultural preferences regarding alphabets, sorting, number formatting, etc. Greenplum Database uses the standard ISO C and POSIX locale facilities provided by the server operating system. For additional information refer to the documentation of your operating system.

Locale support is automatically initialized when a Greenplum Database system is initialized. The initialization utility, `gpinitsystem`, will initialize the Greenplum array with the locale setting of its execution environment by default, so if your system is already set to use the locale that you want in your Greenplum Database system then there is nothing else you need to do.

When you are ready to initiate Greenplum Database and you want to use a different locale (or you are not sure which locale your system is set to), you can instruct `gpinitsystem` exactly which locale to use by specifying the `-n locale` option. For example:

```
$ gpinitsystem -c gp_init_config -n sv_SE
```

See Initializing a Greenplum Database System for information about the database initialization process.

The example above sets the locale to Swedish (sv) as spoken in Sweden (SE). Other possibilities might be `en_US` (U.S. English) and `fr_CA` (French Canadian). If more than one character set can be useful for a locale then the specifications look like this: `cs_CZ.ISO8859-2`. What locales are available under what names on your system depends on what was provided by the operating system vendor and what was installed. On most systems, the command `locale -a` will provide a list of available locales.

Occasionally it is useful to mix rules from several locales, for example use English collation rules but Spanish messages. To support that, a set of locale subcategories exist that control only a certain aspect of the localization rules:

- `LC_COLLATE` — String sort order
- `LC_CTYPE` — Character classification (What is a letter? Its upper-case equivalent?)
- `LC_MESSAGES` — Language of messages
- `LC_MONETARY` — Formatting of currency amounts
- `LC_NUMERIC` — Formatting of numbers
- `LC_TIME` — Formatting of dates and times

If you want the system to behave as if it had no locale support, use the special locale `C` or `POSIX`.

The nature of some locale categories is that their value has to be fixed for the lifetime of a Greenplum Database system. That is, once `gpinitsystem` has run, you cannot change them anymore. `LC_COLLATE` and `LC_CTYPE` are those categories. They affect the sort order of indexes, so they must be kept fixed, or indexes on text columns will become corrupt. Greenplum Database enforces this by recording the values of `LC_COLLATE` and `LC_CTYPE` that are seen by `gpinitsystem`. The server automatically adopts those two values based on the locale that was chosen at initialization time.
The other locale categories can be changed as desired whenever the server is running by setting the server configuration parameters that have the same name as the locale categories (see the Greenplum Database Reference Guide for more information on setting server configuration parameters). The defaults that are chosen by gpinitsystem are written into the master and segment postgresql.conf configuration files to serve as defaults when the Greenplum Database system is started. If you delete these assignments from the master and each segment postgresql.conf files then the server will inherit the settings from its execution environment.

Note that the locale behavior of the server is determined by the environment variables seen by the server, not by the environment of any client. Therefore, be careful to configure the correct locale settings on each Greenplum Database host (master and segments) before starting the system. A consequence of this is that if client and server are set up in different locales, messages may appear in different languages depending on where they originated.

Inheriting the locale from the execution environment means the following on most operating systems: For a given locale category, say the collation, the following environment variables are consulted in this order until one is found to be set: LC_ALL, LC_COLLATE (the variable corresponding to the respective category), LANG. If none of these environment variables are set then the locale defaults to C.

Some message localization libraries also look at the environment variable LANGUAGE which overrides all other locale settings for the purpose of setting the language of messages. If in doubt, please refer to the documentation for your operating system, in particular the documentation about gettext, for more information.

Native language support (NLS), which enables messages to be translated to the user’s preferred language, is not enabled in Greenplum Database for languages other than English. This is independent of the other locale support.

**Locale Behavior**

The locale settings influence the following SQL features:

- Sort order in queries using ORDER BY on textual data
- The ability to use indexes with LIKE clauses
- The upper, lower, and initcap functions
- The to_char family of functions

The drawback of using locales other than C or POSIX in Greenplum Database is its performance impact. It slows character handling and prevents ordinary indexes from being used by LIKE. For this reason use locales only if you actually need them.

**Troubleshooting Locales**

If locale support does not work as expected, check that the locale support in your operating system is correctly configured. To check what locales are installed on your system, you may use the command locale -a if your operating system provides it.

Check that Greenplum Database is actually using the locale that you think it is. LC_COLLATE and LC_CTYPE settings are determined at initialization time and cannot be changed without redoing gpinitsystem. Other locale settings including LC_MESSAGES and LC_MONETARY are initially determined by the operating system environment of the master and/or segment host, but can be changed after initialization by editing the postgresql.conf file of each Greenplum master and segment instance. You can check the active locale settings of the master host using the SHOW command. Note that every host in your Greenplum Database array should be using identical locale settings.

**Character Set Support**

The character set support in Greenplum Database allows you to store text in a variety of character sets, including single-byte character sets such as the ISO 8859 series and multiple-byte character sets such
as EUC (Extended Unix Code), UTF-8, and Mule internal code. All supported character sets can be used transparently by clients, but a few are not supported for use within the server (that is, as a server-side encoding). The default character set is selected while initializing your Greenplum Database array using `gpinitsystem`. It can be overridden when you create a database, so you can have multiple databases each with a different character set.

**Table 10: Greenplum Database Character Sets**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Language</th>
<th>Server?</th>
<th>Bytes/Char</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIG5</td>
<td>Big Five</td>
<td>Traditional Chinese</td>
<td>No</td>
<td>1-2</td>
<td>WIN950, Windows950</td>
</tr>
<tr>
<td>EUC_CN</td>
<td>Extended UNIX Code-CN</td>
<td>Simplified Chinese</td>
<td>Yes</td>
<td>1-3</td>
<td></td>
</tr>
<tr>
<td>EUC_TW</td>
<td>Extended UNIX Code-TW</td>
<td>Traditional Chinese, Taiwanese</td>
<td>Yes</td>
<td>1-3</td>
<td></td>
</tr>
<tr>
<td>GB18030</td>
<td>National Standard</td>
<td>Chinese</td>
<td>No</td>
<td>1-2</td>
<td></td>
</tr>
<tr>
<td>GBK</td>
<td>Extended National Standard</td>
<td>Simplified Chinese</td>
<td>No</td>
<td>1-2</td>
<td>WIN936, Windows936</td>
</tr>
<tr>
<td>ISO_8859_5</td>
<td>ISO 8859-5, ECMA 113</td>
<td>Latin/Cyrillic</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ISO_8859_6</td>
<td>ISO 8859-6, ECMA 114</td>
<td>Latin/Arabic</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ISO_8859_7</td>
<td>ISO 8859-7, ECMA 118</td>
<td>Latin/Greek</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ISO_8859_8</td>
<td>ISO 8859-8, ECMA 121</td>
<td>Latin/Hebrew</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>JOHAB</td>
<td>JOHA</td>
<td>Korean (Hangul)</td>
<td>Yes</td>
<td>1-3</td>
<td></td>
</tr>
<tr>
<td>KOI8</td>
<td>KOI8-R(U)</td>
<td>Cyrillic</td>
<td>Yes</td>
<td>1</td>
<td>KOI8R</td>
</tr>
<tr>
<td>LATIN1</td>
<td>ISO 8859-1, ECMA 94</td>
<td>Western European</td>
<td>Yes</td>
<td>1</td>
<td>ISO88591</td>
</tr>
<tr>
<td>LATIN2</td>
<td>ISO 8859-2, ECMA 94</td>
<td>Central European</td>
<td>Yes</td>
<td>1</td>
<td>ISO88592</td>
</tr>
<tr>
<td>LATIN3</td>
<td>ISO 8859-3, ECMA 94</td>
<td>South European</td>
<td>Yes</td>
<td>1</td>
<td>ISO88593</td>
</tr>
</tbody>
</table>

\footnote{Not all APIs support all the listed character sets. For example, the JDBC driver does not support MULE_INTERNAL, LATIN6, LATIN8, and LATIN10.}
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Language</th>
<th>Server?</th>
<th>Bytes/Char</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>LATIN4</td>
<td>ISO 8859-4,</td>
<td>North European</td>
<td>Yes</td>
<td>1</td>
<td>ISO88594</td>
</tr>
<tr>
<td>LATIN5</td>
<td>ISO 8859-9,</td>
<td>Turkish</td>
<td>Yes</td>
<td>1</td>
<td>ISO88599</td>
</tr>
<tr>
<td>LATIN6</td>
<td>ISO 8859-10,</td>
<td>Nordic</td>
<td>Yes</td>
<td>1</td>
<td>ISO885910</td>
</tr>
<tr>
<td>LATIN7</td>
<td>ISO 8859-13</td>
<td>Baltic</td>
<td>Yes</td>
<td>1</td>
<td>ISO885913</td>
</tr>
<tr>
<td>LATIN8</td>
<td>ISO 8859-14</td>
<td>Celtic</td>
<td>Yes</td>
<td>1</td>
<td>ISO885914</td>
</tr>
<tr>
<td>LATIN9</td>
<td>ISO 8859-15</td>
<td>LATIN1 with Euro and accents</td>
<td>Yes</td>
<td>1</td>
<td>ISO885915</td>
</tr>
<tr>
<td>LATIN10</td>
<td>ISO 8859-16,</td>
<td>Romanian</td>
<td>Yes</td>
<td>1</td>
<td>ISO885916</td>
</tr>
<tr>
<td></td>
<td>ASRO SR 14111</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULE_</td>
<td>Mule internal</td>
<td>Multilingual Emacs</td>
<td>Yes</td>
<td>1-4</td>
<td></td>
</tr>
<tr>
<td>INTERNAL</td>
<td>code</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SJIS</td>
<td>Shift JIS</td>
<td>Japanese</td>
<td>No</td>
<td>1-2</td>
<td>Mskanji, ShiftJIS, WIN932, Windows932</td>
</tr>
<tr>
<td>SQL_ASCII</td>
<td>unspecified</td>
<td></td>
<td>No</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>UHC</td>
<td>Unified Hangul</td>
<td>Korean</td>
<td>No</td>
<td>1-2</td>
<td>WIN949, Windows949</td>
</tr>
<tr>
<td>Code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UTF8</td>
<td>Unicode, 8-bit</td>
<td>all</td>
<td>Yes</td>
<td>1-4</td>
<td>Unicode</td>
</tr>
<tr>
<td>WIN866</td>
<td>Windows CP866</td>
<td>Cyrillic</td>
<td>Yes</td>
<td>1</td>
<td>ALT</td>
</tr>
<tr>
<td>WIN874</td>
<td>Windows CP874</td>
<td>Thai</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>WIN1250</td>
<td>Windows CP1250</td>
<td>Central European</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>WIN1251</td>
<td>Windows CP1251</td>
<td>Cyrillic</td>
<td>Yes</td>
<td>1</td>
<td>WIN</td>
</tr>
<tr>
<td>WIN1252</td>
<td>Windows CP1252</td>
<td>Western European</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>WIN1253</td>
<td>Windows CP1253</td>
<td>Greek</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>WIN1254</td>
<td>Windows CP1254</td>
<td>Turkish</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

2 The SQL_ASCII setting behaves considerably differently from the other settings. Byte values 0-127 are interpreted according to the ASCII standard, while byte values 128-255 are taken as uninterpreted characters. If you are working with any non-ASCII data, it is unwise to use the SQL_ASCII setting as a client encoding. SQL_ASCII is not supported as a server encoding.
### Setting the Character Set

gpinitsystem defines the default character set for a Greenplum Database system by reading the setting of the ENCODING parameter in the `gp_init_config` file at initialization time. The default character set is UNICODE or UTF8.

You can create a database with a different character set besides what is used as the system-wide default. For example:

```
=> CREATE DATABASE korean WITH ENCODING 'EUC_KR';
```

**Important:** Although you can specify any encoding you want for a database, it is unwise to choose an encoding that is not what is expected by the locale you have selected. The `LC_COLLATE` and `LC_CTYPE` settings imply a particular encoding, and locale-dependent operations (such as sorting) are likely to misinterpret data that is in an incompatible encoding.

Since these locale settings are frozen by gpinitsystem, the apparent flexibility to use different encodings in different databases is more theoretical than real.

One way to use multiple encodings safely is to set the locale to `C` or `POSIX` during initialization time, thus disabling any real locale awareness.

### Character Set Conversion Between Server and Client

Greenplum Database supports automatic character set conversion between server and client for certain character set combinations. The conversion information is stored in the master `pg_conversion` system catalog table. Greenplum Database comes with some predefined conversions or you can create a new conversion using the SQL command `CREATE CONVERSION`.

#### Table 11: Client/Server Character Set Conversions

<table>
<thead>
<tr>
<th>Server Character Set</th>
<th>Available Client Character Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIG5</td>
<td>not supported as a server encoding</td>
</tr>
<tr>
<td>EUC_CN</td>
<td>EUC_CN, MULE_INTERNAL, UTF8</td>
</tr>
<tr>
<td>EUC_JP</td>
<td>EUC_JP, MULE_INTERNAL, SJIS, UTF8</td>
</tr>
<tr>
<td>EUC_KR</td>
<td>EUC_KR, MULE_INTERNAL, UTF8</td>
</tr>
<tr>
<td>EUC_TW</td>
<td>EUC_TW, BIG5, MULE_INTERNAL, UTF8</td>
</tr>
<tr>
<td>GB18030</td>
<td>not supported as a server encoding</td>
</tr>
<tr>
<td>GBK</td>
<td>not supported as a server encoding</td>
</tr>
<tr>
<td>Server Character Set</td>
<td>Available Client Character Sets</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>ISO_8859_5</td>
<td>ISO_8859_5, KOI8, MULE_INTERNAL, UTF8, WIN866, WIN1251</td>
</tr>
<tr>
<td>ISO_8859_6</td>
<td>ISO_8859_6, UTF8</td>
</tr>
<tr>
<td>ISO_8859_7</td>
<td>ISO_8859_7, UTF8</td>
</tr>
<tr>
<td>ISO_8859_8</td>
<td>ISO_8859_8, UTF8</td>
</tr>
<tr>
<td>JOHAB</td>
<td>JOHAB, UTF8</td>
</tr>
<tr>
<td>KOI8</td>
<td>KOI8, ISO_8859_5, MULE_INTERNAL, UTF8, WIN866, WIN1251</td>
</tr>
<tr>
<td>LATIN1</td>
<td>LATIN1, MULE_INTERNAL, UTF8</td>
</tr>
<tr>
<td>LATIN2</td>
<td>LATIN2, MULE_INTERNAL, UTF8, WIN1250</td>
</tr>
<tr>
<td>LATIN3</td>
<td>LATIN3, MULE_INTERNAL, UTF8</td>
</tr>
<tr>
<td>LATIN4</td>
<td>LATIN4, MULE_INTERNAL, UTF8</td>
</tr>
<tr>
<td>LATIN5</td>
<td>LATIN5, UTF8</td>
</tr>
<tr>
<td>LATIN6</td>
<td>LATIN6, UTF8</td>
</tr>
<tr>
<td>LATIN7</td>
<td>LATIN7, UTF8</td>
</tr>
<tr>
<td>LATIN8</td>
<td>LATIN8, UTF8</td>
</tr>
<tr>
<td>LATIN9</td>
<td>LATIN9, UTF8</td>
</tr>
<tr>
<td>LATIN10</td>
<td>LATIN10, UTF8</td>
</tr>
<tr>
<td>MULE_INTERNAL</td>
<td>MULE_INTERNAL, BIG5, EUC_CN, EUC_JP, EUC_KR, EUC_TW, ISO_8859_5, KOI8, LATIN1 to LATIN4, SJIS, WIN866, WIN1250, WIN1251</td>
</tr>
<tr>
<td>SJIS</td>
<td>not supported as a server encoding</td>
</tr>
<tr>
<td>SQL_ASCII</td>
<td>not supported as a server encoding</td>
</tr>
<tr>
<td>UHC</td>
<td>not supported as a server encoding</td>
</tr>
<tr>
<td>UTF8</td>
<td>all supported encodings</td>
</tr>
<tr>
<td>WIN866</td>
<td>WIN866</td>
</tr>
<tr>
<td>ISO_8859_5</td>
<td>KOI8, MULE_INTERNAL, UTF8, WIN1251</td>
</tr>
<tr>
<td>WIN874</td>
<td>WIN874, UTF8</td>
</tr>
<tr>
<td>WIN1250</td>
<td>WIN1250, LATIN2, MULE_INTERNAL, UTF8</td>
</tr>
<tr>
<td>WIN1251</td>
<td>WIN1251, ISO_8859_5, KOI8, MULE_INTERNAL, UTF8, WIN866</td>
</tr>
<tr>
<td>WIN1252</td>
<td>WIN1252, UTF8</td>
</tr>
<tr>
<td>WIN1253</td>
<td>WIN1253, UTF8</td>
</tr>
<tr>
<td>WIN1254</td>
<td>WIN1254, UTF8</td>
</tr>
<tr>
<td>WIN1255</td>
<td>WIN1255, UTF8</td>
</tr>
<tr>
<td>WIN1256</td>
<td>WIN1256, UTF8</td>
</tr>
</tbody>
</table>
To enable automatic character set conversion, you have to tell Greenplum Database the character set (encoding) you would like to use in the client. There are several ways to accomplish this:

- **Using the \encoding command in psql**, which allows you to change client encoding on the fly.
- **Using SET client_encoding TO**. Setting the client encoding can be done with this SQL command:

  ```sql
  => SET CLIENT_ENCODING TO 'value';
  ```

  To query the current client encoding:

  ```sql
  => SHOW client_encoding;
  ```

  To return to the default encoding:

  ```sql
  => RESET client_encoding;
  ```

- **Using the PGCLIENTENCODING environment variable**. When PGCLIENTENCODING is defined in the client’s environment, that client encoding is automatically selected when a connection to the server is made. (This can subsequently be overridden using any of the other methods mentioned above.)

- **Setting the configuration parameter client_encoding**. If client_encoding is set in the master postgresql.conf file, that client encoding is automatically selected when a connection to Greenplum Database is made. (This can subsequently be overridden using any of the other methods mentioned above.)

If the conversion of a particular character is not possible — suppose you chose EUC_JP for the server and LATIN1 for the client, then some Japanese characters do not have a representation in LATIN1 — then an error is reported.

If the client character set is defined as SQL_ASCII, encoding conversion is disabled, regardless of the server’s character set. The use of SQL_ASCII is unwise unless you are working with all-ASCII data. SQL_ASCII is not supported as a server encoding.
Initializing a Greenplum Database System

Describes how to initialize a Greenplum Database database system.

The instructions in this chapter assume you have already installed the Greenplum Database software on all of the hosts in the system according to the instructions in Configuring Your Systems and Installing Greenplum.

This chapter contains the following topics:

• Overview
• Initializing Greenplum Database
• Next Steps

Overview

Because Greenplum Database is distributed, the process for initializing a Greenplum Database management system (DBMS) involves initializing several individual PostgreSQL database instances (called segment instances in Greenplum).

Each database instance (the master and all segments) must be initialized across all of the hosts in the system in such a way that they can all work together as a unified DBMS. Greenplum provides its own version of initdb called gpinitsystem, which takes care of initializing the database on the master and on each segment instance, and starting each instance in the correct order.

After the Greenplum Database database system has been initialized and started, you can then create and manage databases as you would in a regular PostgreSQL DBMS by connecting to the Greenplum master.

Initializing Greenplum Database

These are the high-level tasks for initializing Greenplum Database:

1. Make sure you have completed all of the installation tasks described in Configuring Your Systems and Installing Greenplum.
2. Create a host file that contains the host addresses of your segments. See Creating the Initialization Host File.
4. By default, Greenplum Database will be initialized using the locale of the master host system. Make sure this is the correct locale you want to use, as some locale options cannot be changed after initialization. See Configuring Localization Settings for more information.
5. Run the Greenplum Database initialization utility on the master host. See Running the Initialization Utility.

Creating the Initialization Host File

The gpinitsystem utility requires a host file that contains the list of addresses for each segment host. The initialization utility determines the number of segment instances per host by the number host addresses listed per host times the number of data directory locations specified in the gpinitsystem_config file.

This file should only contain segment host addresses (not the master or standby master). For segment machines with multiple, unbonded network interfaces, this file should list the host address names for each interface — one per line.

Note: The Greenplum Database segment host naming convention is sdwN where sdw is a prefix and N is an integer. For example, sdw2 and so on. If hosts have multiple unbonded NICs, the
convention is to append a dash (\-) and number to the host name. For example, sdw1-1 and sdw1-2 are the two interface names for host sdw1. However, NIC bonding is recommended to create a load-balanced, fault-tolerant network.

**To create the initialization host file**

1. Log in as gpadmin.

   ```
   $ su - gpadmin
   ```

2. Create a file named `hostfile_gpinitsystem`. In this file add the host address name(s) of your segment host interfaces, one name per line, no extra lines or spaces. For example, if you have four segment hosts with two unbonded network interfaces each:

   ```
   sdw1-1
   sdw1-2
   sdw2-1
   sdw2-2
   sdw3-1
   sdw3-2
   sdw4-1
   sdw4-2
   ```

3. Save and close the file.

   **Note:** If you are not sure of the host names and/or interface address names used by your machines, look in the `/etc/hosts` file.

**Creating the Greenplum Database Configuration File**

Your Greenplum Database configuration file tells the `gpinitsystem` utility how you want to configure your Greenplum Database system. An example configuration file can be found in `$GPHOME/docs/cli_help/gpconfigs/gpinitsystem_config`.

**To create a gpinitsystem_config file**

1. Log in as gpadmin.

   ```
   $ su - gpadmin
   ```

2. Make a copy of the `gpinitsystem_config` file to use as a starting point. For example:

   ```
   $ cp $GPHOME/docs/cli_help/gpconfigs/gpinitsystem_config \/home/gpadmin/gpconfigs/gpinitsystem_config
   ```

3. Open the file you just copied in a text editor.

   Set all of the required parameters according to your environment. See `gpinitsystem` for more information. A Greenplum Database system must contain a master instance and at least two segment instances (even if setting up a single node system).

   **The `DATA_DIRECTORY` parameter is what determines how many segments per host will be created. If your segment hosts have multiple network interfaces, and you used their interface address names in your host file, the number of segments will be evenly spread over the number of available interfaces.**

   Here is an example of the **required parameters** in the `gpinitsystem_config` file:

   ```
   ARRAY_NAME="EMC Greenplum DW"
   SEG_PREFIX=gpseg
   PORT_BASE=40000
   declare -a DATA_DIRECTORY=(/data1/primary /data1/primary /data1/primary /data2/primary /data2/primary /data2/primary)
   ```
4. (Optional) If you want to deploy mirror segments, uncomment and set the mirroring parameters according to your environment. Here is an example of the optional mirror parameters in the `gpinitsystem_config` file:

```plaintext
MIRROR_PORT_BASE=50000
REPLICATION_PORT_BASE=41000
MIRROR_REPLICATION_PORT_BASE=51000
declare -a MIRROR_DATA_DIRECTORY=(/data1/mirror /data1/mirror /data1/mirror /data2/mirror /data2/mirror /data2/mirror)
```

**Note:** You can initialize your Greenplum system with primary segments only and deploy mirrors later using the `gpaddmirrors` utility.

5. Save and close the file.

### Running the Initialization Utility

The `gpinitsystem` utility will create a Greenplum Database system using the values defined in the configuration file.

#### To run the initialization utility

1. Run the following command referencing the path and file name of your initialization configuration file (`gpinitsystem_config`) and host file (`hostfile_gpinitsystem`). For example:

   ```bash
   $ cd ~
   $ gpinitsystem -c gpconfigs/gpinitsystem_config -h gpconfigs/hostfile_gpinitsystem
   ...
   ```

   For a fully redundant system (with a standby master and a *spread* mirror configuration) include the `-s` and `-S` options. For example:

   ```bash
   $ gpinitsystem -c gpconfigs/gpinitsystem_config -h gpconfigs/hostfile_gpinitsystem \
   -s standby_master_hostname -S
   ...
   ```

2. The utility will verify your setup information and make sure it can connect to each host and access the data directories specified in your configuration. If all of the pre-checks are successful, the utility will prompt you to confirm your configuration. For example:

   ```bash
   => Continue with Greenplum creation? Yy/Nn
   ```

3. Press `y` to start the initialization.

4. The utility will then begin setup and initialization of the master instance and each segment instance in the system. Each segment instance is set up in parallel. Depending on the number of segments, this process can take a while.

5. At the end of a successful setup, the utility will start your Greenplum Database system. You should see:

   ```bash
   => Greenplum Database instance successfully created.
   ```
Troubleshooting Initialization Problems

If the utility encounters any errors while setting up an instance, the entire process will fail, and could possibly leave you with a partially created system. Refer to the error messages and logs to determine the cause of the failure and where in the process the failure occurred. Log files are created in `~/gpAdminLogs`.

Depending on when the error occurred in the process, you may need to clean up and then try the gpinitsystem utility again. For example, if some segment instances were created and some failed, you may need to stop `postgres` processes and remove any utility-created data directories from your data storage area(s). A backout script is created to help with this cleanup if necessary.

Using the Backout Script

If the gpinitsystem utility fails, it will create the following backout script if it has left your system in a partially installed state:

`~/gpAdminLogs/backout_gpinitsystem_<user>_<timestamp>`

You can use this script to clean up a partially created Greenplum Database system. This backout script will remove any utility-created data directories, `postgres` processes, and log files. After correcting the error that caused gpinitsystem to fail and running the backout script, you should be ready to retry initializing your Greenplum Database array.

The following example shows how to run the backout script:

```
$ sh backout_gpinitsystem_gpadmin_20071031_121053
```

Setting Greenplum Environment Variables

You must configure your environment on the Greenplum Database master (and standby master). A `greenplum_path.sh` file is provided in your `$GPHOME` directory with environment variable settings for Greenplum Database. You can source this file in the `gpadmin` user's startup shell profile (such as `.bashrc`).

The Greenplum Database management utilities also require that the `MASTER_DATA_DIRECTORY` environment variable be set. This should point to the directory created by the gpinitsystem utility in the master data directory location.

Note: The `greenplum_path.sh` script changes the operating environment in order to support running the Greenplum Database-specific utilities. These same changes to the environment can negatively affect the operation of other system-level utilities, such as `ps` or `yum`. Use separate accounts for performing system administration and database administration, instead of attempting to perform both functions as `gpadmin`.

To set up your user environment for Greenplum

1. Make sure you are logged in as `gpadmin`:

```
$ su - gpadmin
```

2. Open your profile file (such as `.bashrc`) in a text editor. For example:

```
$ vi ~/.bashrc
```

3. Add lines to this file to source the `greenplum_path.sh` file and set the `MASTER_DATA_DIRECTORY` environment variable. For example:

```
source /usr/local/greenplum-db/greenplum_path.sh
```
export MASTER_DATA_DIRECTORY=/data/master/gpseg-1

4. (Optional) You may also want to set some client session environment variables such as PGPORT, PGUSER and PGDATABASE for convenience. For example:

   export PGPORT=5432
   export PGUSER=gpadmin export PGDATABASE=default_login_database_name

5. (Optional) If you use RHEL 7 or CentOS 7, add the following line to the end of the .bashrc file to enable using the ps command in the greenplum_path.sh environment:

   export LD_PRELOAD=/lib64/libz.so.1 ps

6. Save and close the file.

7. After editing the profile file, source it to make the changes active. For example:

   $ source ~/.bashrc

8. If you have a standby master host, copy your environment file to the standby master as well. For example:

   $ cd ~
   $ scp .bashrc standby_hostname:`pwd`

   Note: The .bashrc file should not produce any output. If you wish to have a message display to users upon logging in, use the .profile file instead.

Next Steps

After your system is up and running, the next steps are:

- Allowing Client Connections
- Creating Databases and Loading Data

Allowing Client Connections

After a Greenplum Database is first initialized it will only allow local connections to the database from the gpadmin role (or whatever system user ran gpinitsystem). If you would like other users or client machines to be able to connect to Greenplum Database, you must give them access. See the Greenplum Database Administrator Guide for more information.

Creating Databases and Loading Data

After verifying your installation, you may want to begin creating databases and loading data. See Defining Database Objects and Loading and Unloading Data in the Greenplum Database Administrator Guide for more information about creating databases, schemas, tables, and other database objects in Greenplum Database and loading your data.
# Installation Management Utilities

References for the command-line management utilities used to install and initialize a Greenplum Database system.

For a full reference of all Greenplum Database utilities, see the *Greenplum Database Utility Guide*.

The following Greenplum Database management utilities are located in `$GPHOME/bin`.

<table>
<thead>
<tr>
<th>Utilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpactivatestandby</td>
</tr>
<tr>
<td>gpaddmirrors</td>
</tr>
<tr>
<td>gpcheck (deprecated)</td>
</tr>
<tr>
<td>gpcheckperf</td>
</tr>
<tr>
<td>gpdeleteesystem</td>
</tr>
<tr>
<td>gpinitstandby</td>
</tr>
<tr>
<td>gpinitesystem</td>
</tr>
<tr>
<td>gppkg</td>
</tr>
<tr>
<td>gpscp</td>
</tr>
<tr>
<td>gpseginstall</td>
</tr>
<tr>
<td>gpssh</td>
</tr>
<tr>
<td>gpssh-exkeys</td>
</tr>
<tr>
<td>gpstart</td>
</tr>
<tr>
<td>gpstop</td>
</tr>
<tr>
<td>gptransfer</td>
</tr>
</tbody>
</table>
Greenplum Environment Variables

Reference of the environment variables to set for Greenplum Database.
Set these in your user’s startup shell profile (such as ~/.bashrc or ~/.bash_profile), or in /etc/profile if you want to set them for all users.

**Required Environment Variables**

*Note:* GPHOME, PATH and LD_LIBRARY_PATH can be set by sourcing the greenplum_path.sh file from your Greenplum Database installation directory

**GPHOME**

This is the installed location of your Greenplum Database software. For example:

```
GPHOME=/usr/local/greenplum-db-4.3.x.x
export GPHOME
```

**PATH**

Your PATH environment variable should point to the location of the Greenplum Database bin directory. For example:

```
PATH=$GPHOME/bin:$PATH
export PATH
```

**LD_LIBRARY_PATH**

The LD_LIBRARY_PATH environment variable should point to the location of the Greenplum Database/PostgreSQL library files. For example:

```
LD_LIBRARY_PATH=$GPHOME/lib
export LD_LIBRARY_PATH
```

**MASTER_DATA_DIRECTORY**

This should point to the directory created by the gpinitsystem utility in the master data directory location. For example:

```
MASTER_DATA_DIRECTORY=/data/master/gpseg-1
export MASTER_DATA_DIRECTORY
```

**Optional Environment Variables**

The following are standard PostgreSQL environment variables, which are also recognized in Greenplum Database. You may want to add the connection-related environment variables to your profile for convenience, so you do not have to type so many options on the command line for client connections. Note that these environment variables should be set on the Greenplum Database master host only.
**PGAPPNAME**

The name of the application that is usually set by an application when it connects to the server. This name is displayed in the activity view and in log entries. The `PGAPPNAME` environmental variable behaves the same as the `application_name` connection parameter. The default value for `application_name` is `psql`. The name cannot be longer than 63 characters.

**PGDATABASE**

The name of the default database to use when connecting.

**PGHOST**

The Greenplum Database master host name.

**PGHOSTADDR**

The numeric IP address of the master host. This can be set instead of or in addition to `PGHOST` to avoid DNS lookup overhead.

**PGPASSWORD**

The password used if the server demands password authentication. Use of this environment variable is not recommended for security reasons (some operating systems allow non-root users to see process environment variables via `ps`). Instead consider using the `~/.pgpass` file.

**PGPASSFILE**

The name of the password file to use for lookups. If not set, it defaults to `~/.pgpass`. See the topic about *The Password File* in the PostgreSQL documentation for more information.

**PGOPTIONS**

Sets additional configuration parameters for the Greenplum Database master server.

**PGPORT**

The port number of the Greenplum Database server on the master host. The default port is 5432.

**PGUSER**

The Greenplum Database user name used to connect.

**PGDATESTYLE**

Sets the default style of date/time representation for a session. (Equivalent to `SET datestyle TO...`)

**PGTZ**

Sets the default time zone for a session. (Equivalent to `SET timezone TO...`)

**PGCLIENTENCODING**

Sets the default client character set encoding for a session. (Equivalent to `SET client_encoding TO...`)
Chapter 7

Greenplum Database Administrator Guide

Information about configuring, managing and monitoring Greenplum Database installations, and administering, monitoring, and working with databases. The guide also contains information about Greenplum Database architecture and concepts such as parallel processing.
Greenplum Database Concepts

This section provides an overview of Greenplum Database components and features such as high availability, parallel data loading features, and management utilities.

This section contains the following topics:

• About the Greenplum Architecture
• About Management and Monitoring Utilities
• About Parallel Data Loading
• About Redundancy and Failover in Greenplum Database
• About Database Statistics in Greenplum Database

About the Greenplum Architecture

Greenplum Database is a massively parallel processing (MPP) database server with an architecture specially designed to manage large-scale analytic data warehouses and business intelligence workloads.

MPP (also known as a shared nothing architecture) refers to systems with two or more processors that cooperate to carry out an operation, each processor with its own memory, operating system and disks. Greenplum uses this high-performance system architecture to distribute the load of multi-terabyte data warehouses, and can use all of a system's resources in parallel to process a query.

Greenplum Database is based on PostgreSQL open-source technology. It is essentially several PostgreSQL disk-oriented database instances acting together as one cohesive database management system (DBMS). It is based on PostgreSQL 8.3.23, and in most cases is very similar to PostgreSQL with regard to SQL support, features, configuration options, and end-user functionality. Database users interact with Greenplum Database as they would with a regular PostgreSQL DBMS.

Greenplum Database can use the append-optimized (AO) storage format for bulk loading and reading of data, and provides performance advantages over HEAP tables. Append-optimized storage provides checksums for data protection, compression and row/column orientation. Both row-oriented or column-oriented append-optimized tables can be compressed.

The main differences between Greenplum Database and PostgreSQL are as follows:

• GPORCA is leveraged for query planning, in addition to the legacy query planner, which is based on the Postgres query planner.
• Greenplum Database can use append-optimized storage.
• Greenplum Database has the option to use column storage, data that is logically organized as a table, using rows and columns that are physically stored in a column-oriented format, rather than as rows. Column storage can only be used with append-optimized tables. Column storage is compressible. It also can provide performance improvements as you only need to return the columns of interest to you. All compression algorithms can be used with either row or column-oriented tables, but Run-Length Encoded (RLE) compression can only be used with column-oriented tables. Greenplum Database provides compression on all Append-Optimized tables that use column storage.

The internals of PostgreSQL have been modified or supplemented to support the parallel structure of Greenplum Database. For example, the system catalog, optimizer, query executor, and transaction manager components have been modified and enhanced to be able to execute queries simultaneously across all of the parallel PostgreSQL database instances. The Greenplum interconnect (the networking layer) enables communication between the distinct PostgreSQL instances and allows the system to behave as one logical database.

Greenplum Database also can use declarative partitions and sub-partitions to implicitly generate partition constraints.
Greenplum Database also includes features designed to optimize PostgreSQL for business intelligence (BI) workloads. For example, Greenplum has added parallel data loading (external tables), resource management, query optimizations, and storage enhancements, which are not found in standard PostgreSQL. Many features and optimizations developed by Greenplum make their way into the PostgreSQL community. For example, table partitioning is a feature first developed by Greenplum, and it is now in standard PostgreSQL.

Greenplum Database queries use a Volcano-style query engine model, where the execution engine takes an execution plan and uses it to generate a tree of physical operators, evaluates tables through physical operators, and delivers results in a query response.

Greenplum Database stores and processes large amounts of data by distributing the data and processing workload across several servers or hosts. Greenplum Database is an array of individual databases based upon PostgreSQL 8.3 working together to present a single database image. The master is the entry point to the Greenplum Database system. It is the database instance to which clients connect and submit SQL statements. The master coordinates its work with the other database instances in the system, called segments, which store and process the data.

![Figure 10: High-Level Greenplum Database Architecture](image)

The following topics describe the components that make up a Greenplum Database system and how they work together.

**About the Greenplum Master**

The Greenplum Database master is the entry to the Greenplum Database system, accepting client connections and SQL queries, and distributing work to the segment instances.

Greenplum Database end-users interact with Greenplum Database (through the master) as they would with a typical PostgreSQL database. They connect to the database using client programs such as psql or application programming interfaces (APIs) such as JDBC, ODBC or libpq (the PostgreSQL C API).
The master is where the *global system catalog* resides. The global system catalog is the set of system
tables that contain metadata about the Greenplum Database system itself. The master does not contain
any user data; data resides only on the *segments*. The master authenticates client connections, processes
incoming SQL commands, distributes workloads among segments, coordinates the results returned by
each segment, and presents the final results to the client program.

Greenplum Database uses Write-Ahead Logging (WAL) for master/standby master mirroring. In WAL-
based logging, all modifications are written to the log before being applied, to ensure data integrity for any
in-process operations.

**Note:** WAL logging is not yet available for segment mirroring.

### About the Greenplum Segments

Greenplum Database segment instances are independent PostgreSQL databases that each store a portion
of the data and perform the majority of query processing.

When a user connects to the database via the Greenplum master and issues a query, processes are
created in each segment database to handle the work of that query. For more information about query
processes, see About Greenplum Query Processing.

User-defined tables and their indexes are distributed across the available segments in a Greenplum
Database system; each segment contains a distinct portion of data. The database server processes that
serve segment data run under the corresponding segment instances. Users interact with segments in a
Greenplum Database system through the master.

Segments run on a servers called *segment hosts*. A segment host typically executes from two to eight
Greenplum segments, depending on the CPU cores, RAM, storage, network interfaces, and workloads.
Segment hosts are expected to be identically configured. The key to obtaining the best performance from
Greenplum Database is to distribute data and workloads *evenly* across a large number of equally capable
segments so that all segments begin working on a task simultaneously and complete their work at the
same time.

### About the Greenplum Interconnect

The interconect is the networking layer of the Greenplum Database architecture.

The *interconnect* refers to the inter-process communication between segments and the network
infrastructure on which this communication relies. The Greenplum interconnect uses a standard Ethernet
switching fabric. For performance reasons, a 10-Gigabit system, or faster, is recommended.

By default, the interconnect uses User Datagram Protocol with flow control (UDPIFC) for interconnect
traffic to send messages over the network. The Greenplum software performs packet verification beyond
what is provided by UDP. This means the reliability is equivalent to Transmission Control Protocol (TCP),
and the performance and scalability exceeds TCP. If the interconnect is changed to TCP, Greenplum
Database has a scalability limit of 1000 segment instances. With UDPIFC as the default protocol for the
interconnect, this limit is not applicable.

### About Management and Monitoring Utilities

Greenplum Database provides standard command-line utilities for performing common monitoring and
administration tasks.

Greenplum command-line utilities are located in the *@GPHOME/bin* directory and are executed on the
master host. Greenplum provides utilities for the following administration tasks:

- Installing Greenplum Database on an array
- Initializing a Greenplum Database System
- Starting and stopping Greenplum Database
- Adding or removing a host
- Expanding the array and redistributing tables among new segments
• Managing recovery for failed segment instances
• Managing failover and recovery for a failed master instance
• Backing up and restoring a database (in parallel)
• Loading data in parallel
• Transferring data between Greenplum databases
• System state reporting

Greenplum Database includes an optional performance management database that contains query status information and system metrics. The `gpperfmon_install` management utility creates the database, named `gpperfmon`, and enables data collection agents that execute on the Greenplum Database master and segment hosts. Data collection agents on the segment hosts collect query status from the segments, as well as system metrics such as CPU and memory utilization. An agent on the master host periodically (typically every 15 seconds) retrieves the data from the segment host agents and updates the `gpperfmon` database. Users can query the `gpperfmon` database to see the query and system metrics.

Pivotal provides an optional system monitoring and management tool, Greenplum Command Center, which administrators can install and enable with Greenplum Database. Greenplum Command Center, which depends upon the `gpperfmon` database, provides a web-based user interface for viewing the system metrics and allows administrators to perform additional system management tasks. For more information about Greenplum Command Center, see the Greenplum Command Center documentation.

![Figure 11: Greenplum Command Center Architecture](image)

**About Concurrency Control in Greenplum Database**

Greenplum Database uses the PostgreSQL Multiversion Concurrency Control (MVCC) model to manage concurrent transactions for heap tables.

Concurrency control in a database management system allows concurrent queries to complete with correct results while ensuring the integrity of the database. Traditional databases use a two-phase locking protocol that prevents a transaction from modifying data that has been read by another concurrent transaction and prevents any concurrent transaction from reading or writing data that another transaction has updated.
The locks required to coordinate transactions add contention to the database, reducing overall transaction throughput.

Greenplum Database uses the PostgreSQL Multiversion Concurrency Control (MVCC) model to manage concurrency for heap tables. With MVCC, each query operates on a snapshot of the database when the query starts. While it executes, a query cannot see changes made by other concurrent transactions. This ensures that a query sees a consistent view of the database. Queries that read rows can never block waiting for transactions that write rows. Conversely, queries that write rows cannot be blocked by transactions that read rows. This allows much greater concurrency than traditional database systems that employ locks to coordinate access between transactions that read and write data.

**Note:** Append-optimized tables are managed with a different concurrency control model than the MVCC model discussed in this topic. They are intended for “write-once, read-many” applications that never, or only very rarely, perform row-level updates.

**Snapshots**

The MVCC model depends on the system's ability to manage multiple versions of data rows. A query operates on a snapshot of the database at the start of the query. A snapshot is the set of rows that are visible at the beginning of a statement or transaction. The snapshot ensures the query has a consistent and valid view of the database for the duration of its execution.

Each transaction is assigned a unique transaction ID (XID), an incrementing 32-bit value. When a new transaction starts, it is assigned the next XID. An SQL statement that is not enclosed in a transaction is treated as a single-statement transaction—the BEGIN and COMMIT are added implicitly. This is similar to autocommit in some database systems.

**Note:** Greenplum Database assigns XID values only to transactions that involve DDL or DML operations, which are typically the only transactions that require an XID.

When a transaction inserts a row, the XID is saved with the row in the xmin system column. When a transaction deletes a row, the XID is saved in the xmax system column. Updating a row is treated as a delete and an insert, so the XID is saved to the xmax of the current row and the xmin of the newly inserted row. The xmin and xmax columns, together with the transaction completion status, specify a range of transactions for which the version of the row is visible. A transaction can see the effects of all transactions less than xmin, which are guaranteed to be committed, but it cannot see the effects of any transaction greater than or equal to xmax.

Multi-statement transactions must also record which command within a transaction inserted a row (cmin) or deleted a row (cmax) so that the transaction can see changes made by previous commands in the transaction. The command sequence is only relevant during the transaction, so the sequence is reset to 0 at the beginning of a transaction.

XID is a property of the database. Each segment database has its own XID sequence that cannot be compared to the XIDs of other segment databases. The master coordinates distributed transactions with the segments using a cluster-wide session ID number, called gp_session_id. The segments maintain a mapping of distributed transaction IDs with their local XIDs. The master coordinates distributed transactions across all of the segments with the two-phase commit protocol. If a transaction fails on any one segment, it is rolled back on all segments.

You can see the xmin, xmax, cmin, and cmax columns for any row with a SELECT statement:

```
SELECT xmin, xmax, cmin, cmax, * FROM tablename;
```

Because you run the SELECT command on the master, the XIDs are the distributed transactions IDs. If you could execute the command in an individual segment database, the xmin and xmax values would be the segment's local XIDs.
Transaction ID Wraparound

The MVCC model uses transaction IDs (XIDs) to determine which rows are visible at the beginning of a query or transaction. The XID is a 32-bit value, so a database could theoretically execute over four billion transactions before the value overflows and wraps to zero. However, Greenplum Database uses modulo $2^{32}$ arithmetic with XIDs, which allows the transaction IDs to wrap around, much as a clock wraps at twelve o’clock. For any given XID, there could be about two billion past XIDs and two billion future XIDs. This works until a version of a row persists through about two billion transactions, when it suddenly appears to be a new row. To prevent this, Greenplum has a special XID, called FrozenXID, which is always considered older than any regular XID it is compared with. The $\text{xmin}$ of a row must be replaced with FrozenXID within two billion transactions, and this is one of the functions the VACUUM command performs.

Vacuuming the database at least every two billion transactions prevents XID wraparound. Greenplum Database monitors the transaction ID and warns if a VACUUM operation is required.

A warning is issued when a significant portion of the transaction IDs are no longer available and before transaction ID wraparound occurs:

```
WARNING: database "database_name" must be vacuumed
within number_of_transactions transactions
```

When the warning is issued, a VACUUM operation is required. If a VACUUM operation is not performed, Greenplum Database stops creating transactions to avoid possible data loss when it reaches a limit prior to when transaction ID wraparound occurs and issues this error:

```
FATAL: database is not accepting commands to avoid wraparound data loss in
database "database_name"
```

See Recovering from a Transaction ID Limit Error for the procedure to recover from this error.

The server configuration parameters xid_warn_limit and xid_stop_limit control when the warning and error are displayed. The xid_warn_limit parameter specifies the number of transaction IDs before the xid_stop_limit when the warning is issued. The xid_stop_limit parameter specifies the number of transaction IDs before wraparound would occur when the error is issued and new transactions cannot be created.

Transaction Isolation Modes

The SQL standard describes three phenomena that can occur when database transactions run concurrently:

- **Dirty read** – a transaction can read uncommitted data from another concurrent transaction.
- **Non-repeatable read** – a row read twice in a transaction can change because another concurrent transaction committed changes after the transaction began.
- **Phantom read** – a query executed twice in the same transaction can return two different sets of rows because another concurrent transaction added rows.

The SQL standard defines four transaction isolation modes that database systems must support:

<table>
<thead>
<tr>
<th>Level</th>
<th>Dirty Read</th>
<th>Non-Repeatable</th>
<th>Phantom Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Uncommitted</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Read Committed</td>
<td>Impossible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Repeatable Read</td>
<td>Impossible</td>
<td>Impossible</td>
<td>Possible</td>
</tr>
<tr>
<td>Serializable</td>
<td>Impossible</td>
<td>Impossible</td>
<td>Impossible</td>
</tr>
</tbody>
</table>
The Greenplum Database SQL commands allow you to request \texttt{READ UNCOMMITTED}, \texttt{READ COMMITTED}, or \texttt{SERIALIZABLE}. Greenplum Database treats \texttt{READ UNCOMMITTED} the same as \texttt{READ COMMITTED}. Requesting \texttt{REPEATABLE READ} produces an error; use \texttt{SERIALIZABLE} instead. The default isolation mode is \texttt{READ COMMITTED}.

The difference between \texttt{READ COMMITTED} and \texttt{SERIALIZABLE} is that in \texttt{READ COMMITTED} mode, each statement in a transaction sees only rows committed before the statement started, while in \texttt{SERIALIZABLE} mode, all statements in a transaction see only rows committed before the transaction started.

The \texttt{READ COMMITTED} isolation mode permits greater concurrency and better performance than the \texttt{SERIALIZABLE} mode. It allows \textit{non-repeatable reads}, where the values in a row retrieved twice in a transaction can differ because another concurrent transaction has committed changes since the transaction began. \texttt{READ COMMITTED} mode also permits \textit{phantom reads}, where a query executed twice in the same transaction can return two different sets of rows.

The \texttt{SERIALIZABLE} isolation mode prevents both non-repeatable reads and phantom reads, but at the cost of concurrency and performance. Each concurrent transaction has a consistent view of the database taken at the beginning of execution. A concurrent transaction that attempts to modify data modified by another transaction is rolled back. Applications that execute transactions in \texttt{SERIALIZABLE} mode must be prepared to handle transactions that fail due to serialization errors. If \texttt{SERIALIZABLE} isolation mode is not required by the application, it is better to use \texttt{READ COMMITTED} mode.

The SQL standard specifies that concurrent serializable transactions produce the same database state they would produce if executed sequentially. The MVCC snapshot isolation model prevents dirty reads, non-repeatable reads, and phantom reads without expensive locking, but there are other interactions that can occur between some \texttt{SERIALIZABLE} transactions in Greenplum Database that prevent them from being truly serializable. These anomalies can often be attributed to the fact that Greenplum Database does not perform \textit{predicate locking}, which means that a write in one transaction can affect the result of a previous read in another concurrent transaction.

Transactions that run concurrently should be examined to identify interactions that are not prevented by disallowing concurrent updates of the same data. Problems identified can be prevented by using explicit table locks or by requiring the conflicting transactions to update a dummy row introduced to represent the conflict.

The SQL \texttt{SET TRANSACTION ISOLATION LEVEL} statement sets the isolation mode for the current transaction. The mode must be set before any \texttt{SELECT}, \texttt{INSERT}, \texttt{DELETE}, \texttt{UPDATE}, or \texttt{COPY} statements:

\begin{verbatim}
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
...
COMMIT;
\end{verbatim}

The isolation mode can also be specified as part of the \texttt{BEGIN} statement:

\begin{verbatim}
BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
\end{verbatim}

The default transaction isolation mode can be changed for a session by setting the \texttt{default_transaction_isolation} configuration property.

### Removing Dead Rows from Tables

Updating or deleting a row leaves an expired version of the row in the table. When an expired row is no longer referenced by any active transactions, it can be removed and the space it occupied can be reused. The \texttt{VACUUM} command marks the space used by expired rows for reuse.

When expired rows accumulate in a table, the disk files must be extended to accommodate new rows. Performance suffers due to the increased disk I/O required to execute queries. This condition is called \textit{bloat} and it should be managed by regularly vacuuming tables.
The **VACUUM** command (without **FULL**) can run concurrently with other queries. It marks the space previously used by the expired rows as free. If the amount of remaining free space is significant, it adds the page to the table's free space map. When Greenplum Database later needs space for new rows, it first consults the table's free space map to find pages with available space. If none are found, new pages will be appended to the file.

**VACUUM (without FULL)** does not consolidate pages or reduce the size of the table on disk. The space it recovers is only available through the free space map. To prevent disk files from growing, it is important to run **VACUUM** often enough. The frequency of required **VACUUM** runs depends on the frequency of updates and deletes in the table (inserts only ever add new rows). Heavily updated tables might require several **VACUUM** runs per day, to ensure that the available free space can be found through the free space map. It is also important to run **VACUUM** after running a transaction that updates or deletes a large number of rows.

The **VACUUM FULL** command rewrites the table without expired rows, reducing the table to its minimum size. Every page in the table is checked, and visible rows are moved up into pages which are not yet fully packed. Empty pages are discarded. The table is locked until **VACUUM FULL** completes. This is very expensive compared to the regular **VACUUM** command, and can be avoided or postponed by vacuuming regularly. It is best to run **VACUUM FULL** during a maintenance period. An alternative to **VACUUM FULL** is to recreate the table with a **CREATE TABLE AS** statement and then drop the old table.

The free space map resides in shared memory and keeps track of free space for all tables and indexes. Each table or index uses about 60 bytes of memory and each page with free space consumes six bytes. Two system configuration parameters configure the size of the free space map:

**max_fsm_pages**

Sets the maximum number of disk pages that can be added to the shared free space map. Six bytes of shared memory are consumed for each page slot. The default is 200000. This parameter must be set to at least 16 times the value of **max_fsm_relations**.

**max_fsm_relations**

Sets the maximum number of relations that will be tracked in the shared memory free space map. This parameter should be set to a value larger than the total number of **tables + indexes + system tables**. The default is 1000. About 60 bytes of memory are consumed for each relation per segment instance. It is better to set the parameter too high than too low.

If the free space map is undersized, some disk pages with available space will not be added to the map, and that space cannot be reused until at least the next **VACUUM** command runs. This causes files to grow.

You can run **VACUUM VERBOSE** **tablename** to get a report, by segment, of the number of dead rows removed, the number of pages affected, and the number of pages with usable free space.

Query the **pg_class** system table to find out how many pages a table is using across all segments. Be sure to **ANALYZE** the table first to get accurate data.

```sql
SELECT relname, relpages, reltuples FROM pg_class WHERE relname='tablename';
```

Another useful tool is the **gp_bloat_diag** view in the **gp_toolkit** schema, which identifies bloat in tables by comparing the actual number of pages used by a table to the expected number. See "The **gp_toolkit** Administrative Schema" in the Greenplum Database Reference Guide for more about **gp_bloat_diag**.

### Example of Managing Transaction IDs

For Greenplum Database, the transaction ID (XID) value an incrementing 32-bit \(2^{32}\) value. The maximum unsigned 32-bit value is 4,294,967,295, or about four billion. The XID values restart at 0 after the maximum is reached. Greenplum Database handles the limit of XID values with two features:
Calculations on XID values using modulo-\(2^{32}\) arithmetic that allow Greenplum Database to reuse XID values. The modulo calculations determine the order of transactions, whether one transaction has occurred before or after another, based on the XID.

Every XID value can have up to two billion (\(2^{31}\)) XID values that are considered previous transactions and two billion (\(2^{31} - 1\)) XID values that are considered newer transactions. The XID values can be considered a circular set of values with no endpoint similar to a 24 hour clock.

Using the Greenplum Database modulo calculations, as long as two XIDs are within \(2^{31}\) transactions of each other, comparing them yields the correct result.

A frozen XID value that Greenplum Database uses as the XID for current (visible) data rows. Setting a row's XID to the frozen XID performs two functions.

- When Greenplum Database compares XIDs using the modulo calculations, the frozen XID is always smaller, earlier, when compared to any other XID. If a row's XID is not set to the frozen XID and \(2^{31}\) new transactions are executed, the row appears to be executed in the future based on the modulo calculation.
- When the row's XID is set to the frozen XID, the original XID can be used, without duplicating the XID. This keeps the number of data rows on disk with assigned XIDs below \(2^{32}\).

**Note:** Greenplum Database assigns XID values only to transactions that involve DDL or DML operations, which are typically the only transactions that require an XID.

**Simple MVCC Example**

This is a simple example of the concepts of a MVCC database and how it manages data and transactions with transaction IDs. This simple MVCC database example consists of a single table:

- The table is a simple table with 2 columns and 4 rows of data.
- The valid transaction ID (XID) values are from 0 up to 9, after 9 the XID restarts at 0.
- The frozen XID is -2. This is different than the Greenplum Database frozen XID.
- Transactions are performed on a single row.
- Only insert and update operations are performed.
- All updated rows remain on disk, no operations are performed to remove obsolete rows.

The example only updates the amount values. No other changes to the table.

The example shows these concepts.

- **How transaction IDs are used to manage multiple, simultaneous transactions on a table.**
- **How transaction IDs are managed with the frozen XID**
- **How the modulo calculation determines the order of transactions based on transaction IDs**

**Managing Simultaneous Transactions**

This table is the initial table data on disk with no updates. The table contains two database columns for transaction IDs, \(x_{\text{min}}\) (transaction that created the row) and \(x_{\text{max}}\) (transaction that updated the row). In the table, changes are added, in order, to the bottom of the table.

**Table 13: Example Table**

<table>
<thead>
<tr>
<th>item</th>
<th>amount</th>
<th>(x_{\text{min}})</th>
<th>(x_{\text{max}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget</td>
<td>100</td>
<td>0</td>
<td>null</td>
</tr>
<tr>
<td>giblet</td>
<td>200</td>
<td>1</td>
<td>null</td>
</tr>
<tr>
<td>sprocket</td>
<td>300</td>
<td>2</td>
<td>null</td>
</tr>
<tr>
<td>gizmo</td>
<td>400</td>
<td>3</td>
<td>null</td>
</tr>
</tbody>
</table>
The next table shows the table data on disk after some updates on the amount values have been performed.

\[
\begin{align*}
\text{xid} &= 4: \text{update tbl set amount}=208 \text{ where item} = '\text{widget}' \\
\text{xid} &= 5: \text{update tbl set amount}=133 \text{ where item} = '\text{sprocket}' \\
\text{xid} &= 6: \text{update tbl set amount}=16 \text{ where item} = '\text{widget}'
\end{align*}
\]

In the next table, the bold items are the current rows for the table. The other rows are obsolete rows, table data that on disk but is no longer current. Using the xmax value, you can determine the current rows of the table by selecting the rows with null value. Greenplum Database uses a slightly different method to determine current table rows.

**Table 14: Example Table with Updates**

<table>
<thead>
<tr>
<th>item</th>
<th>amount</th>
<th>xmin</th>
<th>xmax</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget</td>
<td>100</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>giblet</td>
<td>200</td>
<td>1</td>
<td>null</td>
</tr>
<tr>
<td>sprocket</td>
<td>300</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>gizmo</td>
<td>400</td>
<td>3</td>
<td>null</td>
</tr>
<tr>
<td>widget</td>
<td>208</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>sprocket</td>
<td>133</td>
<td>5</td>
<td>null</td>
</tr>
<tr>
<td>widget</td>
<td>16</td>
<td>6</td>
<td>null</td>
</tr>
</tbody>
</table>

The simple MVCC database works with XID values to determine the state of the table. For example, both these independent transactions execute concurrently.

- **UPDATE** command changes the sprocket amount value to 133 (xmin value 5)
- **SELECT** command returns the value of sprocket.

During the **UPDATE** transaction, the database returns the value of sprocket 300, until the **UPDATE** transaction completes.

### Managing XIDs and the Frozen XID

For this simple example, the database is close to running out of available XID values. When Greenplum Database is close to running out of available XID values, Greenplum Database takes these actions.

- Greenplum Database issues a warning stating that the database is running out of XID values.

```
WARNING: database "database_name" must be vacuumed within number_of_transactions transactions
```

- Before the last XID is assigned, Greenplum Database stops accepting transactions to prevent assigning an XID value twice and issues this message.

```
FATAL: database is not accepting commands to avoid wraparound data loss in database "database_name"
```

To manage transaction IDs and table data that is stored on disk, Greenplum Database provides the **VACUUM** command.

- A **VACUUM** operation frees up XID values so that a table can have more than 10 rows by changing the xmin values to the frozen XID.
- A **VACUUM** operation manages obsolete or deleted table rows on disk. This database’s **VACUUM** command changes the XID values **obsolescent** to indicate obsolete rows.
**VACUUM** operation, without the **FULL** option, deletes the data opportunistically to remove rows on disk with minimal impact to performance and data availability.

For the example table, a **VACUUM** operation has been performed on the table. The command updated table data on disk. This version of the **VACUUM** command performs slightly differently than the Greenplum Database command, but the concepts are the same.

- For the widget and sprocket rows on disk that are no longer current, the rows have been marked as **obsolete**.
- For the giblet and gizmo rows that are current, the xmin has been changed to the frozen XID.

The values are still current table values (the row's xmax value is **null**). However, the table row is visible to all transactions because the xmin value is frozen XID value that is older than all other XID values when modulo calculations are performed.

After the **VACUUM** operation, the XID values 0, 1, 2, and 3 available for use.

**Table 15: Example Table after VACUUM**

<table>
<thead>
<tr>
<th>item</th>
<th>amount</th>
<th>xmin</th>
<th>xmax</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget</td>
<td>100</td>
<td>obsolete</td>
<td>obsolete</td>
</tr>
<tr>
<td>giblet</td>
<td>200</td>
<td>-2</td>
<td>null</td>
</tr>
<tr>
<td>sprocket</td>
<td>300</td>
<td>obsolete</td>
<td>obsolete</td>
</tr>
<tr>
<td>gizmo</td>
<td>400</td>
<td>-2</td>
<td>null</td>
</tr>
<tr>
<td>widget</td>
<td>208</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>sproket</td>
<td>133</td>
<td>5</td>
<td>null</td>
</tr>
<tr>
<td>widget</td>
<td>16</td>
<td>6</td>
<td>null</td>
</tr>
</tbody>
</table>

When a row disk with the xmin value of -2 is updated, the xmax value is replaced with the transaction XID as usual, and the row on disk is considered obsolete after any concurrent transactions that access the row have completed.

Obsolete rows can be deleted from disk. For Greenplum Database, the **VACUUM** command, with **FULL** option, does more extensive processing to reclaim disk space.

**Example of XID Modulo Calculations**

The next table shows the table data on disk after more **UPDATE** transactions. The XID values have rolled over and start over at 0. No additional **VACUUM** operations have been performed.

**Table 16: Example Table with Wrapping XID**

<table>
<thead>
<tr>
<th>item</th>
<th>amount</th>
<th>xmin</th>
<th>xmax</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget</td>
<td>100</td>
<td>obsolete</td>
<td>obsolete</td>
</tr>
<tr>
<td>giblet</td>
<td>200</td>
<td>-2</td>
<td>1</td>
</tr>
<tr>
<td>sprocket</td>
<td>300</td>
<td>obsolete</td>
<td>obsolete</td>
</tr>
<tr>
<td>gizmo</td>
<td>400</td>
<td>-2</td>
<td>9</td>
</tr>
<tr>
<td>widget</td>
<td>208</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>sproket</td>
<td>133</td>
<td>5</td>
<td>null</td>
</tr>
<tr>
<td>widget</td>
<td>16</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
When performing the modulo calculations that compare XIDs, Greenplum Database, considers the XIDs of the rows and the current range of available XIDs to determine if XID wrapping has occurred between row XIDs.

For the example table XID wrapping has occurred. The XID 1 for giblet row is a later transaction than the XID 7 for widget row based on the modulo calculations for XID values even though the XID value 7 is larger than 1.

For the widget and sprocket rows, XID wrapping has not occurred and XID 7 is a later transaction than XID 5.

### About Parallel Data Loading

This topic provides a short introduction to Greenplum Database data loading features.

In a large scale, multi-terabyte data warehouse, large amounts of data must be loaded within a relatively small maintenance window. Greenplum supports fast, parallel data loading with its external tables feature. Administrators can also load external tables in single row error isolation mode to filter bad rows into a separate error table while continuing to load properly formatted rows. Administrators can specify an error threshold for a load operation to control how many improperly formatted rows cause Greenplum to abort the load operation.

By using external tables in conjunction with Greenplum Database's parallel file server (gpfdist), administrators can achieve maximum parallelism and load bandwidth from their Greenplum Database system.

<table>
<thead>
<tr>
<th>item</th>
<th>amount</th>
<th>xmin</th>
<th>xmax</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget</td>
<td>222</td>
<td>7</td>
<td>null</td>
</tr>
<tr>
<td>giblet</td>
<td>233</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>gizmo</td>
<td>18</td>
<td>9</td>
<td>null</td>
</tr>
<tr>
<td>giblet</td>
<td>88</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>giblet</td>
<td>44</td>
<td>1</td>
<td>null</td>
</tr>
</tbody>
</table>
Another Greenplum utility, `gpload`, runs a load task that you specify in a YAML-formatted control file. You describe the source data locations, format, transformations required, participating hosts, database destinations, and other particulars in the control file and `gpload` executes the load. This allows you to describe a complex task and execute it in a controlled, repeatable fashion.

**About Redundancy and Failover in Greenplum Database**

This topic provides a high-level overview of Greenplum Database high availability features.

You can deploy Greenplum Database without a single point of failure by mirroring components. The following sections describe the strategies for mirroring the main components of a Greenplum system. For a more detailed overview of Greenplum high availability features, see [Overview of Greenplum Database High Availability](#).

**Important:** When data loss is not acceptable for a Pivotal Greenplum Database cluster, master and segment mirroring must be enabled in order for the cluster to be supported by Pivotal. Without mirroring, system and data availability is not guaranteed; Pivotal will make best efforts to restore a cluster in this case.

**About Segment Mirroring**

When you deploy your Greenplum Database system, you can configure `mirror` segments. Mirror segments allow database queries to fail over to a backup segment if the primary segment becomes unavailable. Mirroring is strongly recommended for production systems and required for Pivotal support.

The secondary (mirror) segment must always reside on a different host than its primary segment to protect against a single host failure. In virtualized environments, the secondary (mirror) must always reside on a different storage system than the master. Mirror segments can be arranged over the remaining hosts in the cluster in configurations designed to maximize availability, or minimize the performance degradation when hosts or multiple primary segments fail.

Two standard mirroring configurations are available when you initialize or expand a Greenplum system. The default configuration, called `group mirroring`, places all the mirrors for a host's primary segments on
one other host in the cluster. The other standard configuration, spread mirroring, can be selected with a command-line option. Spread mirroring spreads each host's mirrors over the remaining hosts and requires that there are more hosts in the cluster than primary segments per host.

Figure 13: Spread Mirroring in Greenplum Database shows how table data is distributed across segments when spread mirroring is configured.

Figure 13: Spread Mirroring in Greenplum Database

**Segment Failover and Recovery**

When mirroring is enabled in a Greenplum Database system, the system will automatically fail over to the mirror segment if a primary copy becomes unavailable. A Greenplum Database system can remain operational if a segment instance or host goes down as long as all the data is available on the remaining active segments.

If the master cannot connect to a segment instance, it marks that segment instance as down in the Greenplum Database system catalog and brings up the mirror segment in its place. A failed segment instance will remain out of operation until an administrator takes steps to bring that segment back online. The gpstate utility can be used to identify failed segments. An administrator can recover a failed segment while the system is up and running. The recovery process copies over only the changes that were missed while the segment was out of operation.

If you do not have mirroring enabled, the system will automatically shut down if a segment instance becomes invalid. You must recover all failed segments before operations can continue.

**About Master Mirroring**

You can also optionally deploy a backup or mirror of the master instance on a separate host from the master node. A backup master host serves as a warm standby in the event that the primary master host becomes unoperational. The standby master is kept up to date by a transaction log replication process,
which runs on the standby master host and synchronizes the data between the primary and standby master hosts.

If the primary master fails, the log replication process stops, and the standby master can be activated in its place. The switchover does not happen automatically, but must be triggered externally. Upon activation of the standby master, the replicated logs are used to reconstruct the state of the master host at the time of the last successfully committed transaction. The activated standby master effectively becomes the Greenplum Database master, accepting client connections on the master port (which must be set to the same port number on the master host and the backup master host).

Since the master does not contain any user data, only the system catalog tables need to be synchronized between the primary and backup copies. When these tables are updated, changes are automatically copied over to the standby master to ensure synchronization with the primary master.

About Interconnect Redundancy

The interconnect refers to the inter-process communication between the segments and the network infrastructure on which this communication relies. You can achieve a highly available interconnect using by deploying dual Gigabit Ethernet switches on your network and redundant Gigabit connections to the Greenplum Database host (master and segment) servers. For performance reasons, 10-Gb Ethernet, or faster, is recommended.

About Database Statistics in Greenplum Database

An overview of statistics gathered by the `ANALYZE` command in Greenplum Database.

Statistics are metadata that describe the data stored in the database. The query optimizer needs up-to-date statistics to choose the best execution plan for a query. For example, if a query joins two tables and one of them must be broadcast to all segments, the optimizer can choose the smaller of the two tables to minimize network traffic.

The statistics used by the optimizer are calculated and saved in the system catalog by the `ANALYZE` command. There are three ways to initiate an analyze operation:

- You can run the `ANALYZE` command directly.
- You can run the `analyzedb` management utility outside of the database, at the command line.
- An automatic analyze operation can be triggered when DML operations are performed on tables that have no statistics or when a DML operation modifies a number of rows greater than a specified threshold.

These methods are described in the following sections. The `VACUUM ANALYZE` command is another way to initiate an analyze operation, but its use is discouraged because vacuum and analyze are different operations with different purposes.
Calculating statistics consumes time and resources, so Greenplum Database produces estimates by calculating statistics on samples of large tables. In most cases, the default settings provide the information needed to generate correct execution plans for queries. If the statistics produced are not producing optimal query execution plans, the administrator can tune configuration parameters to produce more accurate statistics by increasing the sample size or the granularity of statistics saved in the system catalog. Producing more accurate statistics has CPU and storage costs and may not produce better plans, so it is important to view explain plans and test query performance to ensure that the additional statistics-related costs result in better query performance.

**System Statistics**

**Table Size**

The query planner seeks to minimize the disk I/O and network traffic required to execute a query, using estimates of the number of rows that must be processed and the number of disk pages the query must access. The data from which these estimates are derived are the `pg_class` system table columns `reltuples` and `replpages`, which contain the number of rows and pages at the time a VACUUM or ANALYZE command was last run. As rows are added or deleted, the numbers become less accurate. However, an accurate count of disk pages is always available from the operating system, so as long as the ratio of `reltuples` to `replpages` does not change significantly, the optimizer can produce an estimate of the number of rows that is sufficiently accurate to choose the correct query execution plan.

When the `reltuples` column differs significantly from the row count returned by `SELECT COUNT(*)`, an analyze should be performed to update the statistics.

When a REINDEX command finishes recreating an index, the `replpages` and `reltuples` columns are set to zero. The ANALYZE command should be run on the base table to update these columns.

**The pg_statistic System Table and pg_stats View**

The `pg_statistic` system table holds the results of the last ANALYZE operation on each database table. There is a row for each column of every table. It has the following columns:

- **starelid**: The object ID of the table or index the column belongs to.
- **staatnum**: The number of the described column, beginning with 1.
- **stanullfrac**: The fraction of the column's entries that are null.
- **stawidth**: The average stored width, in bytes, of non-null entries.
- **stadistinct**: A positive number is an estimate of the number of distinct values in the column; the number is not expected to vary with the number of rows. A negative value is the number of distinct values divided by the number of rows, that is, the ratio of rows with distinct values for the column, negated. This form is used when the number of distinct values increases with the number of rows. A unique column, for example, has an `n_distinct` value of -1.0. Columns with an average width greater than 1024 are considered unique.
- **stakindN**: A code number indicating the kind of statistics stored in the Nth slot of the `pg_statistic` row.
- **staopN**: An operator used to derive the statistics stored in the Nth slot. For example, a histogram slot would show the < operator that defines the sort order of the data.
**stanumbersN**

float4 array containing numerical statistics of the appropriate kind for the Nth slot, or NULL if the slot kind does not involve numerical values.

**stavaluesN**

Column data values of the appropriate kind for the Nth slot, or NULL if the slot kind does not store any data values. Each array's element values are actually of the specific column's data type, so there is no way to define these columns' types more specifically than *anyarray*.

The statistics collected for a column vary for different data types, so the pg_statistic table stores statistics that are appropriate for the data type in four slots, consisting of four columns per slot. For example, the first slot, which normally contains the most common values for a column, consists of the columns `stakind1, staop1, stanumbers1, and stavalues1`.

The `stakindN` columns each contain a numeric code to describe the type of statistics stored in their slot. The `stakind` code numbers from 1 to 99 are reserved for core PostgreSQL data types. Greenplum Database uses code numbers 1, 2, and 3. A value of 0 means the slot is unused. The following table describes the kinds of statistics stored for the three codes.

**Table 17: Contents of pg_statistic "slots"**

<table>
<thead>
<tr>
<th>stakind Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Most CommonValues (MCV) Slot</strong></td>
</tr>
<tr>
<td></td>
<td>• staop contains the object ID of the &quot;=&quot; operator, used to decide whether values are the same or not.</td>
</tr>
<tr>
<td></td>
<td>• stavalues contains an array of the K most common non-null values appearing in the column.</td>
</tr>
<tr>
<td></td>
<td>• stanumbers contains the frequencies (fractions of total row count) of the values in the stavalues array.</td>
</tr>
<tr>
<td></td>
<td>The values are ordered in decreasing frequency. Since the arrays are variable-size, K can be chosen by the statistics collector. Values must occur more than once to be added to the stavalues array; a unique column has no MCV slot.</td>
</tr>
<tr>
<td>2</td>
<td><strong>Histogram Slot</strong> – describes the distribution of scalar data.</td>
</tr>
<tr>
<td></td>
<td>• staop is the object ID of the &quot;&lt;&quot; operator, which describes the sort ordering.</td>
</tr>
<tr>
<td></td>
<td>• stavalues contains M (where M&gt;=2) non-null values that divide the non-null column data values into M-1 bins of approximately equal population. The first stavalues item is the minimum value and the last is the maximum value.</td>
</tr>
<tr>
<td></td>
<td>• stanumbers is not used and should be null.</td>
</tr>
<tr>
<td></td>
<td>If a Most Common Values slot is also provided, then the histogram describes the data distribution after removing the values listed in the MCV array. (It is a <em>compressed histogram</em> in the technical parlance). This allows a more accurate representation of the distribution of a column with some very common values. In a column with only a few distinct values, it is possible that the MCV list describes the entire data population; in this case the histogram reduces to empty and should be omitted.</td>
</tr>
<tr>
<td>stkind Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 3           | **Correlation Slot** – describes the correlation between the physical order of table tuples and the ordering of data values of this column.  
- staop is the object ID of the "<" operator. As with the histogram, more than one entry could theoretically appear.  
- stavalues is not used and should be NULL.  
- stanumbers contains a single entry, the correlation coefficient between the sequence of data values and the sequence of their actual tuple positions. The coefficient ranges from +1 to -1. |

The `pg_stats` view presents the contents of `pg_statistic` in a friendlier format. The `pg_stats` view has the following columns:

- **schemaname**: The name of the schema containing the table.
- **tablename**: The name of the table.
- **attname**: The name of the column this row describes.
- **null_frac**: The fraction of column entries that are null.
- **avg_width**: The average storage width in bytes of the column's entries, calculated as \( \text{avg}(\text{pg\_column\_size}(\text{column\_name})) \).
- **n_distinct**: A positive number is an estimate of the number of distinct values in the column; the number is not expected to vary with the number of rows. A negative value is the number of distinct values divided by the number of rows, that is, the ratio of rows with distinct values for the column, negated. This form is used when the number of distinct values increases with the number of rows. A unique column, for example, has an `n_distinct` value of -1.0. Columns with an average width greater than 1024 are considered unique.
- **most_common_vals**: An array containing the most common values in the column, or null if no values seem to be more common. If the `n_distinct` column is -1, `most_common_vals` is null. The length of the array is the lesser of the number of actual distinct column values or the value of the `default_statistics_target` configuration parameter. The number of values can be overridden for a column using `ALTER TABLE table SET COLUMN column SET STATISTICS N`.
- **most_common_freqs**: An array containing the frequencies of the values in the `most_common_vals` array. This is the number of occurrences of the value divided by the total number of rows. The array is the same length as the `most_common_vals` array. It is null if `most_common_vals` is null.
- **histogram_bounds**: An array of values that divide the column values into groups of approximately the same size. A
Greenplum Database Administrator Guide

histogram can be defined only if there is a `max()` aggregate function for the column. The number of groups in the histogram is the same as the `most_common_vals` array size.

correlation

Greenplum Database does not calculate the correlation statistic.

Newly created tables and indexes have no statistics. You can check for tables with missing statistics using the `gp_stats_missing` view, which is in the `gp_toolkit` schema:

```
SELECT * from gp_toolkit.gp_stats_missing;
```

Sampling

When calculating statistics for large tables, Greenplum Database creates a smaller table by sampling the base table. If the table is partitioned, samples are taken from all partitions.

If the number of rows in the base table is estimated to be less than the value of the `gp_statistics_sampling_threshold` configuration parameter, the entire base table is used to calculate the statistics.

If a sample table is created, the number of rows in the sample is calculated to provide a maximum acceptable relative error. The amount of acceptable error is specified with the `gp_analyze_relative_error` system configuration parameter, which is set to 0.25 (25%) by default. This is usually sufficiently accurate to generate correct query plans. If `ANALYZE` is not producing good estimates for a table column, you can increase the sample size by setting the `gp_analyze_relative_error` configuration parameter to a lower value. Beware that setting this parameter to a low value can lead to a very large sample size and dramatically increase analyze time.

Updating Statistics

Running `ANALYZE` with no arguments updates statistics for all tables in the database. This could take a very long time, so it is better to analyze tables selectively after data has changed. You can also analyze a subset of the columns in a table, for example columns used in joins, `WHERE` clauses, `SORT` clauses, `GROUP BY` clauses, or `HAVING` clauses.

Analyzing a severely bloated table can generate poor statistics if the sample contains empty pages, so it is good practice to vacuum a bloated table before analyzing it.

See the SQL Command Reference in the Greenplum Database Reference Guide for details of running the `ANALYZE` command.

Refer to the Greenplum Database Management Utility Reference for details of running the `analyzedb` command.

Analyzing Partitioned and Append-Optimized Tables

When the `ANALYZE` command is run on a partitioned table, it analyzes each leaf-level subpartition, one at a time. You can run `ANALYZE` on just new or changed partition files to avoid analyzing partitions that have not changed. If a table is partitioned, you can analyze just new or changed partitions.

The `analyzedb` command-line utility skips unchanged partitions automatically. It also runs concurrent sessions so it can analyze several partitions concurrently. It runs five sessions by default, but the number of sessions can be set from 1 to 10 with the `-p` command-line option. Each time `analyzedb` runs, it saves state information for append-optimized tables and partitions in the `db_analyze` directory in the master data directory. The next time it runs, `analyzedb` compares the current state of each table with the saved state and skips analyzing a table or partition if it is unchanged. Heap tables are always analyzed.

If GPORCA is enabled (the default), you also need to run `ANALYZE_ROOTPARTITION` to refresh the root partition statistics. GPORCA requires statistics at the root level for partitioned tables. The
legacy optimizer does not use these statistics. Enable GPORCA by setting both the optimizer and optimizer_analyze_root_partition system configuration parameters to on. The root level statistics are then updated when you run ANALYZE or ANALYZE ROOTPARTITION. The time to run ANALYZE ROOTPARTITION is similar to the time to analyze a non-partitioned table with the same data since ANALYZE ROOTPARTITION does not collect statistics on the leaf partitions, the data is only sampled. The analyzedb utility updates root partition statistics by default but you can add the --skip_root_stats option to leave root partition statistics empty if you do not use GPORCA.

**Configuring Statistics**

There are several options for configuring Greenplum Database statistics collection.

**Statistics Target**

The statistics target is the size of the most_common_vals, most_common_freqs, and histogram_bounds arrays for an individual column. By default, the target is 25. The default target can be changed by setting a server configuration parameter and the target can be set for any column using the ALTER TABLE command. Larger values increase the time needed to do ANALYZE, but may improve the quality of the legacy query optimizer (planner) estimates.

Set the system default statistics target to a different value by setting the default_statistics_target server configuration parameter. The default value is usually sufficient, and you should only raise or lower it if your tests demonstrate that query plans improve with the new target. For example, to raise the default statistics target from 25 to 50 you can use the gpconfig utility:

```
gpconfig -c default_statistics_target -v 50
```

The statistics target for individual columns can be set with the ALTER TABLE command. For example, some queries can be improved by increasing the target for certain columns, especially columns that have irregular distributions. You can set the target to zero for columns that never contribute to query optimization. When the target is 0, ANALYZE ignores the column. For example, the following ALTER TABLE command sets the statistics target for the notes column in the emp table to zero:

```
ALTER TABLE emp ALTER COLUMN notes SET STATISTICS 0;
```

The statistics target can be set in the range 0 to 1000, or set it to -1 to revert to using the system default statistics target.

Setting the statistics target on a parent partition table affects the child partitions. If you set statistics to 0 on some columns on the parent table, the statistics for the same columns are set to 0 for all children partitions. However, if you later add or exchange another child partition, the new child partition will use either the default statistics target or, in the case of an exchange, the previous statistics target. Therefore, if you add or exchange child partitions, you should set the statistics targets on the new child table.

**Automatic Statistics Collection**

Greenplum Database can be set to automatically run ANALYZE on a table that either has no statistics or has changed significantly when certain operations are performed on the table. For partitioned tables, automatic statistics collection is only triggered when the operation is run directly on a leaf table, and then only the leaf table is analyzed.

Automatic statistics collection has three modes:

- **none** disables automatic statistics collection.
- **on_no_stats** triggers an analyze operation for a table with no existing statistics when any of the commands CREATE TABLE AS SELECT, INSERT, or COPY are executed on the table.
- **on_change** triggers an analyze operation when any of the commands CREATE TABLE AS SELECT, UPDATE, DELETE, INSERT, or COPY are executed on the table and the number of rows affected
exceeds the threshold defined by the `gp_autostats_on_change_threshold` configuration parameter.

The automatic statistics collection mode is set separately for commands that occur within a procedural language function and commands that execute outside of a function:

- The `gp_autostats_mode` configuration parameter controls automatic statistics collection behavior outside of functions and is set to `on_no_stats` by default.
- The `gp_autostats_mode_in_functions` parameter controls the behavior when table operations are performed within a procedural language function and is set to `none` by default.

With the `on_change` mode, `ANALYZE` is triggered only if the number of rows affected exceeds the threshold defined by the `gp_autostats_on_change_threshold` configuration parameter. The default value for this parameter is a very high value, 2147483647, which effectively disables automatic statistics collection; you must set the threshold to a lower number to enable it. The `on_change` mode could trigger large, unexpected analyze operations that could disrupt the system, so it is not recommended to set it globally. It could be useful in a session, for example to automatically analyze a table following a load.

To disable automatic statistics collection outside of functions, set the `gp_autostats_mode` parameter to `none`:

```
gpconfigure -c gp_autostats_mode -v none
```

To enable automatic statistics collection in functions for tables that have no statistics, change `gp_autostats_mode_in_functions` to `on_no_stats`:

```
gpconfigure -c gp_autostats_mode_in_functions -v on_no_stats
```

Set the `log_autostats` system configuration parameter to `on` if you want to log automatic statistics collection operations.
Managing a Greenplum System

This section describes basic system administration tasks performed by a Greenplum Database system administrator.

This section contains the following topics:

- Starting and Stopping Greenplum Database
- Accessing the Database
- Configuring the Greenplum Database System
- Enabling High Availability and Data Consistency Features
- Backing Up and Restoring Databases
- Expanding a Greenplum System
- Migrating Data with gptransfer
- Defining Database Objects
- Routine System Maintenance Tasks

Starting and Stopping Greenplum Database

In a Greenplum Database DBMS, the database server instances (the master and all segments) are started or stopped across all of the hosts in the system in such a way that they can work together as a unified DBMS.

Because a Greenplum Database system is distributed across many machines, the process for starting and stopping a Greenplum Database system is different than the process for starting and stopping a regular PostgreSQL DBMS.

Use the `gpstart` and `gpstop` utilities to start and stop Greenplum Database, respectively. These utilities are located in the `$GPHOME/bin` directory on your Greenplum Database master host.

**Important:** Do not issue a `kill` command to end any Postgres process. Instead, use the `database` command `pg_cancel_backend()`.

Issuing a `kill -9` or `kill -11` can introduce database corruption and prevent root cause analysis from being performed.

For information about `gpstart` and `gpstop`, see the *Greenplum Database Utility Guide*.

Starting Greenplum Database

Start an initialized Greenplum Database system by running the `gpstart` utility on the master instance.

Use the `gpstart` utility to start a Greenplum Database system that has already been initialized by the `gpinitsystem` utility, but has been stopped by the `gpstop` utility. The `gpstart` utility starts Greenplum Database by starting all the Postgres database instances on the Greenplum Database cluster. `gpstart` orchestrates this process and performs the process in parallel.

- Run `gpstart` on the master host to start Greenplum Database:

  ```
  $ gpstart
  ```

Restarting Greenplum Database

Stop the Greenplum Database system and then restart it.

The `gpstop` utility with the `-r` option can stop and then restart Greenplum Database after the shutdown completes.
• To restart Greenplum Database, enter the following command on the master host:

```
$ gpstop -r
```

### Reloading Configuration File Changes Only

Reload changes to Greenplum Database configuration files without interrupting the system.

The `gpstop` utility can reload changes to the `pg_hba.conf` configuration file and to `runtime` parameters in the master `postgresql.conf` file and `pg_hba.conf` file without service interruption. Active sessions pick up changes when they reconnect to the database. Many server configuration parameters require a full system restart (`gpstop -r`) to activate. For information about server configuration parameters, see the *Greenplum Database Reference Guide*.

• Reload configuration file changes without shutting down the system using the `gpstop` utility:

```
$ gpstop -u
```

### Starting the Master in Maintenance Mode

Start only the master to perform maintenance or administrative tasks without affecting data on the segments.

Maintenance mode should only be used with direction from Pivotal Technical Support. For example, you could connect to a database only on the master instance in maintenance mode and edit system catalog settings. For more information about system catalog tables, see the *Greenplum Database Reference Guide*.

1. Run `gpstart` using the `-m` option:

```
$ gpstart -m
```

2. Connect to the master in maintenance mode to do catalog maintenance. For example:

```
$ PGOPTIONS='-c gp_session_role=utility' psql postgres
```

3. After completing your administrative tasks, stop the master in utility mode. Then, restart it in production mode.

```
$ gpstop -mr
```

**Warning:**

Incorrect use of maintenance mode connections can result in an inconsistent system state. Only Technical Support should perform this operation.

### Stopping Greenplum Database

The `gpstop` utility stops or restarts your Greenplum Database system and always runs on the master host. When activated, `gpstop` stops all `postgres` processes in the system, including the master and all segment instances. The `gpstop` utility uses a default of up to 64 parallel worker threads to bring down the Postgres instances that make up the Greenplum Database cluster. The system waits for any active transactions to finish before shutting down. To stop Greenplum Database immediately, use fast mode.

• To stop Greenplum Database:

```
$ gpstop
```
To stop Greenplum Database in fast mode:

```bash
$ gpstop -M fast
```

By default, you are not allowed to shut down Greenplum Database if there are any client connections to the database. Use the `-M fast` option to roll back all in progress transactions and terminate any connections before shutting down.

### Accessing the Database

This topic describes the various client tools you can use to connect to Greenplum Database, and how to establish a database session.

### Establishing a Database Session

Users can connect to Greenplum Database using a PostgreSQL-compatible client program, such as `psql`. Users and administrators *always* connect to Greenplum Database through the `master`; the segments cannot accept client connections.

In order to establish a connection to the Greenplum Database master, you will need to know the following connection information and configure your client program accordingly.

#### Table 18: Connection Parameters

<table>
<thead>
<tr>
<th>Connection Parameter</th>
<th>Description</th>
<th>Environment Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application name</td>
<td>The application name that is connecting to the database. The default value, held in the <code>application_name</code> connection parameter is <code>psql</code>.</td>
<td><code>$PGAPPNAME</code></td>
</tr>
<tr>
<td>Database name</td>
<td>The name of the database to which you want to connect. For a newly initialized system, use the <code>postgres</code> database to connect for the first time.</td>
<td><code>$PGDATABASE</code></td>
</tr>
<tr>
<td>Host name</td>
<td>The host name of the Greenplum Database master. The default host is the local host.</td>
<td><code>$PGHOST</code></td>
</tr>
<tr>
<td>Port</td>
<td>The port number that the Greenplum Database master instance is running on. The default is 5432.</td>
<td><code>$PGPORT</code></td>
</tr>
<tr>
<td>User name</td>
<td>The database user (role) name to connect as. This is not necessarily the same as your OS user name. Check with your Greenplum administrator if you are not sure what you database user name is. Note that every Greenplum Database system has one superuser account that is created automatically at initialization time. This account has the same name as the OS name of the user who initialized the Greenplum system (typically <code>gpadmin</code>).</td>
<td><code>$PGUSER</code></td>
</tr>
</tbody>
</table>

*Connecting with `psql` provides example commands for connecting to Greenplum Database.*

### Supported Client Applications

Users can connect to Greenplum Database using various client applications:

- A number of Greenplum Database Client Applications are provided with your Greenplum installation. The `psql` client application provides an interactive command-line interface to Greenplum Database.
• Using standard *Database Application Interfaces*, such as ODBC and JDBC, users can create their own client applications that interface to Greenplum Database.

• Most client tools that use standard database interfaces, such as ODBC and JDBC, can be configured to connect to Greenplum Database.

**Greenplum Database Client Applications**

Greenplum Database comes installed with a number of client utility applications located in the $GPHOME/bin directory of your Greenplum Database master host installation. The following are the most commonly used client utility applications:

**Table 19: Commonly used client applications**

<table>
<thead>
<tr>
<th>Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>createdb</td>
<td>create a new database</td>
</tr>
<tr>
<td>createlang</td>
<td>define a new procedural language</td>
</tr>
<tr>
<td>createuser</td>
<td>define a new database role</td>
</tr>
<tr>
<td>dropdb</td>
<td>remove a database</td>
</tr>
<tr>
<td>droplang</td>
<td>remove a procedural language</td>
</tr>
<tr>
<td>dropuser</td>
<td>remove a role</td>
</tr>
<tr>
<td>psql</td>
<td>PostgreSQL interactive terminal</td>
</tr>
<tr>
<td>reindexdb</td>
<td>reindex a database</td>
</tr>
<tr>
<td>vacuumdb</td>
<td>garbage-collect and analyze a database</td>
</tr>
</tbody>
</table>

When using these client applications, you must connect to a database through the Greenplum master instance. You will need to know the name of your target database, the host name and port number of the master, and what database user name to connect as. This information can be provided on the command-line using the options `-d`, `-h`, `-p`, and `-U` respectively. If an argument is found that does not belong to any option, it will be interpreted as the database name first.

All of these options have default values which will be used if the option is not specified. The default host is the local host. The default port number is 5432. The default user name is your OS system user name, as is the default database name. Note that OS user names and Greenplum Database user names are not necessarily the same.

If the default values are not correct, you can set the environment variables PGDATABASE, PGHOST, PGPORT, and PGUSER to the appropriate values, or use a psql ~/.pgpass file to contain frequently-used passwords.

For information about Greenplum Database environment variables, see the *Greenplum Database Reference Guide*. For information about psql, see the *Greenplum Database Utility Guide*.

**Connecting with psql**

Depending on the default values used or the environment variables you have set, the following examples show how to access a database via *psql*:

```bash
$ psql -d gpdatabase -h master_host -p 5432 -U gpadmin
```
If a user-defined database has not yet been created, you can access the system by connecting to the `postgres` database. For example:

```plaintext
$ psql postgres
```

After connecting to a database, `psql` provides a prompt with the name of the database to which `psql` is currently connected, followed by the string `=>` (or `=#` if you are the database superuser). For example:

```plaintext
gpdatabase=>
```

At the prompt, you may type in SQL commands. A SQL command must end with a `;` (semicolon) in order to be sent to the server and executed. For example:

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

See the *Greenplum Reference Guide* for information about using the `psql` client application and SQL commands and syntax.

**pgAdmin 4 with Greenplum Database**

If you prefer a graphic interface, use pgAdmin 4. This GUI client supports PostgreSQL databases with all standard pgAdmin 4 features, while adding support for some Greenplum-specific features. pgAdmin 4 is available for download at [https://www.pgadmin.org/docs/pgadmin4/1.x/](https://www.pgadmin.org/docs/pgadmin4/1.x/).

**Note:** For limitations on using pgAdmin4 with Greenplum Database, refer to the *Greenplum Database Release Notes*.

pgAdmin 4 supports the following Greenplum-specific features:

- Viewing Greenplum table DDL.
- Append-optimized tables, including compressed append-optimized tables.

  **Note:** Only limited functionality for Greenplum partitioned tables is available with pgAdmin 4.

  **Note:** Graphical `EXPLAIN` features are currently not available with Greenplum Database.
Installing pgAdmin 4

The installation package for pgAdmin 4 is available for download from the official pgAdmin 4 download site (https://www.pgadmin.org). Installation instructions are included in the installation package.

When configuring the server connection for Greenplum Database, retain the default Maintenance database of postgres, but specify gpadmin for the Username. When connected, Greenplum Database connections will be indicated by the Greenplum logo.

Viewing DDL Data with pgAdmin 4

You can use pgAdmin 4 to view Greenplum Database table DDL.

To view the table DDL, simply drill down through the database information in the left navigation and select the desired table. Click on the SQL tab. The DDL will be displayed. You can also display DDL information for individual columns.
Using the PgBouncer Connection Pooler

The PgBouncer utility manages connection pools for PostgreSQL and Greenplum Database connections.

The Greenplum Database installation includes the PgBouncer connection pooling software. The following topics describe how to set up and use PgBouncer with Greenplum Database. See the PgBouncer Web site for information about using PgBouncer with PostgreSQL.

- Overview
- Configuring and Starting PgBouncer
- Managing PgBouncer
- Setting up LDAP Authentication with PgBouncer
- Securing PgBouncer Connections with stunnel

Also, see reference information for PgBouncer in the Greenplum Database Utility Guide.
Overview

A database connection pool is a cache of database connections. Once a pool of connections is established, connection pooling eliminates the overhead of creating database connections, so clients connect much faster and the server load is reduced.

The PgBouncer connection pooler, from the PostgreSQL community, is included with Greenplum Database. PgBouncer can manage connection pools for multiple databases, and databases may be on different Greenplum Database clusters or PostgreSQL backends. PgBouncer creates a pool for each database user and database combination. A pooled connection can only be reused for another connection request for the same user and database.

The client application requires no software changes, but connects to the connection pool's host and port instead of the Greenplum Database master host and port. PgBouncer either creates a new database connection or reuses an existing connection. When the client disconnects, the connection is returned to the pool for re-use.

PgBouncer supports the standard connection interface that PostgreSQL and Greenplum Database share. A client requesting a database connection provides the host name and port where PgBouncer is running, as well as the database name, username, and password. PgBouncer looks up the requested database (which may be an alias for the actual database) in its configuration file to find the host name, port, and database name for the database connection. The configuration file entry also determines how to authenticate the user and what database role will be used for the connection—a “forced user” can override the username provided with the client's connection request.

PgBouncer requires an authentication file, a text file that contains a list of users and passwords. Passwords may be either clear text, MD5-encoded, or an LDAP/AD lookup string. You can also set up PgBouncer to query the destination database for users that are not in the authentication file.

PgBouncer shares connections in one of three pool modes:

- **Session pooling** – When a client connects, a connection is assigned to it as long as it remains connected. When the client disconnects, the connection is placed back into the pool.
- **Transaction pooling** – A connection is assigned to a client for the duration of a transaction. When PgBouncer notices the transaction is done, the connection is placed back into the pool. This mode can be used only with applications that do not use features that depend upon a session.
- **Statement pooling** – Statement pooling is like transaction pooling, but multi-statement transactions are not allowed. This mode is intended to enforce autocommit mode on the client and is targeted for PL/Proxy on PostgreSQL.

A default pool mode can be set for the PgBouncer instance and the mode can be overridden for individual databases and users.

By connecting to a virtual pgbouncer database, you can monitor and manage PgBouncer using SQL-like commands. Configuration parameters can be changed without having to restart PgBouncer, and the configuration file can be reloaded to pick up changes.

PgBouncer does not yet support SSL connections. If you want to encrypt traffic between clients and PgBouncer, you can use stunnel, a free software utility that creates TLS-encrypted tunnels using the OpenSSL cryptography library. See **Securing PgBouncer Connections with stunnel** for directions.

Configuring and Starting PgBouncer

PgBouncer can be run on the Greenplum Database master or on another server. If you install PgBouncer on a separate server, you can easily switch clients to the standby master by updating the PgBouncer configuration file and reloading the configuration using the PgBouncer Administration Console.

Follow these steps to set up PgBouncer.
1. Create a PgBouncer configuration file, for example `pgbouncer.ini`. Here is a simple configuration file:

```
[databases]
postgres = host=127.0.0.1 port=5432 dbname=postgres
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
```

The file is in the `.ini` file format and has three sections: `databases`, `pgbouncer`, and `users`. The `databases` section lists databases with their connection details. The `pgbouncer` section configures the PgBouncer instance.

2. Create an authentication file. The name of the file must match the `auth_file` parameter in the `pgbouncer.ini` file, `users.txt` in this example. Each line contains a user name and password. The format of the password string matches the `auth_type` parameter in the PgBouncer configuration file. If the `auth_type` parameter is `plain`, the password string is a clear text password, for example:

```
"gpadmin" "gpadmin1234"
```

The `auth_type` in the following example is `md5`, so the authentication field must be MD5-encoded. The format for an MD5-encoded password is:

```
"md5" + MD5(<password><username>)
```

You can use the Linux `md5sum` command to calculate the MD5 string. For example, if the `gpadmin` password is `admin1234` the following command prints the string for the password field:

```
$ user=gpadmin; passwd=admin1234; echo -n md5; echo $passwd$user | md5sum
md53ce96652dedd8226c498e09ae2d26220
```

And here is the MD5-encoded entry for the `gpadmin` user in the PgBouncer authentication file:

```
"gpadmin" "md53ce96652dedd8226c498e09ae2d26220"
```

To authenticate users against an LDAP or Active Directory server, the password field contains an LDAP lookup string. For example:

```
"gpdbuser1" "ldap://10.0.0.11:10389/uid=gpdbuser1,ou=users,ou=system"
```

For Active Directory, a user entry looks like this example:

```
"gpdbuser2" "ldap://10.0.0.12:389/gpdbuser2"
```

See Setting up LDAP Authentication with PgBouncer for the steps to configure PgBouncer to authenticate users with an LDAP server.

For details about the authentication file, see the "Authentication File Format" section of the PgBouncer reference in the Greenplum Database Utility Guide.
3. **Launch pgbouncer:**
   
   ```
   $ $GPHOME/bin/pgbouncer -d pgbouncer.ini
   ```
   
   The `-d` or `--daemon` option runs PgBouncer as a background process. See the PgBouncer reference in the *Greenplum Database Utility Guide* for the `pgbouncer` command syntax and its options.

4. **Update client applications to connect to pgbouncer instead of directly to Greenplum Database server.**
   
   To start `psql`, for example:
   
   ```
   $ psql -p 6543 -U someuser postgres
   ```

### Server Reset Query

When a connection is returned to the pool, it must be reset to the state of a newly created connection. PgBouncer accomplishes this by issuing a query before returning a connection to the pool. PostgreSQL 8.3 and later have the `DISCARD ALL` command for this purpose and it is the default reset query for the standard PgBouncer distribution. Greenplum Database does not support `DISCARD ALL` and if it is used an error is logged.

A reset query can be specified by setting the `server_reset_query` parameter in the PgBouncer configuration file. For Greenplum Database with PgBouncer in session pooling mode, the `server_reset_query` parameter can be set to this query:

```
RESET ALL; SET SESSION AUTHORIZATION DEFAULT
```

This is the default server reset query in the modified version of PgBouncer included with Greenplum Database 4.7.0 and later. Although the default differs from PgBouncer, it helps to ensure that the connection pool is transparent to Greenplum Database users and client applications do not have to be modified to use a connection pool.

For more information about the `server_reset_query` parameter and the PgBouncer configuration file, see the PgBouncer reference in the *Greenplum Database Utility Guide*.

### Managing PgBouncer

PgBouncer has an administrative console, which is accessed by logging into the `pgbouncer` virtual database. The console accepts SQL-like commands that allow you to monitor, reconfigure, and manage PgBouncer.

Follow these steps to get started with the PgBouncer administrative console.

1. **Log in to the pgbouncer virtual database with psql:**
   
   ```
   $ psql -p 6543 -U username pgbouncer
   ```
   
   The username must be set in the `admin_users` parameter in the `pgbouncer.ini` configuration file. You can also log in with the current Unix username if the `pgbouncer` process is running with that user's UID.

2. **To see the available commands, run the `show help` command:**
   
   ```
   pgbouncer=# show help;
   NOTICE:  Console usage
   DETAIL:   SHOW HELP|CONFIG|DATABASES|POOLS|CLIENTS|SERVERS|VERSION
   SHOW STATS|FDS|SOCKETS|ACTIVE SOCKETS|LISTS|MEM
   SHOW DNS_HOSTS|DNS_ZONES
   SET key = arg
   RELOAD
   PAUSE [<db>]
   RESUME [</db>]
   ```
3. If you make changes to the `pgbouncer.ini` file, you can reload it with the `RELOAD` command:

   ```
   pgbouncer=# RELOAD;
   ```

To map clients to server connections, use the `SHOW CLIENTS` and `SHOW SERVERS` views:

1. Use `ptr` and `link` to map the local client connection to the server connection.
2. Use `addr` and `port` of the client connection to identify the TCP connection from the client.
3. Use `local_addr` and `local_port` to identify the TCP connection to the server.

For complete documentation for all of the administration console commands, see the “PgBouncer Administration Console Commands” section of the PgBouncer reference in the *Greenplum Database Utility Guide*.

**Upgrading PgBouncer**

You can upgrade PgBouncer without dropping connections. Just launch the new PgBouncer process with the `-R` option and the same configuration file:

```
$ pgbouncer -R -d config.ini
```

The `-R` (reboot) option causes the new process to connect to the console of the old process through a Unix socket and issue the following commands:

```
SUSPEND;
SHOW FDS;
SHUTDOWN;
```

When the new process sees that the old process is gone, it resumes the work with the old connections. This is possible because the `SHOW FDS` command sends actual file descriptors to the new process. If the transition fails for any reason, kill the new process and the old process will resume.

**Setting up LDAP Authentication with PgBouncer**

You can authenticate Greenplum Database users against your LDAP or Active Directory service when using the PgBouncer connection pooler. When you have LDAP authentication working, you can secure client connections to PgBouncer by setting up stunnel, as described in [Securing PgBouncer Connections with stunnel](#).

Before you begin, you must have an LDAP or Active Directory server and a Greenplum Database cluster.

1. Create Greenplum Database users in the LDAP/AD server, for example `gpdbuser1` and `gpdbuser2`.
2. Create the corresponding Greenplum Database users in the database:

   ```
   createuser gpdbuser1
   createuser gpdbuser2
   ```

3. Add the users to the Greenplum Database `pg_hba.conf` file:

   ```
   host     all     gpdbuser1     0.0.0.0/0     trust
   host     all     gpdbuser2     0.0.0.0/0     trust
   ```

4. Create a PgBouncer `config.ini` file with the following contents:
* = host=GPDB_host_addr port=GPDB_port

[pgbouncer]
listen_port = 6432
listen_addr = 0.0.0.0
auth_type = plain
auth_file = users.txt
logfile = pgbouncer.log
pidfile = pgbouncer.pid
ignore_startup_parameters=options

5. Create the users.txt PgBouncer authentication file. The first field is the user name and the second is the LDAP or Active Directory lookup string for the user. For OpenLDAP or ApacheDS, for example, a user entry in the users.txt looks like this:

"gpdbuser1" "ldap://10.0.0.11:10389/uid=gpdbuser1,ou=users,ou=system"

For Active Directory, a user entry looks like this example:

"gpdbuser1" "ldap://10.0.0.12:389/gpdbuser1"

6. Start PgBouncer with the config.ini file you created:

$ pgbouncer -d config.ini

Securing PgBouncer Connections with stunnel

PgBouncer does not have SSL support, so connections between database clients and PgBouncer are not encrypted. To encrypt these connections, you can use stunnel, a free software utility. stunnel is a proxy that uses the OpenSSL cryptographic library to add TLS encryption to network services. Newer versions of stunnel support the PostgreSQL libpq protocol, so Greenplum and PostgreSQL clients can use libpq SSL support to connect securely over the network to a stunnel instance set up as a proxy for PgBouncer. If you use a version of stunnel without libpq support, you need to set up a stunnel instance on both the client and PgBouncer host to create a secure tunnel over the network.

On most platforms, you can install stunnel using the system package manager. For Windows, you can download an installer from https://www.stunnel.org/index.html. You can also download and install stunnel from source if a packaged version is unavailable, if you want to upgrade to a newer version, or if you want to upgrade to a newer OpenSSL version. The next section provides steps to download, compile, and install OpenSSL and stunnel.

For complete stunnel documentation, visit the stunnel documentation web site.

Installing stunnel From Source

stunnel requires a version of the OpenSSL development package greater than 1.0. The following instructions download and build OpenSSL and stunnel for Red Hat or CentOS. Be sure to check https://www.stunnel.org/index.html and http://openssl.org for the latest versions.

1. Set the PREFIX environment variable to the directory where you want to install stunnel, for example /usr/local/stunnel or /home/gpadmin/stunnel. If you choose to install stunnel in a system directory, you may need to run the make install commands as root.

   export PREFIX=/path/to/install_stunnel

2. The following commands download, build, and install OpenSSL version 1.0.2c:

   wget ftp://ftp.openssl.org/source/openssl-1.0.2c.tar.gz
tar xzf openssl-1.0.2c.tar.gz
cd openssl-1.0.2c
./Configure linux-x86_64 --prefix=$PREFIX shared no-asm
3. Download, build, and install stunnel with the following commands:

```
make
make install_sw
cd ..
```

### Setting up Stunnel for PgBouncer

Set up a stunnel instance as a proxy for PgBouncer. The stunnel instance accepts secure connections from database clients on its own port, decrypts the packets and forwards them to the PgBouncer instance on the same host (or on another host in a protected local network). PgBouncer results, including database results from connections managed by PgBouncer, are encrypted and returned over the network to the client.

1. On the server running PgBouncer, create a directory for stunnel configuration files, for example `/home/gpadmin/stunnel`.

2. Create a certificate and key file to use for authenticating. You can also use an existing certificate and key, or create a new pair with the following `openssl` command:

   ```
   $ openssl req -new -x509 -days 3650 -nodes -out stunnel.pem -keyout stunnel.key
   ```

   Move the `stunnel.key` and `stunnel.pem` files into your stunnel configuration directory.

3. Create a stunnel configuration file, for example, `stunnel.conf`, with the following contents:

   ```
   debug = info
   socket = l:TCP_NODELAY=1
   socket = r:TCP_NODELAY=1
   debug = 7
   cert = /PATH/TO/stunnel.pem
   key = /PATH/TO/stunnel.key
   [pg-server]
   client=no
   accept = 0.0.0.0:5433   # This is the SSL listening port
   connect = PGHOST:PGPORT   # This is the PgBouncer listen port
   protocol = pgsql
   ```

   See the [stunnel configuration file reference](https://www.stunnel.org/docs/stunnel.conf) for additional configuration options.

4. Run `stunnel` on the server:

   ```
   stunnel /path/to/stunnel-srv.conf
   ```

5. On the client, connect to stunnel on its host and accept port. For example:

   ```
   psql -h pgbouncer-host -p stunnel-accept-port database-name
   ```
Database Application Interfaces

You may want to develop your own client applications that interface to Greenplum Database. PostgreSQL provides a number of database drivers for the most commonly used database application programming interfaces (APIs), which can also be used with Greenplum Database. These drivers are available as a separate download. Each driver (except libpq, which comes with PostgreSQL) is an independent PostgreSQL development project and must be downloaded, installed and configured to connect to Greenplum Database. The following drivers are available:

Table 20: Greenplum Database Interfaces

<table>
<thead>
<tr>
<th>API</th>
<th>PostgreSQL Driver</th>
<th>Download Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDBC</td>
<td>Greenplum DataDirect JDBC Driver</td>
<td><a href="https://network.pivotal.io/products/pivotal-gpdb">https://network.pivotal.io/products/pivotal-gpdb</a></td>
</tr>
<tr>
<td>Perl DBI</td>
<td>pgperl</td>
<td><a href="http://search.cpan.org/dist/DBD-Pg/">http://search.cpan.org/dist/DBD-Pg/</a></td>
</tr>
<tr>
<td>Python DBI</td>
<td>pygresql</td>
<td><a href="http://www.pygresql.org/">http://www.pygresql.org/</a></td>
</tr>
<tr>
<td>libpq C Library</td>
<td>libpq</td>
<td><a href="https://www.postgresql.org/docs/8.3/static/libpq.html">https://www.postgresql.org/docs/8.3/static/libpq.html</a></td>
</tr>
</tbody>
</table>

General instructions for accessing a Greenplum Database with an API are:

1. Download your programming language platform and respective API from the appropriate source. For example, you can get the Java Development Kit (JDK) and JDBC API from Oracle.
2. Write your client application according to the API specifications. When programming your application, be aware of the SQL support in Greenplum Database so you do not include any unsupported SQL syntax.

   See the Greenplum Database Reference Guide for more information.

Download the appropriate driver and configure connectivity to your Greenplum Database master instance.

Troubleshooting Connection Problems

A number of things can prevent a client application from successfully connecting to Greenplum Database. This topic explains some of the common causes of connection problems and how to correct them.

Table 21: Common connection problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>No pg_hba.conf entry for host or user</td>
<td>To enable Greenplum Database to accept remote client connections, you must configure your Greenplum Database master instance so that connections are allowed from the client hosts and database users that will be connecting to Greenplum Database. This is done by adding the appropriate entries to the pg_hba.conf configuration file (located in the master instance's data directory). For more detailed information, see Allowing Connections to Greenplum Database.</td>
</tr>
<tr>
<td>Greenplum Database is not running</td>
<td>If the Greenplum Database master instance is down, users will not be able to connect. You can verify that the Greenplum Database system is up by running the gpstate utility on the Greenplum master host.</td>
</tr>
<tr>
<td>Problem</td>
<td>Solution</td>
</tr>
<tr>
<td>------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Network problems             | If users connect to the Greenplum master host from a remote client, network problems can prevent a connection (for example, DNS host name resolution problems, the host system is down, and so on.). To ensure that network problems are not the cause, connect to the Greenplum master host from the remote client host. For example: `ping hostname`.
| Interconnect timeouts        | If the system cannot resolve the host names and IP addresses of the hosts involved in Greenplum Database, queries and connections will fail. For some operations, connections to the Greenplum Database master use `localhost` and others use the actual host name, so you must be able to resolve both. If you encounter this error, first make sure you can connect to each host in your Greenplum Database array from the master host over the network. In the `/etc/hosts` file of the master and all segments, make sure you have the correct host names and IP addresses for all hosts involved in the Greenplum Database array. The `127.0.0.1` IP must resolve to `localhost`. |
| Too many clients already     | By default, Greenplum Database is configured to allow a maximum of 250 concurrent user connections on the master and 750 on a segment. A connection attempt that causes that limit to be exceeded will be refused. This limit is controlled by the `max_connections` parameter in the `postgresql.conf` configuration file of the Greenplum Database master. If you change this setting for the master, you must also make appropriate changes at the segments. |

**Configuring the Greenplum Database System**

Server configuration parameters affect the behavior of Greenplum Database. They are part of the PostgreSQL "Grand Unified Configuration" system, so they are sometimes called "GUCs." Most of the Greenplum Database server configuration parameters are the same as the PostgreSQL configuration parameters, but some are Greenplum-specific.

**About Greenplum Database Master and Local Parameters**

Server configuration files contain parameters that configure server behavior. The Greenplum Database configuration file, `postgresql.conf`, resides in the data directory of the database instance.

The master and each segment instance have their own `postgresql.conf` file. Some parameters are `local`: each segment instance examines its `postgresql.conf` file to get the value of that parameter. Set local parameters on the master and on each segment instance.

Other parameters are `master` parameters that you set on the master instance. The value is passed down to (or in some cases ignored by) the segment instances at query run time.

See the Greenplum Database Reference Guide for information about `local` and `master` server configuration parameters.

**Setting Configuration Parameters**

Many configuration parameters limit who can change them and where or when they can be set. For example, to change certain parameters, you must be a Greenplum Database superuser. Other parameters can be set only at the system level in the `postgresql.conf` file or require a system restart to take effect.

Many configuration parameters are `session` parameters. You can set session parameters at the system level, the database level, the role level or the session level. Database users can change most session parameters within their session, but some require superuser permissions.

See the Greenplum Database Reference Guide for information about setting server configuration parameters.
**Setting a Local Configuration Parameter**

To change a local configuration parameter across multiple segments, update the parameter in the `postgresql.conf` file of each targeted segment, both primary and mirror. Use the `gpconfig` utility to set a parameter in all Greenplum `postgresql.conf` files. For example:

```
$ gpconfig -c gp_vmem_protect_limit -v 4096
```

Restart Greenplum Database to make the configuration changes effective:

```
$ gpstop -r
```

**Setting a Master Configuration Parameter**

To set a master configuration parameter, set it at the Greenplum Database master instance. If it is also a session parameter, you can set the parameter for a particular database, role or session. If a parameter is set at multiple levels, the most granular level takes precedence. For example, session overrides role, role overrides database, and database overrides system.

**Setting Parameters at the System Level**

Master parameter settings in the master `postgresql.conf` file are the system-wide default. To set a master parameter:

1. Edit the `$MASTER_DATA_DIRECTORY/postgresql.conf` file.
2. Find the parameter to set, uncomment it (remove the preceding `#` character), and type the desired value.
3. Save and close the file.
4. For session parameters that do not require a server restart, upload the `postgresql.conf` changes as follows:
   
   $ gpstop -u

5. For parameter changes that require a server restart, restart Greenplum Database as follows:

   $ gpstop -r

For details about the server configuration parameters, see the *Greenplum Database Reference Guide*.

**Setting Parameters at the Database Level**

Use `ALTER DATABASE` to set parameters at the database level. For example:

```
=# ALTER DATABASE mydatabase SET search_path TO myschema;
```

When you set a session parameter at the database level, every session that connects to that database uses that parameter setting. Settings at the database level override settings at the system level.

**Setting Parameters at the Role Level**

Use `ALTER ROLE` to set a parameter at the role level. For example:

```
=# ALTER ROLE bob SET search_path TO bobschema;
```

When you set a session parameter at the role level, every session initiated by that role uses that parameter setting. Settings at the role level override settings at the database level.
**Setting Parameters in a Session**

Any session parameter can be set in an active database session using the `SET` command. For example:

```
=# SET statement_mem TO '200MB';
```

The parameter setting is valid for the rest of that session or until you issue a `RESET` command. For example:

```
=# RESET statement_mem;
```

Settings at the session level override those at the role level.

**Viewing Server Configuration Parameter Settings**

The SQL command `SHOW` allows you to see the current server configuration parameter settings. For example, to see the settings for all parameters:

```
$ psql -c 'SHOW ALL;'
```

`SHOW` lists the settings for the master instance only. To see the value of a particular parameter across the entire system (master and all segments), use the `gpconfig` utility. For example:

```
$ gpconfig --show max_connections
```

**Configuration Parameter Categories**

Configuration parameters affect categories of server behaviors, such as resource consumption, query tuning, and authentication. The following topics describe Greenplum Database configuration parameter categories.

For details about configuration parameter categories, see the *Greenplum Database Reference Guide*.

**Connection and Authentication Parameters**

These parameters control how clients connect and authenticate to Greenplum Database.

See *Managing Greenplum Database Access* for information about configuring client authentication.

**Connection Parameters**

<table>
<thead>
<tr>
<th>gp_connection_send_timeout</th>
<th>tcp_keepalives_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp_vmem_idle_resource_timeout</td>
<td>tcp_keepalives_idle</td>
</tr>
<tr>
<td>listen_addresses</td>
<td>tcp_keepalives_interval</td>
</tr>
<tr>
<td>max_connections</td>
<td>unix_socket_directory</td>
</tr>
<tr>
<td>max_prepared_transactions</td>
<td>unix_socket_group</td>
</tr>
<tr>
<td>superuser_reserved_connections</td>
<td>unix_socket_permissions</td>
</tr>
</tbody>
</table>

**Security and Authentication Parameters**

<table>
<thead>
<tr>
<th>authentication_timeout</th>
<th>krb_srvname</th>
</tr>
</thead>
<tbody>
<tr>
<td>db_user_namespace</td>
<td>password_encryption</td>
</tr>
</tbody>
</table>
System Resource Consumption Parameters

Memory Consumption Parameters

These parameters control system memory usage. You can adjust `gp_vmem_protect_limit` to avoid running out of memory at the segment hosts during query processing.

- `gp_vmem_idle_resource_timeout`
- `gp_vmem_protect_limit`
- `gp_vmem_protect_segworker_cache_limit`
- `gp_workfile_limit_files_per_query`
- `gp_workfile_limit_per_query`
- `gp_workfile_limit_per_segment`
- `max_appendonly_tables`
- `max_prepared_transactions`
- `max_stack_depth`
- `shared_buffers`
- `temp_buffers`

Free Space Map Parameters

These parameters control the sizing of the free space map, which contains expired rows. Use VACUUM to reclaim the free space map disk space.

See Vacuum and Analyze for Query Optimization for information about vacuuming a database.

- `max_fsm_pages`
- `max_fsm_relations`

OS Resource Parameters

- `max_files_per_process`
- `shared_preload_libraries`

Cost-Based Vacuum Delay Parameters

Warning: Using cost-based vacuum delay is discouraged because it runs asynchronously among the segment instances. The vacuum cost limit and delay is invoked at the segment level without taking into account the state of the entire Greenplum array.

You can configure the execution cost of VACUUM and ANALYZE commands to reduce the I/O impact on concurrent database activity. When the accumulated cost of I/O operations reaches the limit, the process performing the operation sleeps for a while, then resets the counter and continues execution.

- `vacuum_cost_delay`
- `vacuum_cost_limit`
- `vacuum_cost_page_dirty`
- `vacuum_cost_page_hit`
- `vacuum_cost_page_miss`

Transaction ID Management Parameters

- `xid_stop_limit`
• **xid_warn_limit**

**Query Tuning Parameters**

**GPORCA Configuration Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimizer</td>
<td>optimizer_join_arity_for_associativity_commutativity</td>
</tr>
<tr>
<td>optimizer_analyze_root_partition</td>
<td>optimizer_join_order_threshold</td>
</tr>
<tr>
<td>optimizer_arrayExpansion_threshold</td>
<td>optimizer_mdcache_size</td>
</tr>
<tr>
<td>optimizer_cteInlining_bound</td>
<td>optimizer_metadata_caching</td>
</tr>
<tr>
<td>optimizer_control</td>
<td>optimizer_parallel_union</td>
</tr>
<tr>
<td>optimizer_enableMasterOnlyQueries</td>
<td>optimizer_print_missing_stats</td>
</tr>
<tr>
<td>optimizer_forceMultistage_agg</td>
<td>optimizer_print_optimization_stats</td>
</tr>
<tr>
<td>optimizer_forceThreeStageScalarDqa</td>
<td>optimizer_sort_factor</td>
</tr>
</tbody>
</table>

**Query Plan Operator Control Parameters**

The following parameters control the types of plan operations the legacy query optimizer can use. Enable or disable plan operations to force the legacy query optimizer to choose a different plan. This is useful for testing and comparing query performance using different plan types.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>enableBitmapScan</td>
<td>gp_enable_agg_distinct_pruning</td>
</tr>
<tr>
<td>enableGroupAgg</td>
<td>gp_enable_direct_dispatch</td>
</tr>
<tr>
<td>enableHashAgg</td>
<td>gp_enable_fallback_plan</td>
</tr>
<tr>
<td>enableHashJoin</td>
<td>gp_enable_fast_sri</td>
</tr>
<tr>
<td>enableIndexScan</td>
<td>gp_enable_groupext_distinct_gather</td>
</tr>
<tr>
<td>enableMergeJoin</td>
<td>gp_enable_groupext_distinct_pruning</td>
</tr>
<tr>
<td>enableNestLoop</td>
<td>gp_enable_multiphase_agg</td>
</tr>
<tr>
<td>enableSeqScan</td>
<td>gp_enable_predicate_propagation</td>
</tr>
<tr>
<td>enableSort</td>
<td>gp_enable_preunique</td>
</tr>
<tr>
<td>enableTidScan</td>
<td>gp_enable_sequential_window_plans</td>
</tr>
<tr>
<td>gp_enableAdaptiveNestLoop</td>
<td>gp_enable_sort_distinct</td>
</tr>
<tr>
<td>gp_enableAggDistinct</td>
<td>gp_enable_sort_limit</td>
</tr>
</tbody>
</table>

**Legacy Query Optimizer Costing Parameters**

**Warning:** Do not adjust these query costing parameters. They are tuned to reflect Greenplum Database hardware configurations and typical workloads. All of these parameters are related. Changing one without changing the others can have adverse effects on performance.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu_index_tuple_cost</td>
<td>gp_motion_cost_per_row</td>
</tr>
<tr>
<td>cpu_operator_cost</td>
<td>gp_segments_for_planner</td>
</tr>
<tr>
<td>cpu_tuple_cost</td>
<td>random_page_cost</td>
</tr>
<tr>
<td>cursor_tuple_fraction</td>
<td>seq_page_cost</td>
</tr>
</tbody>
</table>
**Database Statistics Sampling Parameters**

These parameters adjust the amount of data sampled by an `ANALYZE` operation. Adjusting these parameters affects statistics collection system-wide. You can configure statistics collection on particular tables and columns by using the `ALTER TABLE SET STATISTICS` clause.

- `default_statistics_target`
- `gp_analyze_relative_error`

**Sort Operator Configuration Parameters**

- `gp_enable_sort_distinct`
- `gp_enable_sort_limit`

**Aggregate Operator Configuration Parameters**

<table>
<thead>
<tr>
<th>gp_enable_agg_distinct</th>
<th>gp_enable_groupext_distinct__gather</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp_enable_agg_distinct_pruning</td>
<td>gp_enable_groupext_distinct__pruning</td>
</tr>
<tr>
<td>gp_enable_multiphase_agg</td>
<td>gp_workfile_compress_algorithm</td>
</tr>
<tr>
<td>gp_enable_preunique</td>
<td></td>
</tr>
</tbody>
</table>

**Join Operator Configuration Parameters**

<table>
<thead>
<tr>
<th>joinCollapse_limit</th>
<th>gp_statistics_use_fkeys</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp_adjust_selectivity_for_outerjoins</td>
<td>gp_workfile_compress_algorithm</td>
</tr>
<tr>
<td>gp_hashjoin_tuples_per_bucket</td>
<td></td>
</tr>
</tbody>
</table>

**Other Legacy Query Optimizer Configuration Parameters**

- `fromCollapse_limit`
- `gp_enable_predicate_propagation`
- `gp_max_plan_size`
- `gp_statistics_pullup_from_child_partition`

**Error Reporting and Logging Parameters**

**Log Rotation**

<table>
<thead>
<tr>
<th>log_rotation_age</th>
<th>log_truncate_on_rotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_rotation_size</td>
<td></td>
</tr>
</tbody>
</table>

**When to Log**

<table>
<thead>
<tr>
<th>client_min_messages</th>
<th>log_min_error_statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp_interconnect_debug_retry_interval</td>
<td>log_min_messages</td>
</tr>
<tr>
<td>log_error_verbosity</td>
<td>optimizer_minidump</td>
</tr>
</tbody>
</table>
### What to Log

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug_pretty_print</td>
<td>log_executor_stats</td>
</tr>
<tr>
<td>debug_print_parse</td>
<td>log_hostname</td>
</tr>
<tr>
<td>debug_print_plan</td>
<td>gp_log_interconnect</td>
</tr>
<tr>
<td>debug_print_prelim_plan</td>
<td>log_parser_stats</td>
</tr>
<tr>
<td>debug_print_rewritten</td>
<td>log_planner_stats</td>
</tr>
<tr>
<td>debug_print_slice_table</td>
<td>log_statement</td>
</tr>
<tr>
<td>log_autostats</td>
<td>log_statement_stats</td>
</tr>
<tr>
<td>log_connections</td>
<td>log_timezone</td>
</tr>
<tr>
<td>log_disconnections</td>
<td>gp_debug_linger</td>
</tr>
<tr>
<td>log_dispatch_stats</td>
<td>gp_log_format</td>
</tr>
<tr>
<td>log_duration</td>
<td>gp_log_gang</td>
</tr>
<tr>
<td></td>
<td>gp_max_csv_line_length</td>
</tr>
<tr>
<td></td>
<td>gp_reraise_signal</td>
</tr>
</tbody>
</table>

### System Monitoring Parameters

#### SNMP Alerts

The following parameters send SNMP notifications when events occur.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp_snmp_community</td>
<td>gp_snmp_use_inform_or_trap</td>
</tr>
<tr>
<td>gp_snmp_monitor_address</td>
<td></td>
</tr>
</tbody>
</table>

#### Email Alerts

The following parameters configure the system to send email alerts for fatal error events, such as a segment going down or a server crash and reset.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp_email_from</td>
<td>gp_email_smtp_userid</td>
</tr>
<tr>
<td>gp_email_smtp_password</td>
<td>gp_email_to</td>
</tr>
<tr>
<td>gp_email_smtp_server</td>
<td></td>
</tr>
</tbody>
</table>

#### Greenplum Command Center Agent

The following parameters configure the data collection agents that populate the `gpperfmon` database used by Greenplum Command Center.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp_enable_gpperfmon</td>
<td>gpperfmon_log_alert_level</td>
</tr>
<tr>
<td>gp_gpperfmon_send_interval</td>
<td>gpperfmon_port</td>
</tr>
</tbody>
</table>
**Runtime Statistics Collection Parameters**

These parameters control the server statistics collection feature. When statistics collection is enabled, you can access the statistics data using the *pg_stat* and *pg_statio* family of system catalog views.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>stats_queue_level</td>
<td></td>
</tr>
<tr>
<td>track_activities</td>
<td></td>
</tr>
<tr>
<td>track_counts</td>
<td></td>
</tr>
<tr>
<td>update_process_title</td>
<td></td>
</tr>
</tbody>
</table>

**Automatic Statistics Collection Parameters**

When automatic statistics collection is enabled, you can run `ANALYZE` automatically in the same transaction as an `INSERT`, `UPDATE`, `DELETE`, `COPY` or `CREATE TABLE...AS SELECT` statement when a certain threshold of rows is affected (on_change), or when a newly generated table has no statistics (on_no_stats). To enable this feature, set the following server configuration parameters in your Greenplum master *postgresql.conf* file and restart Greenplum Database:

- `gp_autostats_mode`
- `gp_autostats_mode_in_functions`
- `log_autostats`
- `gp_autostats_on_change_threshold`

**Warning:** Depending on the specific nature of your database operations, automatic statistics collection can have a negative performance impact. Carefully evaluate whether the default setting of `on_no_stats` is appropriate for your system.

**Client Connection Default Parameters**

**Statement Behavior Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>check_function_bodies</td>
<td></td>
</tr>
<tr>
<td>default_tablespace</td>
<td></td>
</tr>
<tr>
<td>default_transaction_isolation</td>
<td></td>
</tr>
<tr>
<td>default_transaction_read_only</td>
<td></td>
</tr>
<tr>
<td>search_path</td>
<td></td>
</tr>
<tr>
<td>statement_timeout</td>
<td></td>
</tr>
<tr>
<td>vacuum_freeze_min_age</td>
<td></td>
</tr>
</tbody>
</table>

**Locale and Formatting Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_encoding</td>
<td></td>
</tr>
<tr>
<td>DateStyle</td>
<td></td>
</tr>
<tr>
<td>extra_float_digits</td>
<td></td>
</tr>
<tr>
<td>IntervalStyle</td>
<td></td>
</tr>
<tr>
<td>lc_collate</td>
<td></td>
</tr>
<tr>
<td>lc_c_type</td>
<td></td>
</tr>
<tr>
<td>lc_messages</td>
<td></td>
</tr>
<tr>
<td>lc_monetary</td>
<td></td>
</tr>
<tr>
<td>lc_numeric</td>
<td></td>
</tr>
<tr>
<td>lc_time</td>
<td></td>
</tr>
<tr>
<td>TimeZone</td>
<td></td>
</tr>
</tbody>
</table>

**Other Client Default Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic_library_path</td>
<td></td>
</tr>
<tr>
<td>explain_pretty_print</td>
<td></td>
</tr>
<tr>
<td>local_preload_libraries</td>
<td></td>
</tr>
</tbody>
</table>
**Lock Management Parameters**

- deadlock_timeout
- max_locks_per_transaction

**Resource Management Parameters**

The following configuration parameters configure Greenplum Database resource queues, query prioritization, memory utilization and concurrency control.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp_resqueue_priority</td>
</tr>
<tr>
<td>gp_resqueue_priority_cpuscores_per_segment</td>
</tr>
<tr>
<td>gp_resqueue_priority_sweeper_interval</td>
</tr>
<tr>
<td>gp_vmem_idle_resource_timeout</td>
</tr>
<tr>
<td>gp_vmem_protect_limit</td>
</tr>
<tr>
<td>gp_vmem_protect_segworker_cache_limit</td>
</tr>
</tbody>
</table>

**External Table Parameters**

The following parameters configure the external tables feature of Greenplum Database.

See *Defining External Tables* for more information about external tables.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp_external_enable_exec</td>
</tr>
<tr>
<td>gp_external_max_segs</td>
</tr>
<tr>
<td>gp_initial_bad_row_limit</td>
</tr>
<tr>
<td>gp_reject_percent_threshold</td>
</tr>
<tr>
<td>readable_external_table_timeout</td>
</tr>
<tr>
<td>writable_external_table_bufsize</td>
</tr>
</tbody>
</table>

**Database Table Parameters**

The following parameter configures default option settings for Greenplum Database tables.

See *Creating and Managing Tables* for more information about Greenplum Database tables.

- gp_create_table_random_default_distribution
- gp_default_storage_options
- gp_enable_exchange_default_partition
- gp_enable_segment_copy_checking

**Append-Optimized Table Parameters**

The following parameters configure the append-optimized tables feature of Greenplum Database.

See *Append-Optimized Storage* for more information about append-optimized tables.

- gp_default_storage_options
- max_appendonly_tables
- gp_appendonly_compaction
- gp_appendonly_compaction_threshold
- validate_previous_free_tid
Database and Tablespace/Filespace Parameters

The following parameters configure the maximum number of databases, tablespaces, and filespaces allowed in a system.

- `gp_max_tablespaces`
- `gp_max_filespaces`
- `gp_max_databases`

Past PostgreSQL Version Compatibility Parameters

The following parameters provide compatibility with older PostgreSQL versions. You do not need to change these parameters in Greenplum Database.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add_missing_from</code></td>
<td><code>regex_flavor</code></td>
</tr>
<tr>
<td><code>array_nulls</code></td>
<td><code>standard_conforming_strings</code></td>
</tr>
<tr>
<td><code>backslash_quote</code></td>
<td><code>transform_null_equals</code></td>
</tr>
<tr>
<td><code>escape_string_warning</code></td>
<td></td>
</tr>
</tbody>
</table>

Greenplum Array Configuration Parameters

The parameters in this topic control the configuration of the Greenplum Database array and its components: segments, master, distributed transaction manager, master mirror, and interconnect.

Interconnect Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gp_interconnect_fc_method</code></td>
<td></td>
</tr>
<tr>
<td><code>gp_interconnect_hash_multiplier</code></td>
<td></td>
</tr>
<tr>
<td><code>gp_interconnect_queue_depth</code></td>
<td></td>
</tr>
<tr>
<td><code>gp_interconnect_setup_timeout</code></td>
<td></td>
</tr>
<tr>
<td><code>gp_interconnect_snd_queue_depth</code></td>
<td></td>
</tr>
<tr>
<td><code>gp_interconnect_type</code></td>
<td></td>
</tr>
<tr>
<td><code>gp_max_packet_size</code></td>
<td></td>
</tr>
</tbody>
</table>

Note: Greenplum Database supports only the UDPIFC and TCP interconnect types.

Dispatch Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gp_cached_segworkers_threshold</code></td>
<td></td>
</tr>
<tr>
<td><code>gp_connections_per_thread</code></td>
<td></td>
</tr>
<tr>
<td><code>gp_enable_direct_dispatch</code></td>
<td></td>
</tr>
<tr>
<td><code>gp_segment_connect_timeout</code></td>
<td></td>
</tr>
<tr>
<td><code>gp_set_proc_affinity</code></td>
<td></td>
</tr>
</tbody>
</table>

Fault Operation Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gp_set_read_only</code></td>
<td></td>
</tr>
<tr>
<td><code>gp_fts_probe_interval</code></td>
<td></td>
</tr>
<tr>
<td><code>gp_fts_probe_threadcount</code></td>
<td></td>
</tr>
</tbody>
</table>

Distributed Transaction Management Parameters

- `gp_max_local_distributed_cache`
**Read-Only Parameters**

- `gp_command_count`
- `gp_content`
- `gp_dbid`
- `gp_num_contents_in_cluster`
- `gp_role`
- `gp_session_id`

**Greenplum Master and Segment Mirroring Parameters**

These parameters control the configuration of the replication between Greenplum Database primary master and standby master.

- `keep_wal_segments`
- `repl_catchup_within_range`
- `replication_timeout`
- `wal_receiver_status_interval`

This parameter controls validation between Greenplum Database primary segment and standby segment during incremental resynchronization.

- `filerep_mirrorvalidation_during_resync`

**Greenplum Database Extension Parameters**

The parameters in this topic control the configuration of Greenplum Database extensions.

- `pljava_classpath`
- `pljava_classpath_insecure`
- `pljava_statement_cache_size`
- `pljava_release_lingering_savepoints`
- `pljava_vmoptions`

**XML Data Parameters**

- `xmlbinary`
- `xmloption`

**Enabling High Availability and Data Consistency Features**

The fault tolerance and the high-availability features of Greenplum Database can be configured.

**Important:** When data loss is not acceptable for a Pivotal Greenplum Database cluster, master and segment mirroring must be enabled in order for the cluster to be supported by Pivotal. Without mirroring, system and data availability is not guaranteed, Pivotal will make best efforts to restore a cluster in this case. For information about master and segment mirroring, see [About Redundancy and Failover](#).

For information about the utilities that are used to enable high availability, see the [Greenplum Database Utility Guide](#).
Overview of Greenplum Database High Availability

A Greenplum Database cluster can be made highly available by providing a fault-tolerant hardware platform, by enabling Greenplum Database high-availability features, and by performing regular monitoring and maintenance procedures to ensure the health of all system components.

Hardware components will eventually fail, whether due to normal wear or an unexpected circumstance. Loss of power can lead to temporarily unavailable components. A system can be made highly available by providing redundant standbys for components that can fail so that services can continue uninterrupted when a failure does occur. In some cases, the cost of redundancy is higher than users' tolerance for interruption in service. When this is the case, the goal is to ensure that full service is able to be restored, and can be restored within an expected timeframe.

With Greenplum Database, fault tolerance and data availability is achieved with:

- Hardware level RAID storage protection
- Data storage checksums
- Greenplum segment mirroring
- Master mirroring
- Dual clusters
- Database backup and restore

Hardware level RAID

A best practice Greenplum Database deployment uses hardware level RAID to provide high performance redundancy for single disk failure without having to go into the database level fault tolerance. This provides a lower level of redundancy at the disk level.

Data storage checksums

Greenplum Database uses checksums to verify that data loaded from disk to memory has not been corrupted on the file system.

Greenplum Database has two kinds of storage for user data: heap and append-optimized. Both storage models use checksums to verify data read from the file system and, with the default settings, they handle checksum verification errors in a similar way.

Greenplum Database primary and segment database processes update data on pages in the memory they manage. When a memory page is updated and flushed to disk, checksums are computed and saved with the page. When a page is later retrieved from disk, the checksums are verified and the page is only permitted to enter managed memory if the verification succeeds. A failed checksum verification is an indication of corruption in the file system and causes Greenplum Database to generate an error, aborting the transaction.

The default checksum settings provide the best level of protection from undetected disk corruption propagating into the database and to mirror segments.

Heap checksum support is enabled by default when the Greenplum Database cluster is initialized with the gpinit system management utility. Although it is strongly discouraged, a cluster can be initialized without heap checksum support by setting the HEAP_CHECKSUM parameter to off in the gpinit system cluster configuration file. See gpinit system.

Once initialized, it is not possible to change heap checksum support for a cluster without reinitializing the system and reloading databases.

You can check the read-only server configuration parameter data_checksums to see if heap checksums are enabled in a cluster:

```
$ gpconfig -s data_checksums
```
When a Greenplum Database cluster starts up, the `gpstart` utility checks that heap checksums are consistently enabled or disabled on the master and all segments. If there are any differences, the cluster fails to start. See `gpstart`.

In cases where it is necessary to ignore heap checksum verification errors so that data can be recovered, setting the `ignore_checksum_failure` system configuration parameter to on causes Greenplum Database to issue a warning when a heap checksum verification fails, but the page is then permitted to load into managed memory. If the page is updated and saved to disk, the corrupted data could be replicated to the mirror segment. Because this can lead to data loss, setting `ignore_checksum_failure` to on should only be done to enable data recovery.

For append-optimized storage, checksum support is one of several storage options set at the time an append-optimized table is created with the `CREATE TABLE` statement. The default storage options are specified in the `gp_default_storage_options` server configuration parameter. The checksum storage option is enabled by default and disabling it is strongly discouraged.

If you choose to disable checksums for an append-optimized table, you can either

- change the `gp_default_storage_options` configuration parameter to include `checksum=false` before creating the table, or
- add the `checksum=false` option to the `WITH storage_options` clause of the `CREATE TABLE` statement.

Note that the `CREATE TABLE` statement allows you to set storage options, including checksums, for individual partition files.

See the `CREATE TABLE` command reference and the `gp_default_storage_options` configuration parameter reference for syntax and examples.

**Segment Mirroring**

Greenplum Database stores data in multiple segments, each of which is a Greenplum Database Postgres instance. The data for each table is spread between the segments based on the distribution policy that is defined for the table in the DDL at the time the table is created. When segment mirroring is enabled, for each segment there is a `primary` and `mirror` pair. The primary and mirror perform the same IO operations and store copies of the same data.

The mirror instance for each segment is usually initialized with the `gpinit system` utility or the `gpexpand` utility. The mirror runs on a different host than the primary instance to protect from a single machine failure. There are different strategies for assigning mirrors to hosts. When choosing the layout of the primaries and mirrors, it is important to consider the failure scenarios to ensure that processing skew is minimized in the case of a single machine failure.

**Master Mirroring**

There are two masters in a highly available cluster, a `primary` and a `standby`. As with segments, the master and standby should be deployed on different hosts so that the cluster can tolerate a single host failure. Clients connect to the primary master and queries can be executed only on the primary master. The secondary master is kept up-to-date by replicating the write-ahead log (WAL) from the primary to the secondary.

If the master fails, the administrator runs the `gpactivatestandby` utility to have the standby master take over as the new primary master. You can configure a virtual IP address for the master and standby so that client programs do not have to switch to a different network address when the current master changes. If the master host fails, the virtual IP address can be swapped to the actual acting master.

**Dual Clusters**

An additional level of redundancy can be provided by maintaining two Greenplum Database clusters, both storing the same data.
Two methods for keeping data synchronized on dual clusters are “dual ETL” and “backup/restore.”

Dual ETL provides a complete standby cluster with the same data as the primary cluster. ETL (extract, transform, and load) refers to the process of cleansing, transforming, validating, and loading incoming data into a data warehouse. With dual ETL, this process is executed twice in parallel, once on each cluster, and is validated each time. It also allows data to be queried on both clusters, doubling the query throughput. Applications can take advantage of both clusters and also ensure that the ETL is successful and validated on both clusters.

To maintain a dual cluster with the backup/restore method, create backups of the primary cluster and restore them on the secondary cluster. This method takes longer to synchronize data on the secondary cluster than the dual ETL strategy, but requires less application logic to be developed. Populating a second cluster with backups is ideal in use cases where data modifications and ETL are performed daily or less frequently.

**Backup and Restore**

Making regular backups of the databases is recommended except in cases where the database can be easily regenerated from the source data. Backups should be taken to protect from operational, software, and hardware errors.

Use the `gpcrondump` utility to backup Greenplum databases. `gpcrondomp` performs the backup in parallel across segments, so backup performance scales up as hardware is added to the cluster.

When designing a backup strategy, a primary concern is where to store the backup data. The data each segment manages can be backed up on the segment's local storage, but should not be stored there permanently—the backup reduces disk space available to the segment and, more importantly, a hardware failure could simultaneously destroy the segment's live data and the backup. After performing a backup, the backup files should be moved from the primary cluster to separate, safe storage. Alternatively, the backup can be made directly to separate storage.

Additional options are available to backup databases, including the following:

**Data Domain**

Through native API integration backups can be streamed to a Dell EMC Data Domain appliance.

**NetBackup**

Through native API integration, backups can be streamed to a Veritas NetBackup cluster.

**NFS**

If an NFS mount is created on each Greenplum Database host in the cluster, backups can be written directly to the NFS mount. A scale out NFS solution is recommended to ensure that backups do not bottleneck on IO throughput of the NFS device. Dell EMC Isilon is an example that can scale out along side the Greenplum cluster.

**Incremental Backups**

Greenplum Database allows incremental backup at the partition level for append-optimized and column-oriented tables. When you perform an incremental backup, only the partitions for append-optimized and column-oriented tables that have changed since the previous backup are backed up. (Heap tables are always backed up.) Restoring an incremental backup requires restoring the previous full backup and subsequent incremental backups.

Incremental backup is beneficial only when the database contains large, partitioned tables where all but one or a few partitions remain unchanged between backups. An incremental backup saves just the changed partitions and the heap tables. By not backing up the unchanged partitions, the backup size and time can be significantly reduced.
If a large fact table is not partitioned, and a single row is added or changed, the entire table is backed up, and there is no savings in backup size or time. Therefore, incremental backup is only recommended for databases with large, partitioned tables and relatively small dimension tables.

If you maintain dual clusters and use incremental backup, you can populate the second cluster with the incremental backups. Use the `--noplan` option to achieve this, allowing backups from the primary site to be applied faster.

**Overview of Segment Mirroring**

When Greenplum Database High Availability is enabled, there are two types of segments: primary and mirror. Each primary segment has one corresponding mirror segment. A primary segment receives requests from the master to make changes to the segment's database and then replicates those changes to the corresponding mirror. If a primary segment becomes unavailable, database queries fail over to the mirror segment.

Segment mirroring employs a physical file replication scheme—data file I/O at the primary is replicated to the secondary so that the mirror's files are identical to the primary's files. Data in Greenplum Database are represented with tuples, which are packed into blocks. Database tables are stored in disk files consisting of one or more blocks. A change to a tuple changes the block it is saved in, which is then written to disk on the primary and copied over the network to the mirror. The mirror updates the corresponding block in its copy of the file.

For heap tables, blocks are saved in an in-memory cache until they are evicted to make room for newly changed blocks. This allows the system to read or update a block in memory multiple times without performing expensive disk I/O. When the block is evicted from the cache, it is written to disk and replicated to the secondary. While the block is held in cache, the primary and mirror have different images of the block. However, the databases are still consistent because the transaction log has been replicated. If a mirror takes over for a failed primary, the transactions in its log are applied to the database tables.

Other database objects — for example file spaces, which are tablespaces internally represented with directories—also use file replication to perform various file operations in a synchronous way.

Append-optimized tables do not use the in-memory caching mechanism. Changes made to append-optimized table blocks are replicated to the mirror immediately. Typically, file write operations are asynchronous, while opening, creating, and synchronizing files are "sync-replicated," which means the primary blocks until it receives the acknowledgment from the secondary.

If a primary segment fails, the file replication process stops and the mirror segment automatically starts as the active segment instance. The now active mirror's system state becomes Change Tracking, which means the mirror maintains a system table and change-log of all blocks updated while the primary segment is unavailable. When the failed primary segment is repaired and ready to be brought back online, an administrator initiates a recovery process and the system goes into Resynchronization state. The recovery process applies the logged changes to the repaired primary segment. The system state changes to Synchronized when the recovery process completes.

If the mirror segment fails or becomes inaccessible while the primary is active, the primary's system state changes to Change Tracking, and it tracks changes to be applied to the mirror when it is recovered.

Mirror segments can be placed on hosts in the cluster in different configurations, as long as the primary and mirror instance for a segment are on different hosts. Each host must have the same number of primary and mirror segments. The default mirroring configuration is group mirroring, where the mirror segments for each host's primary segments are placed on one other host. If a single host fails, the number of active primary segments doubles on the host that backs the failed host. Figure 17: Group Segment Mirroring in Greenplum Database illustrates a group mirroring configuration.
Spread mirroring spreads each host's mirrors over multiple hosts so that if any single host fails, no other host will have more than one mirror promoted to the active primary segment. Spread mirroring is possible only if there are more hosts than segments per host. 

Figure 17: Group Segment Mirroring in Greenplum Database

Figure 18: Spread Segment Mirroring in Greenplum Database illustrates the placement of mirrors in a spread segment mirroring configuration.
The Greenplum Database utilities that create mirror segments support group and spread segment configurations. Custom mirroring configurations can be described in a configuration file and passed on the command line.

**Overview of Master Mirroring**

You can deploy a backup or mirror of the master instance on a separate host machine or on the same host machine. A backup master or standby master serves as a warm standby if the primary master becomes nonoperational. You create a standby master from the primary master while the primary is online.

The primary master continues to provide service to users while a transactional snapshot of the primary master instance is taken. While the transactional snapshot is taken and deployed on the standby master, changes to the primary master are also recorded. After the snapshot is deployed on the standby master, the updates are deployed to synchronize the standby master with the primary master.

Once the primary master and standby master are synchronized, the standby master is kept up to date by the `walsender` and `walreceiver` replication processes. The `walreceiver` is a standby master process. The `walsender` process is a primary master process. The two processes use Write-Ahead Logging (WAL)-based streaming replication to keep the primary and standby masters synchronized. In WAL logging, all modifications are written to the log before being applied, to ensure data integrity for any in-process operations.

**Note:** WAL logging is not yet available for segment mirroring.

Since the master does not house user data, only system catalog tables are synchronized between the primary and standby masters. When these tables are updated, changes are automatically copied to the standby master to keep it current with the primary.
If the primary master fails, the replication process stops, and an administrator can activate the standby master. Upon activation of the standby master, the replicated logs reconstruct the state of the primary master at the time of the last successfully committed transaction. The activated standby then functions as the Greenplum Database master, accepting connections on the port specified when standby master was initialized.

**Overview of Fault Detection and Recovery**

The Greenplum Database server (postgres) subprocess named **ftsprobe** handles fault detection. **ftsprobe** monitors the Greenplum Database array; it connects to and scans all segments and database processes at intervals that you can configure.

If **ftsprobe** cannot connect to a segment, it marks the segment as "down" in the Greenplum Database system catalog. The segment remains nonoperational until an administrator initiates the recovery process.

With mirroring enabled, Greenplum Database automatically fails over to a mirror copy if a primary copy becomes unavailable. The system is operational if a segment instance or host fails provided all data is available on the remaining active segments.

To recover failed segments, an administrator runs the **gprecoverseg** recovery utility. This utility locates the failed segments, verifies they are valid, and compares the transactional state with the currently active segment to determine changes made while the segment was offline. **gprecoverseg** synchronizes the changed database files with the active segment and brings the segment back online. Administrators perform the recovery while Greenplum Database is up and running.

With mirroring disabled, the system automatically shuts down if a segment instance fails. Administrators manually recover all failed segments before operations resume.

See *Detecting a Failed Segment* for a more detailed description of the fault detection and recovery process and configuration options.

**Enabling Mirroring in Greenplum Database**

You can configure your Greenplum Database system with mirroring at setup time using **gpinit system** or enable mirroring later using **gpaddmirrors** and **gpinitstandby**. This topic assumes you are adding mirrors to an existing system that was initialized without mirrors.

You can enable the following types of mirroring:

- **Enabling Segment Mirroring**
- **Enabling Master Mirroring**
### Enabling Segment Mirroring

Mirror segments allow database queries to fail over to a backup segment if the primary segment is unavailable. By default, mirrors are configured on the same array of hosts as the primary segments. You may choose a completely different set of hosts for your mirror segments so they do not share machines with any of your primary segments.

**Important:** During the online data replication process, Greenplum Database should be in a quiescent state, workloads and other queries should not be running.

#### To add segment mirrors to an existing system (same hosts as primaries)

1. Allocate the data storage area for mirror data on all segment hosts. The data storage area must be different from your primary segments' file system location.
2. Use `gpssh-exkeys` to ensure that the segment hosts can SSH and SCP to each other without a password prompt.
3. Run the `gpaddmirrors` utility to enable mirroring in your Greenplum Database system. For example, to add 10000 to your primary segment port numbers to calculate the mirror segment port numbers:

   ```bash
   $ gpaddmirrors -p 10000
   
   Where `-p` specifies the number to add to your primary segment port numbers. Mirrors are added with the default group mirroring configuration.
   
#### To add segment mirrors to an existing system (different hosts from primaries)

1. Ensure the Greenplum Database software is installed on all hosts. See the *Greenplum Database Installation Guide* for detailed installation instructions.
2. Allocate the data storage area for mirror data on all segment hosts.
3. Use `gpssh-exkeys` to ensure the segment hosts can SSH and SCP to each other without a password prompt.
4. Create a configuration file that lists the host names, ports, and data directories on which to create mirrors. To create a sample configuration file to use as a starting point, run:

   ```bash
   $ gpaddmirrors -o filename
   
   The format of the mirror configuration file is:

   ```plaintext
   filespaceOrder=[filespace1 fsname[:filespace2 fsname:...]]
   mirror[content]=content:address:port:mir_replication_port:
   pri_replication_port:fselocation[:fselocation:...]
   
   For example, a configuration for two segment hosts and two segments per host, with no additional filespaces configured besides the default `pg_system` file space:

   ```plaintext
   filespaceOrder=
   mirror0=0:sdw1-1:52001:53001:54001:/gpdata/mir1/gp0
   mirror1=1:sdw1-2:52002:53002:54002:/gpdata/mir1/gp1
   mirror2=2:sdw2-1:52001:53001:54001:/gpdata/mir1/gp2
   mirror3=3:sdw2-2:52002:53002:54002:/gpdata/mir1/gp3
   
   5. Run the `gpaddmirrors` utility to enable mirroring in your Greenplum Database system:

   ```bash
   $ gpaddmirrors -i mirror_config_file
   ```
Where -i names the mirror configuration file you created.

**Enabling Master Mirroring**

You can configure a new Greenplum Database system with a standby master using `gpinitsystem` or enable it later using `gpinitstandby`. This topic assumes you are adding a standby master to an existing system that was initialized without one.

For information about the utilities `gpinitsystem` and `gpinitstandby`, see the *Greenplum Database Utility Guide*.

**To add a standby master to an existing system**

1. Ensure the standby master host is installed and configured: `gpadmin` system user created, Greenplum Database binaries installed, environment variables set, SSH keys exchanged, and data directory created. Greenplum Database

2. Run the `gpinitstandby` utility on the currently active primary master host to add a standby master host to your Greenplum Database system. For example:

   ```
   $ gpinitstandby -s smdw
   ```

   Where -s specifies the standby master host name.

3. To switch operations to a standby master, see *Recovering a Failed Master*.

**To check the status of the master mirroring process (optional)**

You can display the information in the Greenplum Database system view `pg_stat_replication`. The view lists information about the `wal_sender` process that is used for Greenplum Database master mirroring. For example, this command displays the process ID and state of the `wal_sender` process:

```
$ psql dbname -c 'SELECT procpid, state FROM pg_stat_replication;'
```

For information about the `pg_stat_replication` system view, see the *Greenplum Database Reference Guide*.

**Detecting a Failed Segment**

With mirroring enabled, Greenplum Database automatically fails over to a mirror segment when a primary segment goes down. Provided one segment instance is online per portion of data, users may not realize a segment is down. If a transaction is in progress when a fault occurs, the in-progress transaction rolls back and restarts automatically on the reconfigured set of segments.

If the entire Greenplum Database system becomes nonoperational due to a segment failure (for example, if mirroring is not enabled or not enough segments are online to access all user data), users will see errors when trying to connect to a database. The errors returned to the client program may indicate the failure. For example:

```
ERROR: All segment databases are unavailable
```

**How Segment Failure is Detected and Managed**

On the Greenplum Database master host, the Postgres `postmaster` process forks a fault probe process, `ftsprobe`. This is sometimes called the FTS (Fault Tolerance Server) process. The `postmaster` process restarts the FTS if it fails.

The FTS runs in a loop with a sleep interval between each cycle. On each loop, the FTS probes each primary segment database by making a TCP socket connection to the segment database using the hostname and port registered in the `gp_segment_configuration` table. If the connection succeeds, the
segment performs a few simple checks and reports back to the FTS. The checks include executing a `stat` system call on critical segment directories and checking for internal faults in the segment instance. If no issues are detected, a positive reply is sent to the FTS and no action is taken for that segment database.

If the connection cannot be made, or if a reply is not received in the timeout period, then a retry is attempted for the segment database. If the configured maximum number of probe attempts fail, the FTS probes the segment's mirror to ensure that it is up, and then updates the `gp_segment_configuration` table, marking the primary segment "down" and setting the mirror to act as the primary. The FTS updates the `gp_configuration_history` table with the operations performed.

When there is only an active primary segment and the corresponding mirror is down, the primary goes into "Change Tracking Mode." In this mode, changes to the segment are recorded, so the mirror can be synchronized without performing a full copy of data from the primary to the mirror.

The `gprecoverseg` utility is used to bring up a mirror that is down. By default, `gprecoverseg` performs an incremental recovery, placing the mirror into resync mode, which starts to replay the recorded changes from the primary onto the mirror. If the incremental recovery cannot be completed, the recovery fails and `gprecoverseg` should be run again with the `-F` option, to perform full recovery. This causes the primary to copy all of the data to the mirror.

You can see the mode—"change tracking", "resync", or "in-sync"—for each segment, as well as the status "up" or "down", in the `gp_segment_configuration` table.

The `gp_segment_configuration` table also has columns `role` and `preferred_role`. These can have values of either `p` for primary or `m` for mirror. The `role` column shows the segment database's current role and the `preferred_role` shows the original role of the segment. In a balanced system the `role` and `preferred_role` matches for all segments. When they do not match, there may be skew resulting from the number of active primary segments on each hardware host. To rebalance the cluster and bring all the segments into their preferred role, the `gprecoverseg` command can be run with the `-r` option.

There is a set of server configuration parameters that affect FTS behavior:

**gp_fts_probe_threadcount**
- The number of threads used for probing segments. Default: 16

**gp_fts_probe_interval**
- How often, in seconds, to begin a new FTS loop. For example if the setting is 60 and the probe loop takes 10 seconds, the FTS process sleeps 50 seconds. If the setting is 60 and probe loop takes 75 seconds, the process sleeps 0 seconds. The default is 60, and the maximum is 3600.

**gp_fts_probe_timeout**
- Probe timeout between master and segment, in seconds. The default is 20, and the maximum is 3600.

**gp_fts_probe_retries**
- The number of attempts to probe a segment. For example if the setting is 5 there will be 4 retries after the first attempt fails. Default: 5

**gp_log_fts**
- Logging level for FTS. The value may be "off", "terse", "verbose", or "debug". The "verbose" setting can be used in production to provide useful data for troubleshooting. The "debug" setting should not be used in production. Default: "terse"

**gp_segment_connect_timeout**
- The maximum time (in seconds) allowed for a mirror to respond. Default: 180

In addition to the fault checking performed by the FTS, a primary segment that is unable to send data to its mirror can change the status of the mirror to down. The primary queues up the data and after `gp_segment_connect_timeout` seconds passes, indicates a mirror failure, causing the mirror to be marked down and the primary to go into change tracking mode.
**Enabling Alerts and Notifications**

To receive notifications of system events such as segment failures, enable email or SNMP alerts. See [Enabling System Alerts and Notifications](#).

**Checking for Failed Segments**

With mirroring enabled, you may have failed segments in the system without interruption of service or any indication that a failure has occurred. You can verify the status of your system using the `gpstate` utility. `gpstate` provides the status of each individual component of a Greenplum Database system, including primary segments, mirror segments, master, and standby master.

**To check for failed segments**

1. On the master, run the `gpstate` utility with the `-e` option to show segments with error conditions:

   ```bash
   $ gpstate -e
   ```

   Segments in Change Tracking mode indicate the corresponding mirror segment is down. When a segment is not in its preferred role, the segment does not operate in the role to which it was assigned at system initialization. This means the system is in a potentially unbalanced state, as some segment hosts may have more active segments than is optimal for top system performance.

   See [Recovering From Segment Failures](#) for instructions to fix this situation.

2. To get detailed information about a failed segment, check the `gp_segment_configuration` catalog table. For example:

   ```bash
   $ psql -c "SELECT * FROM gp_segment_configuration WHERE status='d';"
   ```

3. For failed segment instances, note the host, port, preferred role, and data directory. This information will help determine the host and segment instances to troubleshoot.

4. To show information about mirror segment instances, run:

   ```bash
   $ gpstate -m
   ```

**Checking the Log Files for Failed Segments**

Log files can provide information to help determine an error's cause. The master and segment instances each have their own log file in `pg_log` of the data directory. The master log file contains the most information and you should always check it first.

Use the `gplogfilter` utility to check the Greenplum Database log files for additional information. To check the segment log files, run `gplogfilter` on the segment hosts using `gpssh`.

**To check the log files**

1. Use `gplogfilter` to check the master log file for WARNING, ERROR, FATAL or PANIC log level messages:

   ```bash
   $ gplogfilter -t
   ```

2. Use `gpssh` to check for WARNING, ERROR, FATAL, or PANIC log level messages on each segment instance. For example:

   ```bash
   $ gpssh -f seg_hosts_file -e 'source /usr/local/greenplum-db/greenplum_path.sh ; gplogfilter -t /data1/primary/*/pg_log/gpdb*.log' > seglog.out
   ```
Recovering a Failed Segment

If the master cannot connect to a segment instance, it marks that segment as down in the Greenplum Database system catalog. The segment instance remains offline until an administrator takes steps to bring the segment back online. The process for recovering a failed segment instance or host depends on the failure cause and whether or not mirroring is enabled. A segment instance can be unavailable for many reasons:

- A segment host is unavailable; for example, due to network or hardware failures.
- A segment instance is not running; for example, there is no postgres database listener process.
- The data directory of the segment instance is corrupt or missing; for example, data is not accessible, the file system is corrupt, or there is a disk failure.

Figure 20: Segment Failure Troubleshooting Matrix shows the high-level steps for each of the preceding failure scenarios.
Recovering From Segment Failures

Segment host failures usually cause multiple segment failures: all primary or mirror segments on the host are marked as down and nonoperational. If mirroring is not enabled and a segment goes down, the system automatically becomes nonoperational.

To recover with mirroring enabled

1. Ensure you can connect to the segment host from the master host. For example:

   $ ping failed_seg_host_address

2. Troubleshoot the problem that prevents the master host from connecting to the segment host. For example, the host machine may need to be restarted or replaced.

3. After the host is online and you can connect to it, run the `gprecoverseg` utility from the master host to reactivate the failed segment instances. For example:

   $ gprecoverseg

4. The recovery process brings up the failed segments and identifies the changed files that need to be synchronized. The process can take some time; wait for the process to complete. During this process, database write activity is suspended.

5. After `gprecoverseg` completes, the system goes into Resynchronizing mode and begins copying the changed files. This process runs in the background while the system is online and accepting database requests.

6. When the resynchronization process completes, the system state is Synchronized. Run the `gpstate` utility to verify the status of the resynchronization process:

   $ gpstate -m

To return all segments to their preferred role

When a primary segment goes down, the mirror activates and becomes the primary segment. After running `gprecoverseg`, the currently active segment remains the primary and the failed segment becomes the mirror. The segment instances are not returned to the preferred role that they were given at system initialization time. This means that the system could be in a potentially unbalanced state if segment hosts have more active segments than is optimal for top system performance. To check for unbalanced segments and rebalance the system, run:

$ gpstate -e

All segments must be online and fully synchronized to rebalance the system. Database sessions remain connected during rebalancing, but queries in progress are canceled and rolled back.

1. Run `gpstate -m` to ensure all mirrors are Synchronized.

   $ gpstate -m

2. If any mirrors are in Resynchronizing mode, wait for them to complete.

3. Run `gprecoverseg` with the `-r` option to return the segments to their preferred roles.

   $ gprecoverseg -r

4. After rebalancing, run `gpstate -e` to confirm all segments are in their preferred roles.

   $ gpstate -e
To recover from a double fault

In a double fault, both a primary segment and its mirror are down. This can occur if hardware failures on different segment hosts happen simultaneously. Greenplum Database is unavailable if a double fault occurs. To recover from a double fault:

1. Restart Greenplum Database:

   ```
   $ gpstop -r
   ```

2. After the system restarts, run gprecoverseg:

   ```
   $ gprecoverseg
   ```

3. After gprecoverseg completes, use gpstate to check the status of your mirrors:

   ```
   $ gpstate -m
   ```

4. If you still have segments in Change Tracking mode, run a full copy recovery:

   ```
   $ gprecoverseg -F
   ```

If a segment host is not recoverable and you have lost one or more segments, recreate your Greenplum Database system from backup files. See Backing Up and Restoring Databases.

To recover without mirroring enabled

1. Ensure you can connect to the segment host from the master host. For example:

   ```
   $ ping failed_seg_host_address
   ```

2. Troubleshoot the problem that is preventing the master host from connecting to the segment host. For example, the host machine may need to be restarted.

3. After the host is online, verify that you can connect to it and restart Greenplum Database. For example:

   ```
   $ gpstop -r
   ```

4. Run the gpstate utility to verify that all segment instances are online:

   ```
   $ gpstate
   ```

When a segment host is not recoverable

If a host is nonoperational, for example, due to hardware failure, recover the segments onto a spare set of hardware resources. If mirroring is enabled, you can recover a segment from its mirror onto an alternate host using gprecoverseg. For example:

```
$ gprecoverseg -i recover_config_file
```

Where the format of recover_config_file is:

```
filespaceOrder=[filespace1_name[:filespace2_name:...]]failed_host_address:port:fselocation [recovery_host_address:port:replication_port:fselocation [:fselocation:...]]
```

For example, to recover to a different host than the failed host without additional filespaces configured (besides the default pg_system filespace):

```
filespaceOrder=sdw5-2:50002:/gpdata/gpseg2 sdw9-2:50002:53002:/gpdata/gpseg2
```
The *gp_segment_configuration* and *pg_filespace_entry* system catalog tables can help determine your current segment configuration so you can plan your mirror recovery configuration. For example, run the following query:

```sql
=# SELECT dbid, content, hostname, address, port,
    replication_port, fselocation as datadir
FROM gp_segment_configuration, pg_filespace_entry
WHERE dbid=fsedbid
ORDER BY dbid;
```

The new recovery segment host must be pre-installed with the Greenplum Database software and configured exactly as the existing segment hosts.

**About the Segment Recovery Process**

This topic describes the process for recovering segments, initiated by the *gprecoverseg* management utility. It describes actions *gprecoverseg* performs and how the Greenplum File Replication and FTS (fault tolerance service) processes complete the recovery initiated with *gprecoverseg*.

Although *gprecoverseg* is an online operation, there are two brief periods during which all IO is paused. First, when recovery is initiated, IO is suspended while empty data files are created on the mirror. Second, after data files are synchronized, IO is suspended while system files such as transaction logs are copied from the primary to the mirror. The duration of these pauses is affected primarily by the number of file system files that must be created on the mirror and the sizes of the system flat files that must be copied. The system is online and available while database tables are replicated from the primary to the mirror, and any changes made to the database during recovery are replicated directly to the mirror.

To initiate recovery, the administrator runs the *gprecoverseg* utility. *gprecoverseg* prepares segments for recovery and initiates synchronization. When synchronization is complete and the segment status is updated in the system catalog, the segments are recovered. If the recovered segments are not running in their preferred roles, *gprecoverseg* `-r` can be used to bring the system back into balance.

Without the `-F` option, *gprecoverseg* recovers segments incrementally, copying only the changes since the mirror entered down status. The `-F` option fully recovers mirrors by deleting their data directories and then synchronizing all persistent data files from the primary to the mirror.

You can run *gpstate -e* to view the mirroring status of the segments before and during the recovery process. The primary and mirror segment statuses are updated as the recovery process proceeds.

Consider a single primary-mirror segment pair where the primary is active and the mirror is down. The following table shows the segment status before beginning recovery of the mirror.

<table>
<thead>
<tr>
<th></th>
<th>preferred_role</th>
<th>role</th>
<th>mode</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary</td>
<td>p</td>
<td>p</td>
<td>c</td>
<td>u</td>
</tr>
<tr>
<td></td>
<td>(primary)</td>
<td>(primary)</td>
<td>(change tracking)</td>
<td>(up)</td>
</tr>
<tr>
<td>Mirror</td>
<td>m</td>
<td>m</td>
<td>s</td>
<td>d</td>
</tr>
<tr>
<td></td>
<td>(mirror)</td>
<td>(mirror)</td>
<td>(synchronizing)</td>
<td>(down)</td>
</tr>
</tbody>
</table>

The segments are in their preferred roles, but the mirror is down. The primary is up and is in change tracking mode because it is unable to send changes to its mirror.

**Segment Recovery Preparation**

The *gprecoverseg* utility prepares the segments for recovery and then exits, allowing the Greenplum file replication processes to copy data from the primary to the mirror.

During the *gprecoverseg* execution the following recovery process steps are completed.
1. The down segments are identified.
2. The mirror segment processes are initialized.
3. For full recovery (-af):
   - The data directories of the down segments are deleted.
   - A new data directory structure is created.
4. The segment mode in the `gp_segment_configuration` system table is updated to 'r' (resynchronization mode).
5. The backend performs the following:
   - Suspends IO—connections to the master are allowed, but reads and writes from the segment being recovered are not allowed.
   - Scans persistent tables on the primary segment.
   - For each persistent file object (`relfilenode` in the `pg_class` system table), creates a data file on the mirror.

The greater the number of data files, the longer IO is suspended.

For incremental recovery, the IO is suspended for a shorter period because only file system objects added (or dropped) on the primary after the mirror was marked down need to be created (or deleted) on the mirror.

6. The `gprecoverseg` script completes.

Once `gprecoverseg` has completed, the segments are in the states shown in the following table.

<table>
<thead>
<tr>
<th></th>
<th>preferred_role</th>
<th>role</th>
<th>mode</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary</td>
<td>p (primary)</td>
<td>p (primary)</td>
<td>r (resynchronizing)</td>
<td>u (up)</td>
</tr>
<tr>
<td>Mirror</td>
<td>m (mirror)</td>
<td>m (mirror)</td>
<td>r (resynchronizing)</td>
<td>u (up)</td>
</tr>
</tbody>
</table>

**Data File Replication**

Data file resynchronization is performed in the background by file replication processes. Run `gpstate -e` to check the process of resynchronization. The Greenplum system is fully available for workloads while this process completes.

Following are steps in the resynchronization process:

1. Data copy (full and incremental recovery):

   After the file system objects are created, data copy is initiated for the affected segments. The ResyncManager process scans the persistent table system catalogs to find the file objects to be synchronized. ResyncWorker processes sync the file objects from the primary to the mirror.

2. Any changes or new data created with database transactions during the data copy are mirrored directly to the mirror.

3. Once data copy has finished synchronizing persistent data files, file replication updates the shared memory state on the current primary segment to 'insync'.

**Flat File Replication**

During this phase, system files in the primary segment's data directory are copied to the segment data directory. IO is suspended while the following flat files are copied from the primary data directory to the segment data directory:

- `pg_xlog/*`
- `pg_clog/*`
• pg_distributedlog/*
• pg_distributedxidmap/*
• pg_multixact/members
• pg_multixact/offsets
• pg_twophase/*
• global/pg_database
• global/pg_auth
• global/pg_auth_time_constraint

IOSUSPEND ends after these files are copied.

The next time the fault tolerance server (ftsprobe) process on the master wakes, it will set the primary and mirror states to synchronized (mode=s, state=u). A distributed query will also trigger the ftsprobe process to update the state.

When all segment recovery and file replication processes are complete, the segment status in the gp_segment_configuration system table and gp_state -e output is as shown in the following table.

<table>
<thead>
<tr>
<th>preferred_role</th>
<th>role</th>
<th>mode</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary</td>
<td>p</td>
<td>p</td>
<td>s</td>
</tr>
<tr>
<td></td>
<td>(primary)</td>
<td>(primary)</td>
<td>(synchronized)</td>
</tr>
<tr>
<td>Mirror</td>
<td>m</td>
<td>m</td>
<td>s</td>
</tr>
<tr>
<td></td>
<td>(mirror)</td>
<td>(mirror)</td>
<td>(synchronized)</td>
</tr>
</tbody>
</table>

Factors Affecting Duration of Segment Recovery

Following are some factors that can affect the duration of the segment recovery process.

• The number of database objects, mainly tables and indexes.
• The number of data files in segment data directories.
• The types of workloads updating data during resynchronization – both DDL and DML (insert, update, delete, and truncate).
• The size of the data.
• The size of system files, such as transaction log files, pg_database, pg_auth, and pg_auth_time_constraint.

Recovering a Failed Master

If the primary master fails, log replication stops. Use the gpstate -f command to check the state of standby replication. Use gpactivestandby to activate the standby master. Upon activation of the standby master, Greenplum Database reconstructs the master host state at the time of the last successfully committed transaction.

To activate the standby master

1. Ensure a standby master host is configured for the system. See Enabling Master Mirroring.
2. Run the gpactivestandby utility from the standby master host you are activating. For example:

   ```
   $ gpactivestandby -d /data/master/gpseg-1
   ```

   Where -d specifies the data directory of the master host you are activating.

   After you activate the standby, it becomes the active or primary master for your Greenplum Database array.
3. After the utility finishes, run `gpstate` to check the status:

   ```
   $ gpstate -f
   ```

   The newly activated master's status should be `Active`. If you configured a new standby host, its status is `Passive`. When a standby master is not configured, the command displays `-No entries found` and the message indicates that a standby master instance is not configured.

4. After switching to the newly active master host, run `ANALYZE` on it. For example:

   ```
   $ psql dbname -c 'ANALYZE;'
   ```

5. Optional: If you did not specify a new standby host when running the `gpactivatestandby` utility, use `gpinitstandby` to configure a new standby master at a later time. Run `gpinitstandby` on your active master host. For example:

   ```
   $ gpinitstandby -s new_standby_master_hostname
   ```

*Restoring Master Mirroring After a Recovery*

After you activate a standby master for recovery, the standby master becomes the primary master. You can continue running that instance as the primary master if it has the same capabilities and dependability as the original master host.

You must initialize a new standby master to continue providing master mirroring unless you have already done so while activating the prior standby master. Run `gpinitstandby` on the active master host to configure a new standby master.

You may restore the primary and standby master instances on the original hosts. This process swaps the roles of the primary and standby master hosts, and it should be performed only if you strongly prefer to run the master instances on the same hosts they occupied prior to the recovery scenario.

For information about the Greenplum Database utilities, see the *Greenplum Database Utility Guide*.

**To restore the master and standby instances on original hosts (optional)**

1. Ensure the original master host is in dependable running condition; ensure the cause of the original failure is fixed.

2. On the original master host, move or remove the data directory, `gpseg-1`. This example moves the directory to `backup_gpseg-1`:

   ```
   $ mv /data/master/gpseg-1 /data/master/backup_gpseg-1
   ```

   You can remove the backup directory once the standby is successfully configured.

3. Initialize a standby master on the original master host. For example, run this command from the current master host, `smdw`:

   ```
   $ gpinitstandby -s mdw
   ```

4. After the initialization completes, check the status of standby master, `mdw`, run `gpstate` with the `-f` option to check the status:

   ```
   $ gpstate -f
   ```

   The status should be `In Synch`. 
5. Stop Greenplum Database master instance on the standby master. For example:

   $ gpstop -m

6. Run the gpactivatestandby utility from the original master host, mdw, that is currently a standby master. For example:

   $ gpactivatestandby -d $MASTER_DATA_DIRECTORY

   Where the -d option specifies the data directory of the host you are activating.

7. After the utility completes, run gpstate to check the status:

   $ gpstate -f

   Verify the original primary master status is Active. When a standby master is not configured, the command displays -No entries found and the message indicates that a standby master instance is not configured.

8. On the standby master host, move or remove the data directory, gpseg-1. This example moves the directory:

   $ mv /data/master/gpseg-1 /data/master/backup_gpseg-1

   You can remove the backup directory once the standby is successfully configured.

9. After the original master host runs the primary Greenplum Database master, you can initialize a standby master on the original standby master host. For example:

   $ gpinitstandby -s smdw

**To check the status of the master mirroring process (optional)**

You can display the information in the Greenplum Database system view pg_stat_replication. The view lists information about the walsender process that is used for Greenplum Database master mirroring. For example, this command displays the process ID and state of the walsender process:

   $ psql dbname -c 'SELECT procpid, state FROM pg_stat_replication;'

---

**Backing Up and Restoring Databases**

This topic describes how to use Greenplum backup and restore features.

Performing backups regularly ensures that you can restore your data or rebuild your Greenplum Database system if data corruption or system failure occur. You can also use backups to migrate data from one Greenplum Database system to another.

**Backup and Restore Overview**

Greenplum Database supports parallel and non-parallel methods for backing up and restoring databases. Parallel operations scale regardless of the number of segments in your system, because segment hosts each write their data to local disk storage simultaneously. With non-parallel backup and restore operations, the data must be sent over the network from the segments to the master, which writes all of the data to its storage. In addition to restricting I/O to one host, non-parallel backup requires that the master have sufficient local disk storage to store the entire database.

**Parallel Backup and Restore**

The Greenplum Database parallel dump utility gpcrondump backs up the Greenplum master instance and each active segment instance at the same time.
By default, `gpcrondump` creates dump files in the `db_dumps` subdirectory of each segment instance. On the master, `gpcrondump` creates several dump files, containing database information such as DDL statements, the system catalog tables, and metadata files. On each segment, `gpcrondump` creates one dump file, which contains commands to recreate the data on that segment. Each file created for a backup begins with a 14-digit timestamp key that identifies the backup set the file belongs to.

**Figure 21: Parallel Backups in Greenplum Database**

The `gpdbrestore` parallel restore utility takes the timestamp key generated by `gpcrondump`, validates the backup set, and restores the database objects and data into a distributed database. Parallel restore operations require a complete backup set created by `gpcrondump`, a full backup, and any required incremental backups. As the following figure illustrates, all segments restore data from local backup files simultaneously.

**Figure 22: Parallel Restores in Greenplum Database**
The `gpdbrestore` utility provides flexibility and verification options for use with the automated backup files produced by `gpcrondump` or with backup files moved from the Greenplum cluster to an alternate location. See *Restoring Greenplum Databases*. `gpdbrestore` can also be used to copy files to the alternate location.

### Non-Parallel Backup and Restore

The PostgreSQL `pg_dump` and `pg_dumpall` non-parallel backup utilities can be used to create a single dump file on the master host that contains all data from all active segments.

The PostgreSQL non-parallel utilities should be used only for special cases. They are much slower than using the Greenplum backup utilities since all of the data must pass through the master. Additionally, it is often the case that the master host has insufficient disk space to save a backup of an entire distributed Greenplum database.

The `pgrestore` utility requires compressed dump files created by `pg_dump` or `pg_dumpall`. Before starting the restore, you should modify the `CREATE TABLE` statements in the dump files to include the Greenplum `DISTRIBUTED` clause. If you do not include the `DISTRIBUTED` clause, Greenplum Database assigns default values, which may not be optimal. For details, see `CREATE TABLE` in the *Greenplum Database Reference Guide*.

To perform a non-parallel restore using parallel backup files, you can copy the backup files from each segment host to the master host, and then load them through the master. See *Restoring to a Different Greenplum System Configuration*.

---

**Figure 23: Non-parallel Restore Using Parallel Backup Files**

Another non-parallel method for backing up Greenplum Database data is to use the `COPY TO SQL` command to copy all or a portion of a table out of the database to a delimited text file on the master host.

### Backup and Restore Options

The Greenplum Database backup and restore utilities support various locations for backup files:

- With the `gpcrondump` utility, backup files may be saved in the default location, the `db_dumps` subdirectory of the master and each segment, or saved to a different directory specified with the `gpcrondump -u` option.
- Both the `gpcrondump` and `gpdbrestore` utilities have integrated support for Dell EMC Data Domain Boost and Veritas NetBackup systems.
• Backup files can be saved through named pipes to any network accessible location.
• Backup files saved to the default location may be moved to an archive server on the network. This allows performing the backup at the highest transfer rates (when segments write the backup data to fast local disk arrays) and then freeing up disk space by moving the files to remote storage.

You can create dumps containing selected database objects:
• You can backup tables belonging to one or more schema you specify on the command line or in a text file.
• You can specify schema to exclude from the backup, as command-line options or in a list provided in a text file.
• You can backup a specified set of tables listed on the command line or in a text file. The table and schema options cannot be used together in a single backup.
• In addition to database objects, gpcrondump can backup the configuration files pg_hba.conf, pg_ident.conf, and postgresql.conf, and global database objects, such as roles and tablespaces.

You can create incremental backups:
• An incremental backup contains only append-optimized and column-oriented tables that have changed since the most recent incremental or full backup.
• For partitioned append-optimized tables, only changed append-optimized/column-oriented table partitions are backed up.
• Incremental backups include all heap tables.
• Use the gpcrondump --incremental flag to specify an incremental backup.
• Restoring an incremental backup requires a full backup and all subsequent incremental backups, up to the backup you are restoring.

The gpdbrestore utility offers many options:
• By default, gpdbrestore restores data to the database it was backed up from.
• The --redirect flag allows you to restore a backup to a different database.
• The restored database can be dropped and recreated, but the default is to restore into an existing database.
• Selected tables can be restored from a backup by listing the tables on the command line or by listing them in a text file and specifying the text file on the command line.
• You can restore a database from backup files moved to an archive server. The backup files are copied back into place on the master host and each segment host and then restored to the database.

**Backing Up with gpcrondump**

Use gpcrondump to backup databases, data, and objects such as database roles and server configuration files.

The gpcrondump utility dumps the contents of a Greenplum database to SQL script files on the master and each segment. The script files can then be used to restore the database.

The master backup files contain SQL commands to create the database schema. The segment data dump files contain SQL statements to load the data into the tables. The segment dump files are compressed using gzip. Optionally, the server configuration files postgresql.conf, pg_ident.conf, and pg_hba.conf and global data such as roles and tablespaces can be included in a backup.

The gpcrondump utility has one required flag, -x, which specifies the database to dump:

```
gpcrondump -x mydb
```

This performs a full backup of the specified database to the default locations.

**Note:** By default, the utility creates the public.gpcrondump_history table that contains details of the database dump. If the public schema has been deleted from the database, you must
specify the -H option to prevent gpcrondump from returning an error when it attempts to create the table.

By default, gpcrondump creates the backup files in the data directory on the master and each segment instance in the data_directory/db_dumps directory. You can specify a different backup location using the -u flag. For example, the following command will save backup files to the /backups directory:

```bash
gpcrondump mydb -u /backups
```

The gpcrondump utility creates the db_dumps subdirectory in the specified directory. If there is more than one primary segment per host, all of the segments on the host write their backup files to the same directory. This differs from the default, where each segment writes backups to its own data directory. This can be used to consolidate backups to a single directory or mounted storage device.

In the db_dumps directory, backups are saved to a directory in the format YYYYMMDD, for example data_directory/db_dumps/20151012 for a backup created on October 12, 2015. The backup file names in the directory contain a full timestamp for the backup in the format YYYYMMDDHHMMSS, for example gp_dump_0_2_20151012195916.gz. The gpdbrestore command uses the most recent backup by default but you can specify an earlier backup to restore.

The utility creates backup files with this file name format.

```
prefix_gdump_content_dbid_timestamp
```

The content and dbid are identifiers for the Greenplum Database segment instances that are assigned by Greenplum Database. For information about the identifiers, see the Greenplum Database system catalog table gp_id in the Greenplum Database Reference Guide.

If you include the -g option, gpcrondump saves the configuration files with the backup. These configuration files are dumped in the master or segment data directory to db_dumps/YYYYMMDD/config_files_timestamp.tar. If --ddboost is specified, the backup is located on the default storage unit in the directory specified by --ddboost-backupdir when the Data Domain Boost credentials were set. The -G option backs up global objects such as roles and tablespaces to a file in the master backup directory named gp_global_-1_1_timestamp.

If --ddboost is specified, the backup is located on the default storage unit in the directory specified by --ddboost-backupdir when the Data Domain Boost credentials were set.

There are many more gpcrondump options available to configure backups. Refer to the Greenplum Utility Reference Guide for information about all of the available options. See Backing Up Databases with Data Domain Boost for details of backing up with Data Domain Boost.

Warning: Backing up a database with gpcrondump while simultaneously running ALTER TABLE might cause gpcrondump to fail.

Backing up a database with gpcrondump while simultaneously running DDL commands might cause issues with locks. You might see either the DDL command or gpcrondump waiting to acquire locks.

Back Up a Set of Tables

You can create a backup that includes a subset of the schema or tables in a database by using the following gpcrondump options:

- `-t schema.tablename` — specify a table to include in the backup. You can use the `-t` option multiple times.
- `--table-file=filename` — specify a file containing a list of tables to include in the backup.
- `-T schema.tablename` — specify a table to exclude from the backup. You can use the `-T` option multiple times.
- `--exclude-table-file=filename` — specify a file containing a list of tables to exclude from the backup.
• `-s schema_name` – include all tables qualified by a specified schema name in the backup. You can use the `-s` option multiple times.
• `--schema-file=filename` – specify a file containing a list of schemas to include in the backup.
• `-S schema_name` – exclude tables qualified by a specified schema name from the backup. You can use the `-S` option multiple times.
• `--exclude-schema-file=filename` – specify a file containing schema names to exclude from the backup.

Only a set of tables or set of schemas can be specified. For example, the `-s` option cannot be specified with the `-t` option.

Refer to Incremental Backup with Sets for additional information about using these `gpcrondump` options with incremental backups.

### Creating Incremental Backups

The `gpcrondump` and `gpdbrestore` utilities support incremental backups and restores of append-optimized tables, including column-oriented tables. Use the `gpcrondump` option `--incremental` to create an incremental backup.

An incremental backup only backs up an append-optimized or column-oriented table if one of the following operations was performed on the table after the last full or incremental backup:

- `ALTER TABLE`
- `DELETE`
- `INSERT`
- `TRUNCATE`
- `UPDATE`
- `DROP` and then re-create the table

For partitioned append-optimized tables, only the changed partitions are backed up.

Heap tables are backed up with every full and incremental backup.

Incremental backups are efficient when the total amount of data in append-optimized table partitions or column-oriented tables that changed is small compared to the data has not changed.

Each time `gpcrondump` runs, it creates state files that contain row counts for each append optimized and column-oriented table and partition in the database. The state files also store metadata operations such as truncate and alter. When `gpcrondump` runs with the `--incremental` option, it compares the current state with the stored state to determine if the table or partition should be included in the incremental backup.

A unique 14-digit timestamp key identifies files that comprise an incremental backup set.

To create an incremental backup or to restore data from an incremental backup, you need the complete backup set. A complete backup set consists of a full backup and any incremental backups that were created since the last full backup. When you archive incremental backups, all incremental backups between the last full backup and the target incremental backup must be archived. You must archive all the files created on the master and all segments.

**Important:** For incremental backup sets, a full backup and associated incremental backups, the backup set must be on a single device. For example, a backup set must all be on a Data Domain system. The backup set cannot have some backups on a Data Domain system and others on the local file system or a NetBackup system.

**Note:** You can use a Data Domain server as an NFS file system (without Data Domain Boost) to perform incremental backups.

Changes to the Greenplum Database segment configuration invalidate incremental backups. After you change the segment configuration you must create a full backup before you can create an incremental backup.
Incremental Backup Example

Each backup set has a key, which is a timestamp taken when the backup is created. For example, if you create a backup on May 14, 2016, the backup set file names contain 20160514hhmmss. The hhmmss represents the time: hour, minute, and second.

For this example, assume you have created both full and incremental backups of the database mytest. To create the full backup, you used the following command:

```bash
gpcrondump -x mytest -u /backupdir
```

Later, after some changes have been made to append optimized tables, you created an increment backup with the following command:

```bash
gpcrondump -x mytest -u /backupdir --incremental
```

When you specify the -u option, the backups are created in the /backupdir directory on each Greenplum Database host. The file names include the following timestamp keys. The full backups have the timestamp key 20160514054532 and 20161114064330. The other backups are incremental backups.

- 20160514054532 (full backup)
- 20160714095512
- 20160914081205
- 20161114064330 (full backup)
- 20170114051246

To create a new incremental backup, you need both the most recent incremental backup 20170114051246 and the preceding full backup 20161114064330. Also, you must specify the same -u option for any incremental backups that are part of the backup set.

To restore a database with the incremental backup 20160914081205, you need the incremental backups 20160914081205 and 20160714095512, and the full backup 20160514054532.

To restore the mytest database with the incremental backup 20170114051246, you need only the incremental backup and the full backup 20161114064330. The restore command would be similar to this command.

```bash
gpdbrestore -t 20170114051246 -u /backupdir
```

Incremental Backup with Sets

To back up a set of database tables with incremental backup, identify the backup set with the --prefix option when you create the full backup with gpcrondump. For example, to create incremental backups for tables in the myschema schema, first create a full backup with a prefix, such as myschema:

```bash
gpcrondump -x mydb -s myschema --prefix myschema
```

The -s option specifies that tables qualified by the myschema schema are to be included in the backup. See Backing Up a Set of Tables for more options to specify a set of tables to back up.

Once you have a full backup you can create an incremental backup for the same set of tables by specifying the gpcrondump --incremental and --prefix options, specifying the prefix you set for the full backup. The incremental backup is automatically limited to only the tables in the full backup. For example:

```bash
gpcrondump -x mydb --incremental --prefix myschema
```

The following command lists the tables that were included or excluded for the full backup.

```bash
gpcrondump -x mydb --incremental --prefix myschema --list-filter-tables
```


**Restoring From an Incremental Backup**

When restoring a backup with `gpdbrestore`, the command-line output displays whether the restore type is incremental or a full database restore. You do not have to specify that the backup is incremental. For example, the following `gpdbrestore` command restores the most recent backup of the `mydb` database. `gpdbrestore` searches the `db_dumps` directory to locate the most recent dump and displays information about the backup it found.

```bash
$ gpdbrestore -s mydb
...
20151015:20:10:34:002664 gpdbrestore:mdw:gpadmin-[INFO]:-------------------------------------------
20151015:20:10:34:002664 gpdbrestore:mdw:gpadmin-[INFO]:-Greenplum database
restore parameters
20151015:20:10:34:002664 gpdbrestore:mdw:gpadmin-[INFO]:-Restore type
    = Incremental Restore
20151015:20:10:34:002664 gpdbrestore:mdw:gpadmin-[INFO]:-Database to be
restored    = mydb
20151015:20:10:34:002664 gpdbrestore:mdw:gpadmin-[INFO]:-Drop and re-create
db      = Off
20151015:20:10:34:002664 gpdbrestore:mdw:gpadmin-[INFO]:-Restore method
    = Search for latest
20151015:20:10:34:002664 gpdbrestore:mdw:gpadmin-[INFO]:-Restore timestamp
    = 20151014194445
20151015:20:10:34:002664 gpdbrestore:mdw:gpadmin-[INFO]:-Restore compressed
dump    = On
20151015:20:10:34:002664 gpdbrestore:mdw:gpadmin-[INFO]:-Restore global
objects     = Off
20151015:20:10:34:002664 gpdbrestore:mdw:gpadmin-[INFO]:-Array fault
tolerance      = f
20151015:20:10:34:002664 gpdbrestore:mdw:gpadmin-[INFO]:-------------------------------------------

Continue with Greenplum restore Yy|Nn (default=N):
```

`gpdbrestore` ensures that the full backup and other required incremental backups are available before restoring the backup. With the `--list-backup` option you can display the full and incremental backup sets required to perform a restore.

If the `gpdbrestore` option `--q` is specified, the backup type information is written to the log file. With the `gpdbrestore` option `--noplan`, you can restore only the data contained in an incremental backup.

**Backup Process and Locks**

During backups, the `gpcrondump` utility locks the `pg_class` system table and the tables that are backed up. Locking the `pg_class` table with an EXCLUSIVE lock ensures that no tables are added, deleted, or altered until `gpcrondump` locks tables that are to be backed up with ACCESS SHARE locks.

The steps below describe the process `gpcrondump` follows to dump databases, including what happens before locks are placed, while locks are held, and after locks are removed.

If more than one database is specified, this process is executed for each database in sequence.

- `gpcrondump` parses the command-line arguments and verifies that options and arguments are properly specified.
- If any of the following filter options are specified, `gpcrondump` prepares filters to determine the set of tables to back up. Otherwise, all tables are included in the backup.
- \( -s \) `schema_name` – Includes all the tables qualified by the specified schema.
- \( -S \) `schema_name` – Excludes all tables qualified by the specified schema.
- \( --schema-file=filename \) – Includes all the tables qualified by the schema listed in `filename`.
- \( --exclude-schema-file=filename \) – Excludes all tables qualified by the schema listed in `filename`.
- \( -t \) `schema.table_name` – Dumps the specified table.
- \( -T \) `schema.table_name` – Excludes the specified table.
- \( --table-file=filename \) – Dumps the tables specified in `filename`.
- \( --exclude-table-file=filename \) – Dumps all tables except tables specified in `filename`.

`gpcrondump` verifies the backup targets:

- Verifies that the database exists.
- Verifies that specified tables or schemas to be dumped exist.
- Verifies that the current primary for each segment is up.
- Verifies that the backup directory exists and is writable for the master and each segment. Note that if more than one segment is backed up to the same disk volume, this disk space check cannot ensure there is space available for all segments.
- Places an EXCLUSIVE lock on the catalog table `pg_class`. The lock permits only concurrent read operations on a table. While the lock is on the catalog table, relations such as tables, indexes, and views cannot be created, altered, or dropped in the database.

`gpcrondump` starts a thread to watch for a lock file (`dump_dir/gp_lockfile_timestamp`) to appear, signaling the parallel backup on the segments has completed.

`gpcrondump` locks tables that are to be backed up with an ACCESS SHARE lock.

An ACCESS SHARE lock only conflicts with an ACCESS EXCLUSIVE lock. The following SQL statements acquire an ACCESS EXCLUSIVE lock:

- `ALTER TABLE`
- `CLUSTER`
- `DROP TABLE`
- `REINDEX`
- `TRUNCATE`
- `VACUUM FULL`

- Threads are created and dispatched for the master and segments to perform the database dump.
- When the threads have acquired ACCESS SHARED locks on all the required tables, the `dump_dir/gp_lockfile_timestamp` lock file is created, signaling `gpcrondump` to release the EXCLUSIVE lock on the `pg_class` catalog table, while tables are being backed up.

`gpcrondump` checks the status files for each primary segment for any errors to report. If a dump fails and the \(-r\) flag was specified, the backup is rolled back.

- The ACCESS SHARE locks on the target tables are released.
- If the backup succeeded and a post-dump script was specified with the \(-R\) option, `gpcrondump` runs the script now.
- `gpcrondump` reports the backup status, sends email if configured, and saves the backup status in the `public.gpcrondump_history` table in the database.

**Using Direct I/O**

The operating system normally caches file I/O operations in memory because memory access is much faster than disk access. The application writes to a block of memory that is later flushed to the storage device, which is usually a RAID controller in a Greenplum Database system. Whenever the application accesses a block that is still resident in memory, a device access is avoided. Direct I/O allows you to bypass the cache so that the application writes directly to the storage device. This reduces CPU
consumption and eliminates a data copy operation. Direct I/O is efficient for operations like backups where file blocks are only handled once.

**Note:** Direct I/O is supported only on Red Hat, CentOS, and SUSE.

### Turning on Direct I/O

Set the `gp_backup_directIO` system configuration parameter on to enable direct I/O for backups:

```
$ gpconfig -c gp_backup_directIO -v on
```

To see if direct I/O is enabled, use this command:

```
$ gpconfig -s gp_backup_directIO
```

### Decrease network data chunks sent to dump when the database is busy

The `gp_backup_directIO_read_chunk_mb` configuration parameter sets the size, in MB, of the I/O chunk when direct I/O is enabled. The default chunk size, 20MB, has been tested and found to be optimal. Decreasing it increases the backup time and increasing it results in little change to backup time.

To find the current direct I/O chunk size, enter this command:

```
$ gpconfig -s gp_backup_directIO_read_chunk_mb
```

The following example changes the default chunk size to 10MB.

```
$ gpconfig -c gp_backup_directIO_read_chunk_mb -v 10
```

### Using Named Pipes

Greenplum Database allows the use of named pipes with `gpcrondump` and `gpdbrestore` to back up and restore Greenplum databases. When backing up a database with regular files, the files that contain the backup information are placed in directories on the Greenplum Database segments. If segment hosts do not have enough local disk space available to backup to files, you can use named pipes to back up to non-local storage, such as storage on another host on the network or to a backup appliance.

Backing up with named pipes is not supported if the `--ddboost` option is specified.

**To back up a Greenplum database using named pipes**

1. Run the `gpcrondump` command with options `-K timestamp and --list-backup-files`.

   This creates two text files that contain the names of backup files, one per line. The file names include the `timestamp` you specified with the `-K timestamp` option and have the suffixes `_pipes` and `_regular_files`. For example:

   ```
   gp_dump_20150519160000_pipes
   gp_dump_20150519160000_regular_files
   ```

   The file names listed in the `_pipes` file are to be created as named pipes. The file names in the `_regular_files` file should not be created as named pipes. `gpcrondump` and `gpdbrestore` use the information in these files during backup and restore operations.

2. Create named pipes on all Greenplum Database segments using the file names in the generated `_pipes` file.

3. Redirect the output of each named pipe to the destination process or file object.

4. Run `gpcrondump` to back up the database using the named pipes.
To create a complete set of Greenplum Database backup files, the files listed in the \_regular\_files file must also be backed up.

To restore a database that used named pipes during backup

1. Direct the contents of each backup file to the input of its named pipe, for example `cat filename > pipename`, if the backup file is accessible as a local file object.
2. Run the `gpdbrestore` command to restore the database using the named pipes.

**Example**

This example shows how to back up a database over the network using named pipes and the `netcat (nc)` Linux command. The segments write backup files to the inputs of the named pipes. The outputs of the named pipes are piped through `nc` commands, which make the files available on TCP ports. Processes on other hosts can then connect to the segment hosts at the designated ports to receive the backup files. This example requires that the `nc` package is installed on all Greenplum hosts.

1. Enter the following `gpcrondump` command to generate the lists of backup files for the `testdb` database in the `/backups` directory.

   ```bash
   $ gpcrondump -x testdb -K 20150519160000 --list-backup-files -u /backups
   ```

2. View the files that `gpcrondump` created in the `/backups` directory:

   ```bash
   $ ls -lR /backups
   /backups: total 4
   drwxrwxr-x 3 gpadmin gpadmin 4096 May 19 21:49 db_dumps
   /backups/db_dumps: total 4
   drwxrwxr-x 2 gpadmin gpadmin 4096 May 19 21:49 20150519
   /backups/db_dumps/20150519: total 8
   -rw-rw-r-- 1 gpadmin gpadmin 256 May 19 21:49 gp_dump_20150519160000_pipes
   -rw-rw-r-- 1 gpadmin gpadmin 391 May 19 21:49 gp_dump_20150519160000_regular_files
   ```

3. View the contents of the `\_pipes` file.

   ```bash
   $ cat /backups/db_dumps/20150519/gp_dump_20150519160000_pipes
   sdw1:/backups/db_dumps/20150519/gp_dump_0_2_20150519160000.gz
   sdw2:/backups/db_dumps/20150519/gp_dump_1_3_20150519160000.gz
   mdw:/backups/db_dumps/20150519/gp_dump_-1_1_20150519160000.gz
   mdw:/backups/db_dumps/20150519/gp_dump_-1_1_20150519160000_post_data.gz
   ```

4. Create the specified named pipes on the Greenplum Database segments. Also set up a reader for the named pipe.

   ```bash
   gpsssh -h sdw1
   [sdw1] mkdir -p /backups/db_dumps/20150519/
   [sdw1] mkfifo /backups/db_dumps/20150519/gp_dump_0_2_20150519160000.gz
   [sdw1] cat /backups/db_dumps/20150519/gp_dump_0_2_20150519160000.gz | nc -l 21000
   [sdw1] exit
   ```

   Complete these steps for each of the named pipes listed in the `\_pipes` file. Be sure to choose an available TCP port for each file.
5. On the destination hosts, receive the backup files with commands like the following:

```plaintext
nc sdwl 21000 > gp_dump_0_2_20150519160000.gz
```

6. Run `gpcrondump` to begin the backup:

```plaintext
gpcrondump -x testdb -K 20150519160000 -u /backups
```

To restore a database with named pipes, reverse the direction of the backup files by sending the contents of the backup files to the inputs of the named pipes and run the `gpdbrestore` command:

```plaintext
gpdbrestore -x testdb -t 20150519160000 -u /backups
```

`gpdbrestore` reads from the named pipes' outputs.

### Backing Up Databases with Data Domain Boost

Dell EMC Data Domain Boost (DD Boost) is Dell EMC software that can be used with the `gpcrondump` and `gpdbrestore` utilities to perform faster backups to the Dell EMC Data Domain storage appliance. Data Domain performs deduplication on the data it stores, so after the initial backup operation, the appliance stores only pointers to data that is unchanged. This reduces the size of backups on disk. When DD Boost is used with `gpcrondump`, Greenplum Database participates in the deduplication process, reducing the volume of data sent over the network. When you restore files from the Data Domain system with Data Domain Boost, some files are copied to the master local disk and are restored from there, and others are restored directly.

With Data Domain Boost managed file replication, you can replicate Greenplum Database backup images that are stored on a Data Domain system for disaster recover purposes. The `gpmfr` utility manages the Greenplum Database backup sets that are on the primary and a remote Data Domain system. For information about `gpmfr`, see the *Greenplum Database Utility Guide*.

Managed file replication requires network configuration when a replication network is being used between two Data Domain systems:

- The Greenplum Database system requires the Data Domain login credentials to be configured using `gpcrondump`. Credentials must be created for both the local and remote Data Domain systems.
- When the non-management network interface is used for replication on the Data Domain systems, static routes must be configured on the systems to pass the replication data traffic to the correct interfaces.

Do not use Data Domain Boost with `pg_dump` or `pg_dumpall`.

Refer to Data Domain Boost documentation for detailed information.

**Important:** For incremental back up sets, a full backup and the associated incremental backups must be on a single device. For example, a backup set must all be on a file system. The backup set cannot have some backups on the local file system and others on single storage unit of a Data Domain system. For backups on a Data Domain system, the backup set must be in a single storage unit.

**Note:** You can use a Data Domain server as an NFS file system (without Data Domain Boost) to perform incremental backups.

### Data Domain Boost Requirements

Using Data Domain Boost requires the following:

- Data Domain Boost is included only with the commercial release of Pivotal Greenplum Database.
- Purchase and install Dell EMC Data Domain Boost and Replicator licenses on the Data Domain systems.
• Obtain sizing recommendations for Data Domain Boost. Make sure the Data Domain system supports sufficient write and read streams for the number of segment hosts in your Greenplum cluster.

Contact your Dell EMC Data Domain account representative for assistance.

**One-Time Data Domain Boost Credential Setup**

There is a one-time process to set up credentials to use Data Domain Boost. Credential setup connects one Greenplum Database instance to one Data Domain instance. If you are using the gpcrondump --replicate option or DD Boost managed file replication capabilities for disaster recovery purposes, you must set up credentials for both the local and remote Data Domain systems.

To set up credentials, run gpcrondump with the following options:

```
--ddboost-host ddboost_hostname
--ddboost-user ddboost_user
--ddboost-backupdir backup_directory
--ddboost-storage-unit storage_unit_ID
```

The `--ddboost-storage-unit` is optional. If not specified, the storage unit ID is GPDB.

To manage credentials for the remote Data Domain system that is used for backup replication, include the `--ddboost-remote` option with the other gpcrondump options. For example, the following options set up credentials for a Data Domain system that is used for backup replication. The system IP address is 192.0.2.230, the user ID is ddboostmyuser, and the location for the backups on the system is GPDB/gp_production:

```
--ddboost-host 192.0.2.230
--ddboost-user ddboostmyuser
--ddboost-backupdir gp_production
--ddboost-remote
```

For details, see gpcrondump in the Greenplum Database Utility Guide.

If you use two or more network connections to connect to the Data Domain system, use gpcrondump to set up the login credentials for the Data Domain hostnames associated with the network interfaces. To perform this setup for two network connections, run gpcrondump with the following options:

```
--ddboost-host ddboost_hostname1
--ddboost-host ddboost_hostname2
--ddboost-user ddboost_user
--ddboost-backupdir backup_directory
```

**About DD Boost Credential Files**

The gpcrondump utility is used to schedule DD Boost backup operations. The utility is also used to set, change, or remove one-time credentials and a storage unit ID for DD Boost. The gpdbrestore, gpdbrestore, and gpmfr utilities use the DD Boost credentials to access Data Domain systems. DD Boost information is stored in these files.

- **DDBOOST_CONFIG** is used by gpdbrestore and gpcrondump for backup and restore operations with the Data Domain system. The gpdbrestore utility creates or updates the file when you specify Data Domain information with the `--ddboost-host` option.
- **DDBOOST_MFR_CONFIG** is used by gpmfr for remote replication operations with the remote Data Domain system. The gpdbrestore utility creates or updates the file when you specify Data Domain information with the `--ddboost-host` option and `--ddboost-remote` option.

The configuration files are created in the current user (gpadmin) home directory on the Greenplum Database master and segment hosts. The path and file name cannot be changed. Information in the configuration files includes:

- Data Domain host name or IP address
- DD Boost user name
- DD Boost password
- Default Data Domain backup directory (DDBOOST_CONFIG only)
- Data Domain storage unit ID: default is GPDB (DDBOOST_CONFIG only)
- Data Domain default log level: default is WARNING
- Data Domain default log size: default is 50

Use the gpcrondump option --ddboost-show-config to display the current DD Boost configuration information from the Greenplum Database master configuration file. Specify the --remote option to display the configuration information for the remote Data Domain system.

About Data Domain Storage Units

When you use a Data Domain system to perform a backup, restore, or remote replication operation with the gpcrondump, gpdbrestore, or gpmfr utility, the operation uses a storage unit on a Data Domain system. You can specify the storage unit ID when you perform these operations:

- When you set the DD Boost credentials with the gpcrondump utility --ddboost-host option.

  If you specify the --ddboost-storage-unit option, the storage unit ID is written to the Greenplum Database DD Boost configuration file DDBOOST_CONFIG. If the storage unit ID is not specified, the default value is GPDB.

  If you specify the --ddboost-storage-unit option and the --ddboost-remote option to set DD Boost credentials for the remote Data Domain server, the storage ID information is ignored. The storage unit ID in the DDBOOST_CONFIG file is the default ID that is used for remote replication operations.

- When you perform a backup, restore, or remote replication operation with gpcrondump, gpdbrestore, or gpmfr.

  When you specify the --ddboost-storage-unit option, the utility uses the specified Data Domain storage unit for the operation. The value in the configuration file is not changed.

A Greenplum Database utility uses the storage unit ID based on this order of precedence from highest to lowest:

- Storage unit ID specified with --ddboost-storage-unit
- Storage unit ID specified in the configuration file
- Default storage unit ID GPDB

Greenplum Database master and segment instances use a single storage unit ID when performing a backup, restore, or remote replication operation.

**Important:** The storage unit ID in the Greenplum Database master and segment host configuration files must be the same. The Data Domain storage unit is created by the gpcrondump utility from the Greenplum Database master host.

The following occurs if storage unit IDs are different in the master and segment host configuration files:

- If all the storage units have not been created, the operation fails.
- If all the storage units have been created, a backup operation completes. However, the backup files are in different storage units and a restore operation fails because a full set of backup files is not in a single storage unit.

When performing a full backup operation (not an incremental backup), the storage unit is created on the Data Domain system if it does not exist.

A storage unit is not created if these gpcrondump options are specified: --incremental, --list-backup-file, --list-filter-tables, -o, or --ddboost-config-remove.

Greenplum Database replication operations use the same storage unit ID on both systems. For example, if you specify the --ddboost-storage-unit option for --replicate or --recover through gpmfr or --replicate from gpcrondump, the storage unit ID applies to both local and remote Data Domain systems.
When performing a replicate or recover operation with `gpmfr`, the storage unit on the destination Data Domain system (where the backup is being copied) is created if it does not exist.

**Configuring Data Domain Boost for Greenplum Database**

After you set up credentials for Data Domain Boost on the Greenplum Database, perform the following tasks in Data Domain to allow Data Domain Boost to work with Greenplum Database:

- **Configuring Distributed Segment Processing in Data Domain**
- **Configuring Advanced Load Balancing and Link Failover in Data Domain**
- **Export the Data Domain Path to Greenplum Database Hosts**

**Configuring Distributed Segment Processing in Data Domain**

Configure the distributed segment processing option on the Data Domain system. The configuration applies to all the Greenplum Database servers with the Data Domain Boost plug-in installed on them. This option is enabled by default, but verify that it is enabled before using Data Domain Boost backups:

```bash
# ddboost option show
```

To enable or disable distributed segment processing:

```bash
# ddboost option set distributed-segment-processing {enabled | disabled}
```

**Configuring Advanced Load Balancing and Link Failover in Data Domain**

If you have multiple network connections on a network subnet, you can create an interface group to provide load balancing and higher network throughput on your Data Domain system. When a Data Domain system on an interface group receives data from the media server clients, the data transfer is load balanced and distributed as separate jobs on the private network. You can achieve optimal throughput with multiple 10 GbE connections.

**Note:** To ensure that interface groups function properly, use interface groups only when using multiple network connections on the same networking subnet.

To create an interface group on the Data Domain system, create interfaces with the `net` command. If interfaces do not already exist, add the interfaces to the group, and register the Data Domain system with the backup application.

1. Add the interfaces to the group:

   ```bash
   # ddboost ifgroup add interface 192.0.2.1
   # ddboost ifgroup add interface 192.0.2.2
   # ddboost ifgroup add interface 192.0.2.3
   # ddboost ifgroup add interface 192.0.2.4
   ```

   **Note:** You can create only one interface group and this group cannot be named.

2. Select one interface on the Data Domain system to register with the backup application. Create a failover aggregated interface and register that interface with the backup application.

   **Note:** You do not have to register one of the `ifgroup` interfaces with the backup application. You can use an interface that is not part of the `ifgroup` to register with the backup application.

3. Enable `ddboost` on the Data Domain system:

   ```bash
   # ddboost ifgroup enable
   ```

4. Verify the Data Domain system configuration as follows:

   ```bash
   # ddboost ifgroup show config
   ```
Results similar to the following are displayed.

```
Interface
---------
192.0.2.1
192.0.2.2
192.0.2.3
192.0.2.4
---------
```

You can add or delete interfaces from the group at any time.

**Note:** Manage Advanced Load Balancing and Link Failover (an interface group) using the `ddboost ifgroup` command or from the **Enterprise Manager Data Management > DD Boost** view.

### Export the Data Domain Path to Greenplum Database Hosts

The commands and options in this topic apply to DDOS 5.0.x and 5.1.x. See the Data Domain documentation for details.

Use the following Data Domain commands to export the `/backup/ost` directory to a Greenplum Database host for Data Domain Boost backups.

```
# nfs add /backup/ost 192.0.2.0/24, 198.51.100.0/24 (insecure)
```

**Note:** The IP addresses refer to the Greenplum Database system working with the Data Domain Boost system.

### Create the Data Domain Login Credentials for the Greenplum Database Host

Create a username and password for the host to access the DD Boost Storage Unit (SU) at the time of backup and restore:

```
# user add user [password password] [priv {admin | security | user}]
```

### Backup Options for Data Domain Boost

Specify the `gpcrondump` options to match the setup.

Data Domain Boost backs up files to a storage unit in the Data Domain system. Status and report files remain on the local disk. If needed, specify the Data Domain system storage unit with the `--ddboost-storage-unit` option. This DD Boost command display the names of all storage units on a Data Domain system.

```
ddboost storage-unit show
```

To configure Data Domain Boost to remove old backup directories before starting a backup operation, specify a `gpcrondump backup expiration` option:

- The `-c` option clears all backup directories.
- The `-o` option clears the oldest backup directory.

To remove the oldest dump directory, specify `gpcrondump --ddboost` with the `-o` option. For example, if your retention period is 30 days, use `gpcrondump --ddboost` with the `-o` option on day 31.

Use `gpcrondump --ddboost` with the `-c` option to clear out all the old dump directories in `db_dumps`. The `-c` option deletes all dump directories that are at least one day old.
Using CRON to Schedule a Data Domain Boost Backup

1. Ensure the One-Time Data Domain Boost Credential Setup is complete.
2. Add the option \(-ddboost\) to the gpcrondump option:

   ```
gpcrondump -x mydatabase -z -v --ddboost
   ```

   If needed, specify the Data Domain system storage unit with the \(--ddboost-storage-unit\) option.

   **Important:** Do not use compression with Data Domain Boost backups. The \(-z\) option turns backup compression off.

   Some of the options available in gpcrondump have different implications when using Data Domain Boost. For details, see gpcrondump in the Greenplum Database Utility Reference.

Restoring From a Data Domain System with Data Domain Boost

1. Ensure the One-Time Data Domain Boost Credential Setup is complete.
2. Add the option \(--ddboost\) to the gpdbrestore command:

   ```
   $ gpdbrestore -t backup_timestamp -v -ddboost
   ```

   If needed, specify the Data Domain system storage unit with the \(--ddboost-storage-unit\) option.

   **Note:** Some of the gpdbrestore options available have different implications when using Data Domain. For details, see gpdbrestore in the Greenplum Database Utility Reference.

Backing Up Databases with Veritas NetBackup

For Greenplum Database on Red Hat Enterprise Linux, you can configure Greenplum Database to perform backup and restore operations with Veritas NetBackup. You configure Greenplum Database and NetBackup and then run a Greenplum Database gpcrondump or gpdbrestore command. The following topics describe how to set up NetBackup and back up or restore Greenplum Databases.

- About NetBackup Software
- Limitations
- Configuring Greenplum Database Hosts for NetBackup
- Configuring NetBackup for Greenplum Database
- Performing a Back Up or Restore with NetBackup
- Example NetBackup Back Up and Restore Commands

About NetBackup Software

NetBackup includes the following server and client software:

- The NetBackup master server manages NetBackup backups, archives, and restores. The master server is responsible for media and device selection for NetBackup.
- NetBackup Media servers are the NetBackup device hosts that provide additional storage by allowing NetBackup to use the storage devices that are attached to them.
- NetBackup client software that resides on the Greenplum Database hosts that contain data to be backed up.

See the Veritas NetBackup Getting Started Guide for information about NetBackup.

See the Release Notes for information about compatible NetBackup versions for this release of Pivotal Greenplum Database.
Limitations

- NetBackup is not compatible with DDBase. Both NetBackup and DDBase cannot be used in a single back up or restore operation.
- For incremental back up sets, a full backup and associated incremental backups, the backup set must be on a single device. For example, a backup set must all be on a NetBackup system. The backup set cannot have some backups on a NetBackup system and others on the local file system or a Data Domain system.

Configuring Greenplum Database Hosts for NetBackup

You install and configure NetBackup client software on the Greenplum Database master host and all segment hosts. The NetBackup client software must be able to communicate with the NetBackup server software.

1. Install the NetBackup client software on Greenplum Database hosts. See the NetBackup installation documentation for information on installing NetBackup clients on UNIX systems.

2. Set parameters in the NetBackup configuration file `/usr/openv/netbackup/bp.conf` on the Greenplum Database master and segment hosts. Set the following parameters on each Greenplum Database host:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVER</td>
<td>Host name of the NetBackup Master Server</td>
</tr>
<tr>
<td>MEDIA_SERVER</td>
<td>Host name of the NetBackup Media Server</td>
</tr>
<tr>
<td>CLIENT_NAME</td>
<td>Host name of the Greenplum Database Host</td>
</tr>
</tbody>
</table>

   See the Veritas NetBackup Administrator's Guide for information about the `bp.conf` file.

3. Set the `LD_LIBRARY_PATH` environment variable for Greenplum Database hosts to use NetBackup client. Greenplum Database installs NetBackup SDK library files that are used with the NetBackup client. To configure Greenplum Database to use the library files that correspond to the version of the NetBackup client that is installed on the hosts, add the following line to the file `$GPHOME/greenplum_path.sh`:

   ```bash
   LD_LIBRARY_PATH=$GPHOME/lib/nbuNN/lib:$LD_LIBRARY_PATH
   ``

   Replace the `NN` with the NetBackup client version that is installed on the host (for example, use 77 for NetBackup 7.7.x).

   The `LD_LIBRARY_PATH` line should be added before this line in `$GPHOME/greenplum_path.sh`:

   ```bash
   export LD_LIBRARY_PATH
   ```

4. Execute this command to remove the current `LD_LIBRARY_PATH` value:

   ```bash
   unset LD_LIBRARY_PATH
   ```

5. Execute this command to update the environment variables for Greenplum Database:

   ```bash
   source $GPHOME/greenplum_path.sh
   ```

   See the Veritas NetBackup Administrator's Guide for information about configuring NetBackup servers.

1. Ensure that the Greenplum Database hosts are listed as NetBackup clients for the NetBackup server.
In the NetBackup Administration Console, the information for the NetBackup clients, Media Server, and Master Server is in the **NetBackup Management** node within the **Host Properties** node.

2. Configure a NetBackup storage unit. The storage unit must be configured to point to a writable disk location.

   In the NetBackup Administration Console, the information for NetBackup storage units is in **NetBackup Management** node within the **Storage** node.

3. Configure a NetBackup backup policy and schedule within the policy.

   In the NetBackup Administration Console, the **Policy** node below the **Master Server** node is where you create a policy and a schedule for the policy.

   • In the **Policy Attributes** tab, these values are required for Greenplum Database:
     
     The value in the **Policy type** field must be DataStore
     
     The value in the **Policy storage** field is the storage unit created in the previous step.
     
     The value in **Limit jobs per policy** field must be at least 3.
     
   • In the **Policy Schedules** tab, create a NetBackup schedule for the policy.

### Configuring NetBackup for Greenplum Database

See the *Veritas NetBackup Administrator’s Guide* for information about configuring NetBackup servers.

1. Ensure that the Greenplum Database hosts are listed as NetBackup clients for the NetBackup server.

   In the NetBackup Administration Console, the information for the NetBackup clients, Media Server, and Master Server is in **NetBackup Management** node within the **Host Properties** node.

2. Configure a NetBackup storage unit. The storage unit must be configured to point to a writable disk location.

   In the NetBackup Administration Console, the information for NetBackup storage units is in **NetBackup Management** node within the **Storage** node.

3. Configure a NetBackup backup policy and schedule within the policy.

   In the NetBackup Administration Console, the **Policy** node below the **Master Server** node is where you create a policy and a schedule for the policy.

   • In the **Policy Attributes** tab, these values are required for Greenplum Database:
     
     The value in the **Policy type** field must be DataStore
     
     The value in the **Policy storage** field is the storage unit created in the previous step.
     
     The value in **Limit jobs per policy** field must be at least 3.
     
   • In the **Policy Schedules** tab, create a NetBackup schedule for the policy.

### Performing a Back Up or Restore with NetBackup

The Greenplum Database *gpcrondump* and *gpdbrestore* utilities support options to back up or restore data to a NetBackup storage unit. When performing a back up, Greenplum Database transfers data files directly to the NetBackup storage unit. No backup data files are created on the Greenplum Database hosts. The backup metadata files are both stored on the hosts and backed up to the NetBackup storage unit.

When performing a restore, the files are retrieved from the NetBackup server, and then restored.

Following are the *gpcrondump* utility options for NetBackup:

```
--netbackup-service-host  netbackup_master_server
--netbackup-policy  policy_name
--netbackup-schedule  schedule_name
--netbackup-block-size  size  (optional)
--netbackup-keyword  keyword  (optional)
```
The `gpdbrestore` utility provides the following options for NetBackup:

```
--netbackup-service-host netbackup_master_server
--netbackup-block-size size (optional)
```

**Note:** When performing a restore operation from NetBackup, you must specify the backup timestamp with the `gpdbrestore` utility `-t` option.

The policy name and schedule name are defined on the NetBackup master server. See *Configuring NetBackup for Greenplum Database* for information about policy name and schedule name. See the *Greenplum Database Utility Guide* for information about the Greenplum Database utilities.

**Note:** You must run the `gpcrondump` or `gpdbrestore` command during a time window defined for the NetBackup schedule.

During a backup or restore operation, a separate NetBackup job is created for the following types of Greenplum Database data:

- Segment data for each segment instance
- C database data
- Metadata
- Post data for the master
- State files (Global objects: `gpcrondump -G` option)
- Configuration files for master and segments (`gpcrondump -g` option)
- Report files (`gpcrondump -h` option)

In the NetBackup Administration Console, the Activity Monitor lists NetBackup jobs. For each job, the job detail displays Greenplum Database backup information.

**Note:** When backing up or restoring a large amount of data, set the NetBackup `CLIENT_READ_TIMEOUT` option to a value that is at least twice the expected duration of the operation (in seconds). The `CLIENT_READ_TIMEOUT` default value is 300 seconds (5 minutes).

For example, if a backup takes 3 hours, set the `CLIENT_READ_TIMEOUT` to 21600 (2 x 3 x 60 x 60). You can use the NetBackup `nbgetconfig` and `nbsetconfig` commands on the NetBackup server to view and change the option value.

For information about `CLIENT_READ_TIMEOUT` and the `nbgetconfig`, and `nbsetconfig` commands, see the NetBackup documentation.

### Example NetBackup Back Up and Restore Commands

This `gpcrondump` command backs up the database `customer` and specifies a NetBackup policy and schedule that are defined on the NetBackup master server `nbu_server1`. A block size of 1024 bytes is used to transfer data to the NetBackup server.

```
gpcrondump -x customer --netbackup-service-host=nbu_server1 \
   --netbackup-policy=gpdb_cust --netbackup-schedule=gpdb_backup \
   --netbackup-block-size=1024
```

This `gpdbrestore` command restores Greenplum Database data from the data managed by NetBackup master server `nbu_server1`. The option `-t 20170530090000` specifies the timestamp generated by `gpcrondump` when the backup was created. The `-e` option specifies that the target database is dropped before it is restored.

```
gpdbrestore -t 20170530090000 -e --netbackup-service-host=nbu_server1
```

### Restoring Greenplum Databases

How you restore a database from parallel backup files depends on how you answer the following questions.
1. Where are your backup files? If your backup files are on the segment hosts where gpcrondump created them, you can restore the database with gpdbrestore. If you moved your backup files away from the Greenplum array, for example to an archive server with gpcrondump, use gpdbrestore.

2. Are you recreating the Greenplum Database system, or just restoring your data? If Greenplum Database is running and you are restoring your data, use gpdbrestore. If you lost your entire array and need to rebuild the entire system from backup, use gpinitsystem.

3. Are you restoring to a system with the same number of segment instances as your backup set? If you are restoring to an array with the same number of segment hosts and segment instances per host, use gpdbrestore. If you are migrating to a different array configuration, you must do a non-parallel restore. See Restoring to a Different Greenplum System Configuration.

Restoring a Database Using gpdbrestore

The gpdbrestore utility restores a database from backup files created by gpcrondump.

The gpdbrestore requires one of the following options to identify the backup set to restore:

- `-t timestamp` – restore the backup with the specified timestamp.
- `-b YYYYMMDD` – restore dump files for the specified date in the db_dumps subdirectories on the segment data directories.
- `-s database_name` – restore the latest dump files for the specified database found in the segment data directories.
- `-R hostname:path` – restore the backup set located in the specified directory of a remote host.

To restore an incremental backup, you need a complete set of backup files—a full backup and any required incremental backups. You can use the `--list-backups` option to list the full and incremental backup sets required for an incremental backup specified by timestamp. For example:

```
$ gpdbrestore -t 20151013195916 --list-backup
```

You can restore a backup to a different database using the `--redirect database` option. The database is created if it does not exist. The following example restores the most recent backup of the mydb database to a new database named mydb_snapshot:

```
$ gpdbrestore -s grants --redirect grants_snapshot
```

You can also restore a backup to a different Greenplum Database system. See Restoring to a Different Greenplum System Configuration for information about this option.

To restore from an archive host using gpdbrestore

You can restore a backup saved on a host outside of the Greenplum cluster using the `-R` option. Although the Greenplum Database software does not have to be installed on the remote host, the remote host must have a gpadmin account configured with passwordless ssh access to all hosts in the Greenplum cluster. This is required because each segment host will use scp to copy its segment backup file from the archive host. See gpssh-exkeys in the Greenplum Database Utility Guide for help adding the remote host to the cluster.

This procedure assumes that the backup set was moved off the Greenplum array to another host in the network.

1. Ensure that the archive host is reachable from the Greenplum master host:

```
$ ping archive_host
```

2. Ensure that you can ssh into the remote host with the gpadmin account and no password.

```
$ ssh gpadmin@archive_host
```
3. Ensure that you can ping the master host from the archive host:

$ ping mdw

4. Ensure that the restore's target database exists. For example:

$ createdb database_name

5. From the master, run the `gpdbrestore` utility. The `-R` option specifies the host name and path to a complete backup set:

$ gpdbrestore -R archive_host:/gpdb/backups/archive/20120714 -e dbname

Omit the `-e dbname` option if the database has already been created.

### Restoring to a Different Greenplum System Configuration

To perform a parallel restore operation using `gpdbrestore`, the system you are restoring to must have the same configuration as the system that was backed up. To restore your database objects and data into a different system configuration, for example, to expand into a system with more segments, restore your parallel backup files by loading them through the Greenplum master. To perform a non-parallel restore, you must have:

- A full backup set created by a `gpcrondump` operation. The backup file of the master contains the DDL to recreate your database objects. The backup files of the segments contain the data.
- A running Greenplum Database system.
- The database you are restoring to exists in the system.

Segment dump files contain a `COPY` command for each table followed by the data in delimited text format. Collect all of the dump files for all of the segment instances and run them through the master to restore your data and redistribute it across the new system configuration.

#### To restore a database to a different system configuration

1. Ensure that you have a complete backup set, including dump files of the master (`gp_dump_-1_1_timestamp, gp_dump_-1_1_timestamp_post_data`) and one for each segment instance (for example, `gp_dump_0_2_timestamp, gp_dump_1_3_timestamp, gp_dump_2_4_timestamp`, and so on). Each dump file must have the same timestamp key. `gpcrondump` creates the dump files in each segment instance's data directory. You must collect all the dump files and move them to one location on the master host. You can copy each segment dump file to the master, load it, and then delete it after it loads successfully.

2. Ensure that the database you are restoring to is created in the system. For example:

   $ createdb database_name

3. Load the master dump file to restore the database objects. For example:

   $ psql database_name -f /gpdb/backups/gp_dump_-1_1_20160714

4. Load each segment dump file to restore the data. For example:

   $ psql database_name -f /gpdb/backups/gp_dump_0_2_20160714
   $ psql database_name -f /gpdb/backups/gp_dump_1_3_20160714
   $ psql database_name -f /gpdb/backups/gp_dump_2_4_20160714
   $ psql database_name -f /gpdb/backups/gp_dump_3_5_20160714
   ...

...
5. Load the post data file to restore database objects such as indexes, triggers, primary key constraints, etc.

   $ psql database_name -f /gpdb/backups/gp_dump_0_5_20160714_post_data

6. Update the database sequences based on the values from the original database.

   You can use the system utilities gunzip and egrep to extract the sequence value information from the original Greenplum Database master dump file gp_dump_-1_1_timestamp.gz into a text file. This command extracts the information into the file schema_path_and_seq_next_val.

   ```
   gunzip -c path_to_master_dump_directory/gp_dump_-1_1_timestamp.gz | egrep "SET search_path|SELECT pg_catalog.setval" > schema_path_and_seq_next_val
   ```

   This example command assumes the original Greenplum Database master dump file is in /data/gpdb/master/gpseg-1/db_dumps/20150112.

   ```
   gunzip -c /data/gpdb/master/gpseg-1/db_dumps/20150112/gp_dump_-1_1_20150112140316.gz | egrep "SET search_path|SELECT pg_catalog.setval" > schema_path_and_seq_next_val
   ```

   After extracting the information, use the Greenplum Database psql utility to update the sequences in the database. This example command updates the sequence information in the database test_restore:

   ```
   psql test_restore -f schema_path_and_seq_next_val
   ```

---

**Expanding a Greenplum System**

To scale up performance and storage capacity, expand your Greenplum system by adding hosts to the array.

Data warehouses typically grow over time as additional data is gathered and the retention periods increase for existing data. At times, it is necessary to increase database capacity to consolidate different data warehouses into a single database. Additional computing capacity (CPU) may also be needed to accommodate newly added analytics projects. Although it is wise to provide capacity for growth when a system is initially specified, it is not generally possible to invest in resources long before they are required. Therefore, you should expect to execute a database expansion project periodically.

Because of the Greenplum MPP architecture, when you add resources to the system, the capacity and performance are the same as if the system had been originally implemented with the added resources. Unlike data warehouse systems that require substantial downtime in order to dump and restore the data, expanding a Greenplum Database system is a phased process with minimal downtime. Regular and ad hoc workloads can continue while data is redistributed and transactional consistency is maintained. The administrator can schedule the distribution activity to fit into ongoing operations and can pause and resume as needed. Tables can be ranked so that datasets are redistributed in a prioritized sequence, either to ensure that critical workloads benefit from the expanded capacity sooner, or to free disk space needed to redistribute very large tables.

The expansion process uses standard Greenplum Database operations so it is transparent and easy for administrators to troubleshoot. Segment mirroring and any replication mechanisms in place remain active, so fault-tolerance is uncompromised and disaster recovery measures remain effective.

---

**System Expansion Overview**

Data warehouses typically grow over time, often at a continuous pace, as additional data is gathered and the retention period increases for existing data. At times, it is necessary to increase database capacity to consolidate disparate data warehouses into a single database. The data warehouse may also require
additional computing capacity (CPU) to accommodate added analytics projects. It is good to provide capacity for growth when a system is initially specified, but even if you anticipate high rates of growth, it is generally unwise to invest in capacity long before it is required. Database expansion, therefore, is a project that you should expect to have to execute periodically.

When you expand your database, you should expect the following qualities:

• Scalable capacity and performance. When you add resources to a Greenplum Database, the capacity and performance are the same as if the system had been originally implemented with the added resources.
• Uninterrupted service during expansion. Regular workloads, both scheduled and ad-hoc, are not interrupted. A short, scheduled downtime period is required to initialize the new servers, similar to downtime required to restart the system. The length of downtime is unrelated to the size of the system before or after expansion.
• Transactional consistency.
• Fault tolerance. During the expansion, standard fault-tolerance mechanisms—such as segment mirroring—remain active, consistent, and effective.
• Replication and disaster recovery. Any existing replication mechanisms continue to function during expansion. Restore mechanisms needed in case of a failure or catastrophic event remain effective.
• Transparency of process. The expansion process employs standard Greenplum Database mechanisms, so administrators can diagnose and troubleshoot any problems.
• Configurable process. Expansion can be a long running process, but it can be fit into a schedule of ongoing operations. The expansion schema’s tables allow administrators to prioritize the order in which tables are redistributed, and the expansion activity can be paused and resumed.

The planning and physical aspects of an expansion project are a greater share of the work than expanding the database itself. It will take a multi-discipline team to plan and execute the project. For on-premise installations, space must be acquired and prepared for the new servers. The servers must be specified, acquired, installed, cabled, configured, and tested. For cloud deployments, similar plans should also be made. Planning New Hardware Platforms describes general considerations for deploying new hardware.

After you provision the new hardware platforms and set up their networks, configure the operating systems and run performance tests using Greenplum utilities. The Greenplum Database software distribution includes utilities that are helpful to test and burn-in the new servers before beginning the software phase of the expansion. See Preparing and Adding Nodes for steps to prepare the new hosts for Greenplum Database.

Once the new servers are installed and tested, the software phase of the Greenplum Database expansion process begins. The software phase is designed to be minimally disruptive, transactionally consistent, reliable, and flexible.

• There is a brief period of downtime while the new segment hosts are initialized and the system is prepared for the expansion process. This downtime can be scheduled to occur during a period of low activity to avoid disrupting ongoing business operations. During the initialization process, the following tasks are performed:
  • Greenplum Database software is installed.
  • Databases and database objects are created on the new segment hosts.
  • An expansion schema is created in the master database to control the expansion process.
  • The distribution policy for each table is changed to DISTRIBUTED RANDOMLY.
• The system is restarted and applications resume.
  • New segments are immediately available and participate in new queries and data loads. The existing data, however, is skewed. It is concentrated on the original segments and must be redistributed across the new total number of primary segments.
  • Because tables now have a random distribution policy, the optimizer creates query plans that are not dependent on distribution keys. Some queries will be less efficient because more data motion operators are needed.
• Using the expansion control tables as a guide, tables and partitions are redistributed. For each table:
• An `ALTER TABLE` statement is issued to change the distribution policy back to the original policy. This causes an automatic data redistribution operation, which spreads data across all of the servers, old and new, according to the original distribution policy.
• The table’s status is updated in the expansion control tables.
• The query optimizer creates more efficient execution plans by including the distribution key in the planning.
• When all tables have been redistributed, the expansion is complete.

Redistributing data is a long-running process that creates a large volume of network and disk activity. It can take days to redistribute some very large databases. To minimize the effects of the increased activity on business operations, system administrators can pause and resume expansion activity on an ad hoc basis, or according to a predetermined schedule. Datasets can be prioritized so that critical applications benefit first from the expansion.

In a typical operation, you run the `gpexpand` utility four times with different options during the complete expansion process.

1. To create an expansion input file:

   ```bash
gpexpand -f hosts_file
   ```

2. To initialize segments and create the expansion schema:

   ```bash
gpexpand -i input_file -D database_name
   ```

   `gpexpand` creates a data directory, copies user tables from all existing databases on the new segments, and captures metadata for each table in an expansion schema for status tracking. After this process completes, the expansion operation is committed and irrevocable.

3. To redistribute tables:

   ```bash
gpexpand -d duration
   ```

   At initialization, `gpexpand` nullifies hash distribution policies on tables in all existing databases, except for parent tables of a partitioned table, and sets the distribution policy for all tables to random distribution.

   To complete system expansion, you must run `gpexpand` to redistribute data tables across the newly added segments. Depending on the size and scale of your system, redistribution can be accomplished in a single session during low-use hours, or you can divide the process into batches over an extended period. Each table or partition is unavailable for read or write operations during redistribution. As each table is redistributed across the new segments, database performance should incrementally improve until it exceeds pre-expansion performance levels.

   You may need to run `gpexpand` several times to complete the expansion in large-scale systems that require multiple redistribution sessions. `gpexpand` can benefit from explicit table redistribution ranking; see Planning Table Redistribution.

   Users can access Greenplum Database after initialization completes and the system is back online, but they may experience performance degradation on systems that rely heavily on hash distribution of tables. Normal operations such as ETL jobs, user queries, and reporting can continue, though users might experience slower response times.

   When a table has a random distribution policy, Greenplum Database cannot enforce unique constraints (such as `PRIMARY KEY`). This can affect your ETL and loading processes until table redistribution completes because duplicate rows do not issue a constraint violation error.

4. To remove the expansion schema:

   ```bash
gpexpand -c
   ```
For information about the `gpexpand` utility and the other utilities that are used for system expansion, see the *Greenplum Database Utility Guide*.

**Planning Greenplum System Expansion**

Careful planning will help to ensure a successful Greenplum expansion project.

The topics in this section help to ensure that you are prepared to execute a system expansion.

- *System Expansion Checklist* is a checklist you can use to prepare for and execute the system expansion process.
- *Planning New Hardware Platforms* covers planning for acquiring and setting up the new hardware.
- *Planning New Segment Initialization* provides information about planning to initialize new segment hosts with `gpexpand`.
- *Planning Table Redistribution* provides information about planning the data redistribution after the new segment hosts have been initialized.

**System Expansion Checklist**

This checklist summarizes the tasks for a Greenplum Database system expansion.

**Table 23: Greenplum Database System Expansion Checklist**

<table>
<thead>
<tr>
<th>Online Pre-Expansion Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>* System is up and available</td>
</tr>
<tr>
<td><strong>☐</strong> Devise and execute a plan for ordering, building, and networking new hardware platforms, or provisioning cloud resources.</td>
</tr>
<tr>
<td><strong>☐</strong> Devise a database expansion plan. Map the number of segments per host, schedule the downtime period for testing performance and creating the expansion schema, and schedule the intervals for table redistribution.</td>
</tr>
<tr>
<td><strong>☐</strong> Perform a complete schema dump.</td>
</tr>
<tr>
<td><strong>☐</strong> Install Greenplum Database binaries on new hosts.</td>
</tr>
<tr>
<td><strong>☐</strong> Copy SSH keys to the new hosts (<em>gpssh-exkeys</em>).</td>
</tr>
<tr>
<td><strong>☐</strong> Validate the operating system environment of the new hardware or cloud resources (<em>gpcheck</em>).</td>
</tr>
<tr>
<td><strong>☐</strong> Validate disk I/O and memory bandwidth of the new hardware or cloud resources (<em>gpcheckperf</em>).</td>
</tr>
<tr>
<td><strong>☐</strong> Validate that the master data directory has no extremely large files in the <code>pg_log</code> or <code>gpperfmon/data</code> directories.</td>
</tr>
<tr>
<td><strong>☐</strong> Validate that there are no catalog issues (<em>gpcheckcat</em>).</td>
</tr>
</tbody>
</table>
Prepare an expansion input file (`gpexpand`).

### Offline Expansion Tasks
- The system is locked and unavailable to all user activity during this process.

- Validate the operating system environment of the combined existing and new hardware or cloud resources (`gpcheck`).

- Validate disk I/O and memory bandwidth of the combined existing and new hardware or cloud resources (`gpcheckperf`).

- Initialize new segments into the array and create an expansion schema (`gpexpand -i input_file`).

### Online Expansion and Table Redistribution
- System is up and available

- Before you start table redistribution, stop any automated snapshot processes or other processes that consume disk space.

- Redistribute tables through the expanded system (`gpexpand`).

- Remove expansion schema (`gpexpand -c`).

- Run `analyze` to update distribution statistics.

During the expansion, use `gpexpand -a`, and post-expansion, use `analyze`.

### Planning New Hardware Platforms
A deliberate, thorough approach to deploy compatible hardware greatly minimizes risk to the expansion process.

Hardware resources and configurations for new segment hosts should match those of the existing hosts. Work with *Greenplum Platform Engineering* before making a hardware purchase to expand Greenplum Database.

The steps to plan and set up new hardware platforms vary for each deployment. Some considerations include how to:

- Prepare the physical space for the new hardware; consider cooling, power supply, and other physical factors.
- Determine the physical networking and cabling required to connect the new and existing hardware.
- Map the existing IP address spaces and developing a networking plan for the expanded system.
- Capture the system configuration (users, profiles, NICs, and so on) from existing hardware to use as a detailed list for ordering new hardware.
- Create a custom build plan for deploying hardware with the desired configuration in the particular site and environment.

After selecting and adding new hardware to your network environment, ensure you perform the burn-in tasks described in *Verifying OS Settings*. 

---

184
**Planning New Segment Initialization**

Expanding Greenplum Database requires a limited period of system downtime. During this period, run `gpexpand` to initialize new segments into the array and create an expansion schema.

The time required depends on the number of schema objects in the Greenplum system and other factors related to hardware performance. In most environments, the initialization of new segments requires less than thirty minutes offline.

**Note:** After you begin initializing new segments, you can no longer restore the system using backup files created for the pre-expansion system. When initialization successfully completes, the expansion is committed and cannot be rolled back.

**Planning Mirror Segments**

If your existing array has mirror segments, the new segments must have mirroring configured. If there are no mirrors configured for existing segments, you cannot add mirrors to new hosts with the `gpexpand` utility.

For Greenplum Database arrays with mirror segments, ensure you add enough new host machines to accommodate new mirror segments. The number of new hosts required depends on your mirroring strategy:

- **Spread Mirroring** — Add at least one more host to the array than the number of segments per host. The number of separate hosts must be greater than the number of segment instances per host to ensure even spreading.
- **Grouped Mirroring** — Add at least two new hosts so the mirrors for the first host can reside on the second host, and the mirrors for the second host can reside on the first. For more information, see About Segment Mirroring.

**Increasing Segments Per Host**

By default, new hosts are initialized with as many primary segments as existing hosts have. You can increase the segments per host or add new segments to existing hosts.

For example, if existing hosts currently have two segments per host, you can use `gpexpand` to initialize two additional segments on existing hosts for a total of four segments and four new segments on new hosts.

The interactive process for creating an expansion input file prompts for this option; you can also specify new segment directories manually in the input configuration file. For more information, see Creating an Input File for System Expansion.

**About the Expansion Schema**

At initialization, `gpexpand` creates an expansion schema. If you do not specify a database at initialization (`gpexpand -D`), the schema is created in the database indicated by the `PGDATABASE` environment variable.

The expansion schema stores metadata for each table in the system so its status can be tracked throughout the expansion process. The expansion schema consists of two tables and a view for tracking expansion operation progress:

- `gpexpand.status`
- `gpexpand.status_detail`
- `gpexpand.expansion_progress`

Control expansion process aspects by modifying `gpexpand.status_detail`. For example, removing a record from this table prevents the system from expanding the table across new segments. Control the order in which tables are processed for redistribution by updating the `rank` value for a record. For more information, see Ranking Tables for Redistribution.
Planning Table Redistribution

Table redistribution is performed while the system is online. For many Greenplum systems, table redistribution completes in a single `gpexpand` session scheduled during a low-use period. Larger systems may require multiple sessions and setting the order of table redistribution to minimize performance impact. Complete the table redistribution in one session if possible.

**Important:** To perform table redistribution, your segment hosts must have enough disk space to temporarily hold a copy of your largest table. All tables are unavailable for read and write operations during redistribution.

The performance impact of table redistribution depends on the size, storage type, and partitioning design of a table. For any given table, redistributing it with `gpexpand` takes as much time as a `CREATE TABLE AS SELECT` operation would. When redistributing a terabyte-scale fact table, the expansion utility can use much of the available system resources, which could affect query performance or other database workloads.

Managing Redistribution in Large-Scale Greenplum Systems

You can manage the order in which tables are redistributed by adjusting their ranking. See **Ranking Tables for Redistribution**. Manipulating the redistribution order can help adjust for limited disk space and restore optimal query performance for high-priority queries sooner.

When planning the redistribution phase, consider the impact of the exclusive lock taken on each table during redistribution. User activity on a table can delay its redistribution, but also tables are unavailable for user activity during redistribution.

Systems with Abundant Free Disk Space

In systems with abundant free disk space (required to store a copy of the largest table), you can focus on restoring optimum query performance as soon as possible by first redistributing important tables that queries use heavily. Assign high ranking to these tables, and schedule redistribution operations for times of low system usage. Run one redistribution process at a time until large or critical tables have been redistributed.

Systems with Limited Free Disk Space

If your existing hosts have limited disk space, you may prefer to first redistribute smaller tables (such as dimension tables) to clear space to store a copy of the largest table. Available disk space on the original segments increases as each table is redistributed across the expanded array. When enough free space exists on all segments to store a copy of the largest table, you can redistribute large or critical tables. Redistribution of large tables requires exclusive locks; schedule this procedure for off-peak hours.

Also consider the following:

- Run multiple parallel redistribution processes during off-peak hours to maximize available system resources.
- When running multiple processes, operate within the connection limits for your Greenplum system. For information about limiting concurrent connections, see **Limiting Concurrent Connections**.

Redistributing Append-Optimized and Compressed Tables

`gpexpand` redistributes append-optimized and compressed append-optimized tables at different rates than heap tables. The CPU capacity required to compress and decompress data tends to increase the impact on system performance. For similar-sized tables with similar data, you may find overall performance differences like the following:

- Uncompressed append-optimized tables expand 10% faster than heap tables.
• zlib-compressed append-optimized tables expand at a significantly slower rate than uncompressed append-optimized tables, potentially up to 80% slower.
• Systems with data compression such as ZFS/LZJB take longer to redistribute.

Important: If your system nodes use data compression, use identical compression on new nodes to avoid disk space shortage.

Redistributing Tables with Primary Key Constraints
There is a time period during which primary key constraints cannot be enforced between the initialization of new segments and successful table redistribution. Duplicate data inserted into tables during this period prevents the expansion utility from redistributing the affected tables.

After a table is redistributed, the primary key constraint is properly enforced again. If an expansion process violates constraints, the expansion utility logs errors and displays warnings when it completes. To fix constraint violations, perform one of the following remedies:

• Clean up duplicate data in the primary key columns, and re-run gpexpand.
• Drop the primary key constraints, and re-run gpexpand.

Redistributing Tables with User-Defined Data Types
You cannot perform redistribution with the expansion utility on tables with dropped columns of user-defined data types. To redistribute tables with dropped columns of user-defined types, first re-create the table using CREATE TABLE AS SELECT. After this process removes the dropped columns, redistribute the table with gpexpand.

Redistributing Partitioned Tables
Because the expansion utility can process each individual partition on a large table, an efficient partition design reduces the performance impact of table redistribution. Only the child tables of a partitioned table are set to a random distribution policy. The read/write lock for redistribution applies to only one child table at a time.

Redistributing Indexed Tables
Because the gpexpand utility must re-index each indexed table after redistribution, a high level of indexing has a large performance impact. Systems with intensive indexing have significantly slower rates of table redistribution.

Preparing and Adding Nodes
Verify your new nodes are ready for integration into the existing Greenplum system.

To prepare new system nodes for expansion, install the Greenplum Database software binaries, exchange the required SSH keys, and run performance tests.

Run performance tests first on the new nodes and then all nodes. Run the tests on all nodes with the system offline so user activity does not distort results.

Generally, you should run performance tests when an administrator modifies node networking or other special conditions in the system. For example, if you will run the expanded system on two network clusters, run tests on each cluster.

Adding New Nodes to the Trusted Host Environment
New nodes must exchange SSH keys with the existing nodes to enable Greenplum administrative utilities to connect to all segments without a password prompt. Perform the key exchange process twice.

First perform the process as root, for administration convenience, and then as the user gpadmin, for management utilities. Perform the following tasks in order:
1. To exchange SSH keys as root
2. To create the gpadmin user
3. To exchange SSH keys as the gpadmin user

Note: The Greenplum Database segment host naming convention is sdwN where sdw is a prefix and N is an integer (sdw1, sdw2 and so on). For hosts with multiple interfaces, the convention is to append a dash (-) and number to the host name. For example, sdw1-1 and sdw1-2 are the two interface names for host sdw1.

To exchange SSH keys as root
1. Create a host file with the existing host names in your array and a separate host file with the new expansion host names. For existing hosts, you can use the same host file used to set up SSH keys in the system. In the files, list all hosts (master, backup master, and segment hosts) with one name per line and no extra lines or spaces. Exchange SSH keys using the configured host names for a given host if you use a multi-NIC configuration. In this example, mdw is configured with a single NIC, and sdw1, sdw2, and sdw3 are configured with 4 NICs:

```
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. Log in as root on the master host, and source the greenplum_path.sh file from your Greenplum installation.

```
$ su -
# source /usr/local/greenplum-db/greenplum_path.sh
```

3. Run the gpssh-exkeys utility referencing the host list files. For example:

```
# gpssh-exkeys -e /home/gpadmin/existing_hosts_file -x /home/gpadmin/new_hosts_file
```

4. gpssh-exkeys checks the remote hosts and performs the key exchange between all hosts. Enter the root user password when prompted. For example:

```
***Enter password for root@hostname: <root_password>
```

To create the gpadmin user
1. Use gpssh to create the gpadmin user on all the new segment hosts (if it does not exist already). Use the list of new hosts you created for the key exchange. For example:

```
# gpssh -f new_hosts_file '/usr/sbin/useradd gpadmin -d /home/gpadmin -s /bin/bash'
```

2. Set a password for the new gpadmin user. On Linux, you can do this on all segment hosts simultaneously using gpssh. For example:

```
# gpssh -f new_hosts_file 'echo gpadmin_password | passwd'
```
3. Verify the `gpadmin` user has been created by looking for its home directory:

```
# gpssh -f new_hosts_file ls -l /home
```

**To exchange SSH keys as the `gpadmin` user**

1. Log in as `gpadmin` and run the `gpssh-exkeys` utility referencing the host list files. For example:

```
# gpssh-exkeys -e /home/gpadmin/existing_hosts_file -x
/home/gpadmin/new_hosts_file
```

2. `gpssh-exkeys` will check the remote hosts and perform the key exchange between all hosts. Enter the `gpadmin` user password when prompted. For example:

```
***Enter password for gpadmin@hostname: <gpadmin_password>
```

**Verifying OS Settings**

Use the `gpcheck` utility to verify all new hosts in your array have the correct OS settings to run Greenplum Database software.

**To run gpcheck**

1. Log in on the master host as the user who will run your Greenplum Database system (for example, `gpadmin`).

```
$ su - gpadmin
```

2. Run the `gpcheck` utility using your host file for new hosts. For example:

```
$ gpcheck -f new_hosts_file
```

**Validating Disk I/O and Memory Bandwidth**

Use the `gpcheckperf` utility to test disk I/O and memory bandwidth.

**To run gpcheckperf**

1. Run the `gpcheckperf` utility using the host file for new hosts. Use the `-d` option to specify the file systems you want to test on each host. You must have write access to these directories. For example:

```
$ gpcheckperf -f new_hosts_file -d /data1 -d /data2 -v
```

2. The utility may take a long time to perform the tests because it is copying very large files between the hosts. When it is finished, you will see the summary results for the Disk Write, Disk Read, and Stream tests.

For a network divided into subnets, repeat this procedure with a separate host file for each subnet.

**Integrating New Hardware into the System**

Before initializing the system with the new segments, shut down the system with `gpstop` to prevent user activity from skewing performance test results. Then, repeat the performance tests using host files that include all nodes, existing and new:

- **Verifying OS Settings**
- **Validating Disk I/O and Memory Bandwidth**
Initializing New Segments

Use the `gpexpand` utility to initialize the new segments, create the expansion schema, and set a system-wide random distribution policy for the database.

The first time you run `gpexpand` with a valid input file it creates the expansion schema and sets the distribution policy for all tables to `DISTRIBUTED RANDOMLY`. After these steps are completed, running `gpexpand` detects if the expansion schema has been created and, if so, performs table redistribution.

- Creating an Input File for System Expansion
- Running `gpexpand` to Initialize New Segments
- Rolling Back a Failed Expansion Setup

Creating an Input File for System Expansion

To begin expansion, `gpexpand` requires an input file containing information about the new segments and hosts. If you run `gpexpand` without specifying an input file, the utility displays an interactive interview that collects the required information and automatically creates an input file.

If you create the input file using the interactive interview, you may specify a file with a list of expansion hosts in the interview prompt. If your platform or command shell limits the length of the host list, specifying the hosts with `-f` may be mandatory.

Creating an input file in Interactive Mode

Before you run `gpexpand` to create an input file in interactive mode, ensure you know:

- The number of new hosts (or a hosts file)
- The new hostnames (or a hosts file)
- The mirroring strategy used in existing hosts, if any
- The number of segments to add per host, if any

The utility automatically generates an input file based on this information, `dbid`, `content ID`, and data directory values stored in `gp_segment_configuration`, and saves the file in the current directory.

To create an input file in interactive mode

1. Log in on the master host as the user who will run your Greenplum Database system; for example, `gpadmin`.
2. Run `gpexpand`. The utility displays messages about how to prepare for an expansion operation, and it prompts you to quit or continue.
   
   Optionally, specify a hosts file using `-f`. For example:

   ```
   $ gpexpand -f /home/gpadmin/new_hosts_file
   ```

3. At the prompt, select `Y` to continue.
4. Unless you specified a hosts file using `-f`, you are prompted to enter hostnames. Enter a comma separated list of the hostnames of the new expansion hosts. Do not include interface hostnames. For example:

   ```
   > sdw4, sdw5, sdw6, sdw7
   ```

   To add segments to existing hosts only, enter a blank line at this prompt. Do not specify `localhost` or any existing host name.

5. Enter the mirroring strategy used in your system, if any. Options are `spread|grouped|none`. The default setting is `grouped`.

   Ensure you have enough hosts for the selected grouping strategy. For more information about mirroring, see `Planning Mirror Segments`. 
6. Enter the number of new primary segments to add, if any. By default, new hosts are initialized with the same number of primary segments as existing hosts. Increase segments per host by entering a number greater than zero. The number you enter will be the number of additional segments initialized on all hosts. For example, if existing hosts currently have two segments each, entering a value of 2 initializes two more segments on existing hosts, and four segments on new hosts.

7. If you are adding new primary segments, enter the new primary data directory root for the new segments. Do not specify the actual data directory name, which is created automatically by `gpexpand` based on the existing data directory names.

   For example, if your existing data directories are as follows:

   `/gpdata/primary/gp0`
   `/gpdata/primary/gp1`

   then enter the following (one at each prompt) to specify the data directories for two new primary segments:

   `/gpdata/primary`
   `/gpdata/primary`

   When the initialization runs, the utility creates the new directories `gp2` and `gp3` under `/gpdata/primary`.

8. If you are adding new mirror segments, enter the new mirror data directory root for the new segments. Do not specify the data directory name; it is created automatically by `gpexpand` based on the existing data directory names.

   For example, if your existing data directories are as follows:

   `/gpdata/mirror/gp0`
   `/gpdata/mirror/gp1`

   enter the following (one at each prompt) to specify the data directories for two new mirror segments:

   `/gpdata/mirror`
   `/gpdata/mirror`

   When the initialization runs, the utility will create the new directories `gp2` and `gp3` under `/gpdata/mirror`.

These primary and mirror root directories for new segments must exist on the hosts, and the user running `gpexpand` must have permissions to create directories in them.

After you have entered all required information, the utility generates an input file and saves it in the current directory. For example:

```
gpexpand_inputfile_yyyymmdd_145134
```

### Expansion Input File Format

Use the interactive interview process to create your own input file unless your expansion scenario has atypical needs.

The format for expansion input files is:

```
hostname:address:port:fselocation:dbid:content:preferred_role:replication_port
```

For example:

```
sdw5:sdw5-1:50011:/gpdata/primary/gp9:11:9:p:53011
sdw5:sdw5-2:50012:/gpdata/primary/gp10:12:10:p:53011
```
For each new segment, this format of expansion input file requires the following:

**Table 24: Data for the expansion configuration file**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Valid Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostname</td>
<td>Hostname</td>
<td>Hostname for the segment host.</td>
</tr>
<tr>
<td>port</td>
<td>An available port number</td>
<td>Database listener port for the segment, incremented on the existing segment port base number.</td>
</tr>
<tr>
<td>fselocation</td>
<td>Directory name</td>
<td>The data directory (filespace) location for a segment as per the pg_filespace_entry system catalog.</td>
</tr>
<tr>
<td>dbid</td>
<td>Integer. Must not conflict with existing dbid values.</td>
<td>Database ID for the segment. The values you enter should be incremented sequentially from existing dbid values shown in the system catalog gp_segment_configuration. For example, to add four nodes to an existing ten-segment array with dbid values of 1-10, list new dbid values of 11, 12, 13 and 14.</td>
</tr>
<tr>
<td>content</td>
<td>Integer. Must not conflict with existing content values.</td>
<td>The content ID of the segment. A primary segment and its mirror should have the same content ID, incremented sequentially from existing values. For more information, see content in the reference for gp_segment_configuration.</td>
</tr>
<tr>
<td>preferred_role</td>
<td>p</td>
<td>m</td>
</tr>
<tr>
<td>replication_port</td>
<td>An available port number</td>
<td>File replication port for the segment, incremented on the existing segment replication_port base number.</td>
</tr>
</tbody>
</table>

**Running gpexpand to Initialize New Segments**

After you have created an input file, run `gpexpand` to initialize new segments. The utility automatically stops Greenplum Database segment initialization and restarts the system when the process finishes.

**To run gpexpand with an input file**

1. Log in on the master host as the user who will run your Greenplum Database system; for example, `gpadmin`.
2. Run the `gpexpand` utility, specifying the input file with `-i`. Optionally, use `-D` to specify the database in which to create the expansion schema. For example:

```
$ gpexpand -i input_file -D database1
```

The utility detects if an expansion schema exists for the Greenplum Database system. If a schema exists, remove it with `gpexpand -c` before you start a new expansion operation. See `Removing the Expansion Schema`.

When the new segments are initialized and the expansion schema is created, the utility prints a success message and exits.

When the initialization process completes, you can connect to Greenplum Database and view the expansion schema. The schema resides in the database you specified with `-D` or in the database specified by the `PGDATABASE` environment variable. For more information, see `About the Expansion Schema`.

**Rolling Back a Failed Expansion Setup**

You can roll back an expansion setup operation only if the operation fails.

If the expansion fails during the initialization step, while the database is down, you must first restart the database in master-only mode by running the `gpstart -m` command.

Roll back the failed expansion with the following command, specifying the database that contains the expansion schema:

```
gpexpand --rollback -D database_name
```

**Redistributing Tables**

Redistribute tables to balance existing data over the newly expanded cluster.

After creating an expansion schema, you can bring Greenplum Database back online and redistribute tables across the entire array with `gpexpand`. Aim to run this during low-use hours when the utility's CPU usage and table locks have minimal impact on operations. Rank tables to redistribute the largest or most critical tables first.

**Note:** When redistributing data, Greenplum Database must be running in production mode. Greenplum Database cannot be restricted mode or in master mode. The `gpstart` options `-R` or `-m` cannot be specified to start Greenplum Database.

While table redistribution is underway, any new tables or partitions created are distributed across all segments exactly as they would be under normal operating conditions. Queries can access all segments, even before the relevant data is redistributed to tables on the new segments. The table or partition being redistributed is locked and unavailable for read or write operations. When its redistribution completes, normal operations resume.

- Ranking Tables for Redistribution
- Redistributing Tables Using gpexpand
- Monitoring Table Redistribution

**Ranking Tables for Redistribution**

For large systems, you can control the table redistribution order. Adjust tables' `rank` values in the expansion schema to prioritize heavily-used tables and minimize performance impact. Available free disk space can affect table ranking; see `Managing Redistribution in Large-Scale Greenplum Systems`. 
To rank tables for redistribution by updating rank values in `gpexpand.status_detail`, connect to Greenplum Database using `psql` or another supported client. Update `gpexpand.status_detail` with commands such as:

```sql
=> UPDATE gpexpand.status_detail SET rank=10;
=> UPDATE gpexpand.status_detail SET rank=1 WHERE fq_name = 'public.lineitem';
=> UPDATE gpexpand.status_detail SET rank=2 WHERE fq_name = 'public.orders';
```

These commands lower the priority of all tables to 10 and then assign a rank of 1 to `lineitem` and a rank of 2 to `orders`. When table redistribution begins, `lineitem` is redistributed first, followed by `orders` and all other tables in `gpexpand.status_detail`. To exclude a table from redistribution, remove the table from `gpexpand.status_detail`.

**Redistributing Tables Using gpexpand**

**To redistribute tables with gpexpand**

1. Log in on the master host as the user who will run your Greenplum Database system, for example, `gpadmin`.
2. Run the `gpexpand` utility. You can use the `-d` or `-e` option to define the expansion session time period. For example, to run the utility for up to 60 consecutive hours:

```
$ gpexpand -d 60:00:00
```

The utility redistributes tables until the last table in the schema completes or it reaches the specified duration or end time. `gpexpand` updates the status and time in `gpexpand.status` when a session starts and finishes.

**Monitoring Table Redistribution**

You can query the expansion schema during the table redistribution process. The view `gpexpand.expansion_progress` provides a current progress summary, including the estimated rate of table redistribution and estimated time to completion. You can query the table `gpexpand.status_detail` for per-table status information.

**Viewing Expansion Status**

After the first table completes redistribution, `gpexpand.expansion_progress` calculates its estimates and refreshes them based on all tables’ redistribution rates. Calculations restart each time you start a table redistribution session with `gpexpand`. To monitor progress, connect to Greenplum Database using `psql` or another supported client; query `gpexpand.expansion_progress` with a command like the following:

```sql
=# SELECT * FROM gpexpand.expansion_progress;

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytes Left</td>
<td>5534842880</td>
</tr>
<tr>
<td>Bytes Done</td>
<td>142475264</td>
</tr>
<tr>
<td>Estimated Expansion Rate</td>
<td>680.75667095996092 MB/s</td>
</tr>
<tr>
<td>Estimated Time to Completion</td>
<td>00:01:01.008047</td>
</tr>
<tr>
<td>Tables Expanded</td>
<td>4</td>
</tr>
<tr>
<td>Tables Left</td>
<td>4</td>
</tr>
</tbody>
</table>
```

(6 rows)
**Viewing Table Status**

The table `gpexpand.status_detail` stores status, time of last update, and more facts about each table in the schema. To see a table's status, connect to Greenplum Database using `psql` or another supported client and query `gpexpand.status_detail`:

```sql
=> SELECT status, expansion_started, source_bytes FROM gpexpand.status_detail WHERE fq_name = 'public.sales';

<table>
<thead>
<tr>
<th>status</th>
<th>expansion_started</th>
<th>source_bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPLETED</td>
<td>2017-02-20 10:54:10.043869</td>
<td>4929748992</td>
</tr>
</tbody>
</table>
```

(1 row)

**Removing the Expansion Schema**

To clean up after expanding the Greenplum cluster, remove the expansion schema.

You can safely remove the expansion schema after the expansion operation is complete and verified. To run another expansion operation on a Greenplum system, first remove the existing expansion schema.

**To remove the expansion schema**

1. Log in on the master host as the user who will be running your Greenplum Database system (for example, `gpadmin`).
2. Run the `gpexpand` utility with the `-c` option. For example:

   ```bash
   $ gpexpand -c
   $```

   **Note:** Some systems require you to press Enter twice.

**Migrating Data with gptransfer**

This topic describes how to use the `gptransfer` utility to transfer data between databases.

The `gptransfer` migration utility transfers Greenplum Database metadata and data from one Greenplum database to another Greenplum database, allowing you to migrate the entire contents of a database, or just selected tables, to another database. The source and destination databases may be in the same or a different cluster. Data is transferred in parallel across all the segments, using the `gpfdist` data loading utility to attain the highest transfer rates.

`gptransfer` handles the setup and execution of the data transfer. Participating clusters must already exist, have network access between all hosts in both clusters, and have certificate-authenticated ssh access between all hosts in both clusters.

The interface includes options to transfer one or more full databases, or one or more database tables. A full database transfer includes the database schema, table data, indexes, views, roles, user-defined functions, and resource queues. Configuration files, including `postgres.conf` and `pg_hba.conf`, must be transferred manually by an administrator. Extensions installed in the database with `gppkg`, such as MADlib, must be installed in the destination database by an administrator.

See the [Greenplum Database Utility Guide](#) for complete syntax and usage information for the `gptransfer` utility.

**Prerequisites**

- The `gptransfer` utility can only be used with Greenplum Database. Apache HAWQ is not supported as a source or destination.
- The source and destination Greenplum clusters must both be version 4.2 or higher.
• At least one Greenplum instance must include the gptransfer utility in its distribution. If neither the source or destination includes gptransfer, you must upgrade one of the clusters to use gptransfer.
• The gptransfer utility can be run from the cluster with the source or destination database.
• The number of segments in the destination cluster must be greater than or equal to the number of hosts in the source cluster. The number of segments in the destination may be smaller than the number of segments in the source, but the data will transfer at a slower rate.
• The segment hosts in both clusters must have network connectivity with each other.
• Every host in both clusters must be able to connect to every other host with certificate-authenticated SSH. You can use the gpssh_exkeys utility to exchange public keys between the hosts of both clusters.

What gptransfer Does

gptransfer uses writable and readable external tables, the Greenplum gpfdist parallel data-loading utility, and named pipes to transfer data from the source database to the destination database. Segments on the source cluster select from the source database table and insert into a writable external table. Segments in the destination cluster select from a readable external table and insert into the destination database table. The writable and readable external tables are backed by named pipes on the source cluster's segment hosts, and each named pipe has a gpfdist process serving the pipe's output to the readable external table on the destination segments.

gptransfer orchestrates the process by processing the database objects to be transferred in batches. For each table to be transferred, it performs the following tasks:
• creates a writable external table in the source database
• creates a readable external table in the destination database
• creates named pipes and gpfdist processes on segment hosts in the source cluster
• executes a SELECT INTO statement in the source database to insert the source data into the writable external table
• executes a SELECT INTO statement in the destination database to insert the data from the readable external table into the destination table
• optionally validates the data by comparing row counts or MD5 hashes of the rows in the source and destination
• cleans up the external tables, named pipes, and gpfdist processes

Fast Mode and Slow Mode

gptransfer sets up data transfer using the gpfdist parallel file serving utility, which serves the data evenly to the destination segments. Running more gpfdist processes increases the parallelism and the data transfer rate. When the destination cluster has the same or a greater number of segments than the source cluster, gptransfer sets up one named pipe and one gpfdist process for each source segment. This is the configuration for optimal data transfer rates and is called fast mode. The following figure illustrates a setup on a segment host when the destination cluster has at least as many segments as the source cluster.
The configuration of the input end of the named pipes differs when there are fewer segments in the destination cluster than in the source cluster. `gptransfer` handles this alternative setup automatically. The difference in configuration means that transferring data into a destination cluster with fewer segments than the source cluster is not as fast as transferring into a destination cluster of the same or greater size. It is called slow mode because there are fewer `gpfdist` processes serving the data to the destination cluster, although the transfer is still quite fast with one `gpfdist` per segment host.

When the destination cluster is smaller than the source cluster, there is one named pipe per segment host and all segments on the host send their data through it. The segments on the source host write their data to a writable external web table connected to a `gpfdist` process on the input end of the named pipe. This consolidates the table data into a single named pipe. A `gpfdist` process on the output of the named pipe serves the consolidated data to the destination cluster. The following figure illustrates this configuration.
On the destination side, \texttt{gptransfer} defines a readable external table with the \texttt{gpfdist} server on the source host as input and selects from the readable external table into the destination table. The data is distributed evenly to all the segments in the destination cluster.

### Batch Size and Sub-batch Size

The degree of parallelism of a \texttt{gptransfer} execution is determined by two command-line options: \texttt{--batch-size} and \texttt{--sub-batch-size}. The \texttt{--batch-size} option specifies the number of tables to transfer in a batch. The default batch size is 2, which means that two table transfers are in process at any time. The minimum batch size is 1 and the maximum is 10. The \texttt{--sub-batch-size} parameter specifies the maximum number of parallel sub-processes to start to do the work of transferring a table. The default is 25 and the maximum is 50. The product of the batch size and sub-batch size is the amount of parallelism. If set to the defaults, for example, \texttt{gptransfer} can perform 50 concurrent tasks. Each thread is a Python process and consumes memory, so setting these values too high can cause a Python Out of Memory error. For this reason, the batch sizes should be tuned for your environment.

### Preparing Hosts for \texttt{gptransfer}

When you install a Greenplum Database cluster, you set up all the master and segment hosts so that the Greenplum Database administrative user (gpadmin) can connect with SSH from every host in the cluster to any other host in the cluster without providing a password. The \texttt{gptransfer} utility requires this capability between every host in the source and destination clusters. First, ensure that the clusters have network connectivity with each other. Then, prepare a hosts file containing a list of all the hosts in both clusters, and use the \texttt{gpssh-exkeys} utility to exchange keys. See the reference for \texttt{gpssh-exkeys} in the Greenplum Database Utility Guide.

The host map file is a text file that lists the segment hosts in the source cluster. It is used to enable communication between the hosts in Greenplum clusters. The file is specified on the \texttt{gptransfer} command line with the \texttt{--source-map-file=host_map_file} command option. It is a required option when using \texttt{gptransfer} to copy data between two separate Greenplum clusters.
The file contains a list in the following format:

```
host1_name,host1_ip_addr
host2_name,host2_ipaddr
...
```

The file uses IP addresses instead of host names to avoid any problems with name resolution between the clusters.

**Limitations**

gptransfer transfers data from user databases only; the postgres, template0, and template1 databases cannot be transferred. Administrators must transfer configuration files manually and install extensions into the destination database with gppkg.

The destination cluster must have at least as many segments as the source cluster has segment hosts. Transferring data to a smaller cluster is not as fast as transferring data to a larger cluster.

Transferring small or empty tables can be unexpectedly slow. There is significant fixed overhead in setting up external tables and communications processes for parallel data loading between segments that occurs whether or not there is actual data to transfer. It can be more efficient to transfer the schema and smaller tables to the destination database using other methods, then use gptransfer with the -t option to transfer large tables.

**Full Mode and Table Mode**

When run with the --full option, gptransfer copies all user-created databases, tables, views, indexes, roles, user-defined functions, and resource queues in the source cluster to the destination cluster. The destination system cannot contain any user-defined databases, only the default databases postgres, template0, and template1. If gptransfer finds a database on the destination it fails with a message like the following:

```
[ERROR]:- gptransfer: error: --full option specified but tables exist on destination system
```

**Note:** The --full option cannot be specified with the -t, -d, -f, or --partition-transfer options.

To copy tables individually, specify the tables using either the -t command-line option (one option per table) or by using the -f command-line option to specify a file containing a list of tables to transfer. Tables are specified in the fully-qualified format `database.schema.table`. The table definition, indexes, and table data are copied. The database must already exist on the destination cluster.

By default, gptransfer fails if you attempt to transfer a table that already exists in the destination database:

```
[INFO]:-Validating transfer table set...
[CRITICAL]:- gptransfer failed. (Reason='Table database.schema.table exists in database database.') exiting...
```

Override this behavior with the --skip-existing, --truncate, or --drop options.

The following table shows the objects that are copied in full mode and table mode.

<table>
<thead>
<tr>
<th>Object</th>
<th>Full Mode</th>
<th>Table Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Indexes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Roles</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Object</td>
<td>Full Mode</td>
<td>Table Mode</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>Functions</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Resource Queues</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>postgres.conf</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>pg_hba.conf</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>gp/pkg</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

The **--full** option and the **--schema-only** option can be used together if you want to copy databases in phases, for example, during scheduled periods of downtime or low activity. Run `gptransfer --full --schema-only ...` to create the databases on the destination cluster, but with no data. Then you can transfer the tables in stages during scheduled down times or periods of low activity. Be sure to include the **--truncate** or **--drop** option when you later transfer tables to prevent the transfer from failing because the table already exists at the destination.

### Locking

The **-x** option enables table locking. An exclusive lock is placed on the source table until the copy and validation, if requested, are complete.

### Validation

By default, `gptransfer` does not validate the data transferred. You can request validation using the **--validate=type** option. The validation type can be one of the following:

- **count** – Compares the row counts for the tables in the source and destination databases.
- **md5** – Sorts tables on both source and destination, and then performs a row-by-row comparison of the MD5 hashes of the sorted rows.

If the database is accessible during the transfer, be sure to add the **-x** option to lock the table. Otherwise, the table could be modified during the transfer, causing validation to fail.

### Failed Transfers

A failure on a table does not end the `gptransfer` job. When a transfer fails, `gptransfer` displays an error message and adds the table name to a failed transfers file. At the end of the `gptransfer` session, `gptransfer` writes a message telling you there were failures, and providing the name of the failed transfer file. For example:

```
[WARNING]: Some tables failed to transfer. A list of these tables
[WARNING]: has been written to the file
  failed_transfer_tables_20140808_101813.txt
[WARNING]: This file can be used with the -f option to continue
```

The failed transfers file is in the format required by the **-f** option, so you can use it to start a new `gptransfer` session to retry the failed transfers.

### Best Practices

Be careful not to exceed host memory by specifying too much parallelism with the **--batch-size** and **--sub-batch-size** command line options. Too many sub-processes can exhaust memory, causing a Python Out of Memory error. Start with a smaller batch size and sub-batch size, and increase based on your experiences.

Transfer a database in stages. First, run `gptransfer` with the **--schema-only** and **-d database** options, then transfer the tables in phases. After running `gptransfer` with the **--schema-only** option, be sure to add the **--truncate** or **--drop** option to prevent a failure because a table already exists.
Be careful choosing `gpfdist` and external table parameters such as the delimiter for external table data and the maximum line length. For example, don't choose a delimiter that can appear within table data.

If you have many empty tables to transfer, consider a DDL script instead of `gptransfer`. The `gptransfer` overhead to set up each table for transfer is significant and not an efficient way to transfer empty tables.

`gptransfer` creates table indexes before transferring the data. This slows the data transfer since indexes are updated at the same time the data is inserted in the table. For large tables especially, consider dropping indexes before running `gptransfer` and recreating the indexes when the transfer is complete.

**Monitoring a Greenplum System**

You can monitor a Greenplum Database system using a variety of tools included with the system or available as add-ons. SNMP support allows Greenplum to be integrated with popular system management frameworks.

Observing the Greenplum Database system day-to-day performance helps administrators understand the system behavior, plan workflow, and troubleshoot problems. This chapter discusses tools for monitoring database performance and activity.

Also, be sure to review *Recommended Monitoring and Maintenance Tasks* for monitoring activities you can script to quickly detect problems in the system.

**Monitoring Database Activity and Performance**

Greenplum Database includes an optional system monitoring and management database, `gpperfmon`, that administrators can enable. The `gpperfmon_install` command-line utility creates the `gpperfmon` database and enables data collection agents that collect and store query and system metrics in the database. Administrators can query metrics in the `gpperfmon` database. See the documentation for the `gpperfmon` database in the *Greenplum Database Reference Guide*.

Pivotal Greenplum Command Center, an optional web-based interface, graphically displays the metrics collected in the `gpperfmon` database and provides additional system management tools. Download the Greenplum Command Center package from Pivotal Network and view the documentation at the Greenplum Command Center Documentation web site.

**Monitoring System State**

As a Greenplum Database administrator, you must monitor the system for problem events such as a segment going down or running out of disk space on a segment host. The following topics describe how to monitor the health of a Greenplum Database system and examine certain state information for a Greenplum Database system.

• **Enabling System Alerts and Notifications**
  • **Checking System State**
  • **Checking Disk Space Usage**
  • **Checking for Data Distribution Skew**
  • **Viewing Metadata Information about Database Objects**
  • **Viewing Session Memory Usage Information**
  • **Viewing Query Workfile Usage Information**

**Enabling System Alerts and Notifications**

You can configure a Greenplum Database system to trigger SNMP (Simple Network Management Protocol) alerts or send email notifications to system administrators if certain database events occur. These events include:

• All PANIC-level error conditions
• All FATAL-level error conditions
• ERROR-level conditions that are "internal errors" (for example, SIGSEGV errors)
• Database system shutdown and restart
• Segment failure and recovery
• Standby master out-of-sync conditions
• Master host manual shutdown or other software problem (in certain failure scenarios, Greenplum Database cannot send an alert or notification)

This topic includes the following sub-topics:
• Using SNMP with a Greenplum Database System
• Enabling Email Notifications

Note that SNMP alerts and email notifications report the same event information. There is no difference in the event information that either tool reports. For information about the SNMP event information, see Greenplum Database SNMP OIDs and Error Codes.

Using SNMP with a Greenplum Database System

Greenplum Database supports SNMP to monitor the state of a Greenplum Database system using MIBs (Management Information Bases). MIBs are collections of objects that describe an SNMP-manageable entity — in this case, a Greenplum Database system.

The Greenplum Database SNMP support allows a Network Management System to obtain information about the hardware, operating system, and Greenplum Database from the same port (161) and IP address. It also enables the auto-discovery of Greenplum Database instances.

Prerequisites

Before setting up SNMP support on Greenplum Database, ensure SNMP is installed on the master host. If the `snmpd` file is not present in the `/usr/sbin` directory, then SNMP is not installed on the system. Depending on the platform on which you are running Greenplum Database, install the following:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Packages¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Hat Enterprise</td>
<td>net-snmp</td>
</tr>
<tr>
<td></td>
<td>net-snmp-libs</td>
</tr>
<tr>
<td></td>
<td>net-snmp-utils</td>
</tr>
<tr>
<td>CentOS</td>
<td>net-snmp</td>
</tr>
<tr>
<td>SUSE</td>
<td>N/A</td>
</tr>
</tbody>
</table>

1. SNMP is installed by default on SUSE platforms.

The `snmp.conf` configuration file is located in `/etc/snmp/`.

Pre-installation Tasks

After you establish that SNMP is on the master host, log in as `root`, open a text editor, and edit the `path_to/snmp/snmpd.conf` file. To use SNMP with Greenplum Database, the minimum configuration change required to the `snmpd.conf` file is specifying a community name. For example:

```
rocommunity public
```
Note: Replace public with the name of your SNMP community. You should also configure syslocation and syscontact. Configure other SNMP settings as required for your environment and save the file.

For more information about the snmpd.conf file, enter:

```
man snmpd.conf
```

Note: On SUSE Linux platforms, make sure to review and configure security settings in the snmp.conf file so snmpd accepts connections from sub-agents and returns all available Object IDs (OIDs).

After you finish configuring the snmpd.conf file, start the system snmpd daemon:

```
# /sbin/chkconfig snmpd on
```

Then, verify the system snmpd daemon is running. Enter:

```
# snmpwalk -v 1 -c community_name localhost .1.3.6.1.2.1.1.1.0
```

For example:

```
# snmpwalk -v 1 -c public localhost .1.3.6.1.2.1.1.1.0
```

If this command returns "Timeout: No Response from localhost", then the system snmpd daemon is not running. If the daemon is running, output similar to the following displays:

```
SNMPv2-MIB::sysDescr.0 = STRING: Linux hostname
2.6.18-92.el5 #1 SMP Tue Jun 10 18:51:06 EDT 2016 x86_64
```

### Setting up SNMP Notifications

1. To configure a Greenplum Database system to send SNMP notifications when alerts occur, set the following parameters on the Greenplum Database master host with the gconfig utility:

   - `gp_snmp_community`: Set this parameter to the community name you specified for your environment.
   - `gp_snmp_monitor_address`: Enter the hostname:port of your network monitor application. Typically, the port number is 162. If there are multiple monitor addresses, separate them with a comma.
   - `gp_snmp_use_inform_or_trap`: Enter either `trap` or `inform`. Trap notifications are SNMP messages sent from one application to another (for example, between Greenplum Database and a network monitoring application). These messages are unacknowledged by the monitoring application, but generate less network overhead.

     Inform notifications are the same as trap messages, except the application sends an acknowledgement to the application that generated the alert. In this case, the monitoring application sends acknowledgement messages to Greenplum Database-generated trap notifications. While inform messages create more overhead, they inform Greenplum Database the monitoring application has received the traps.

The following example commands set the server configuration parameters with the Greenplum Database gconfig utility:

```
$ gpconfig -c gp_snmp_community -v public --masteronly
$ gpconfig -c gp_snmp_monitor_address -v mdw:162 --masteronly
$ gpconfig -c gp_snmp_use_inform_or_trap -v trap --masteronly
```
2. To test SNMP notifications, you can use the `snmptrapd` trap receiver. As root, enter:

```
# /usr/sbin/snmptrapd -m ALL -Lf ~/filename.log
```

- `Lf` indicates that traps are logged to a file. `-Le` indicates that traps are logged to stderr instead. `-m` ALL loads all available MIBs (you can also specify individual MIBs if required).

**Enabling Email Notifications**

Complete the following steps to enable Greenplum Database to send email notifications to system administrators whenever certain database events occur.

1. Open `$MASTER_DATA_DIRECTORY/postgresql.conf` in a text editor.
2. In the EMAIL ALERTS section, uncomment the following parameters and enter the appropriate values for your email server and domain. For example:

```
gp_email_smtp_server='smtp.company.com:25'
gp_email_smtp_userid='gpadmin@example.com'
gp_email_smtp_password='mypassword'
gp_email_from='Greenplum Database <gpadmin@example.com>'
gp_email_to='dba@example.com;John Smith <jsmith@example.com>
```

You may create specific email accounts or groups in your email system that send and receive email alerts from the Greenplum Database system. For example:

```
gp_email_from='GPDB Production Instance <gpdb@example.com>'
gp_email_to='gpdb_dba_group@example.com'
```

You can also specify multiple email addresses for both `gp_email` parameters. Use a semi-colon (`;`) to separate each email address. For example:

```
gp_email_to='gpdb_dba_group@example.com;admin@example.com'
```

3. Save and close the `postgresql.conf` file.
4. Reload the Greenplum Database `postgresql.conf` file:

```
$ gpstop -u
```

**Testing Email Notifications**

The Greenplum Database master host must be able to connect to the SMTP email server you specify for the `gp_email_smtp_server` parameter. To test connectivity, use the `ping` command:

```
$ ping my_email_server
```

If the master host can contact the SMTP server, log in to `psql` and test email notifications with the following command:

```
$ psql postgres
=# SELECT gp_elog('Test GPDB Email',true); gp_elog
```

The address you specified for the `gp_email_to` parameter should receive an email with Test GPDB Email in the subject line.

You can also test email notifications by using a public SMTP server, such as Google's Gmail SMTP server, and an external email address. For example:

```
gp_email_smtp_server='smtp.gmail.com:25'
#gp_email_smtp_userid=''
```
#gp_email_smtp_password=''
gp_email_from='gpadmin@example.com'
gp_email_to='test_account@example.com'

**Note:** If you have difficulty sending and receiving email notifications, verify the security settings for your organization's email server and firewall.

### Checking System State

A Greenplum Database system is comprised of multiple PostgreSQL instances (the master and segments) spanning multiple machines. To monitor a Greenplum Database system, you need to know information about the system as a whole, as well as status information of the individual instances. The *gpstate* utility provides status information about a Greenplum Database system.

### Viewing Master and Segment Status and Configuration

The default *gpstate* action is to check segment instances and show a brief status of the valid and failed segments. For example, to see a quick status of your Greenplum Database system:

```bash
$ gpstate
```

To see more detailed information about your Greenplum Database array configuration, use *gpstate* with the `-s` option:

```bash
$ gpstate -s
```

### Viewing Your Mirroring Configuration and Status

If you are using mirroring for data redundancy, you may want to see the list of mirror segment instances in the system, their current synchronization status, and the mirror to primary mapping. For example, to see the mirror segments in the system and their status:

```bash
$ gpstate -m
```

To see the primary to mirror segment mappings:

```bash
$ gpstate -c
```

To see the status of the standby master mirror:

```bash
$ gpstate -f
```

### Checking Disk Space Usage

A database administrator's most important monitoring task is to make sure the file systems where the master and segment data directories reside do not grow to more than 70 percent full. A filled data disk will not result in data corruption, but it may prevent normal database activity from continuing. If the disk grows too full, it can cause the database server to shut down.

You can use the *gp_disk_free* external table in the *gp_toolkit* administrative schema to check for remaining free space (in kilobytes) on the segment host file systems. For example:

```sql
=# SELECT * FROM gp_toolkit.gp_disk_free
   ORDER BY dfsegment;
```
**Checking Sizing of Distributed Databases and Tables**

The `gp_toolkit` administrative schema contains several views that you can use to determine the disk space usage for a distributed Greenplum Database database, schema, table, or index.

For a list of the available sizing views for checking database object sizes and disk space, see the Greenplum Database Reference Guide.

**Viewing Disk Space Usage for a Database**

To see the total size of a database (in bytes), use the `gp_size_of_database` view in the `gp_toolkit` administrative schema. For example:

```
=> SELECT * FROM gp_toolkit.gp_size_of_database
     ORDER BY sodddatname;
```

**Viewing Disk Space Usage for a Table**

The `gp_toolkit` administrative schema contains several views for checking the size of a table. The table sizing views list the table by object ID (not by name). To check the size of a table by name, you must look up the relation name (`relname`) in the `pg_class` table. For example:

```
=> SELECT relname AS name, sotdsize AS size, sotdtoastsize
    AS toast, sotdadditionalsize AS other
    FROM gp_toolkit.gp_size_of_table_disk AS sotd, pg_class
    WHERE sotd.sotdoid=pg_class.oid ORDER BY relname;
```

For a list of the available table sizing views, see the Greenplum Database Reference Guide.

**Viewing Disk Space Usage for Indexes**

The `gp_toolkit` administrative schema contains a number of views for checking index sizes. To see the total size of all index(es) on a table, use the `gp_size_of_all_table_indexes` view. To see the size of a particular index, use the `gp_size_of_index` view. The index sizing views list tables and indexes by object ID (not by name). To check the size of an index by name, you must look up the relation name (`relname`) in the `pg_class` table. For example:

```
=> 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';
```

**Checking for Data Distribution Skew**

All tables in Greenplum Database are distributed, meaning their data is divided evenly across all of the segments in the system. Unevenly distributed data may diminish query processing performance. A table's distribution policy is determined at table creation time. For information about choosing the table distribution policy, see the following topics:

- Viewing a Table's Distribution Key
- Viewing Data Distribution
- Checking for Query Processing Skew

The `gp_toolkit` administrative schema also contains a number of views for checking data distribution skew on a table. For information about how to check for uneven data distribution, see the Greenplum Database Reference Guide.
**Viewing a Table's Distribution Key**

To see the columns used as the data distribution key for a table, you can use the `\d+` meta-command in `psql` to examine the definition of a table. For example:

```
=# \d+ sales
```

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sale_id</td>
<td>integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>amt</td>
<td>float</td>
<td></td>
<td></td>
</tr>
<tr>
<td>date</td>
<td>date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Has OIDs: no</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distributed by: (sale_id)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Viewing Data Distribution**

To see the data distribution of a table's rows (the number of rows on each segment), you can run a query such as:

```
=# SELECT gp_segment_id, count(*)
       FROM table_name GROUP BY gp_segment_id;
```

A table is considered to have a balanced distribution if all segments have roughly the same number of rows.

**Checking for Query Processing Skew**

When a query is being processed, all segments should have equal workloads to ensure the best possible performance. If you identify a poorly-performing query, you may need to investigate further using the `EXPLAIN` command. For information about using the `EXPLAIN` command and query profiling, see [Query Profiling](#).

Query processing workload can be skewed if the table's data distribution policy and the query predicates are not well matched. To check for processing skew, you can run a query such as:

```
=# SELECT gp_segment_id, count(*) FROM table_name
       WHERE column='value' GROUP BY gp_segment_id;
```

This will show the number of rows returned by segment for the given `WHERE` predicate.

**Avoiding an Extreme Skew Warning**

You may receive the following warning message while executing a query that performs a hash join operation:

```
Extreme skew in the innerside of Hashjoin
```

This occurs when the input to a hash join operator is skewed. It does not prevent the query from completing successfully. You can follow these steps to avoid skew in the plan:

1. Ensure that all fact tables are analyzed.
2. Verify that any populated temporary table used by the query is analyzed.
3. View the `EXPLAIN ANALYZE` plan for the query and look for the following:
   - If there are scans with multi-column filters that are producing more rows than estimated, then set the `gp_selectivity_damping_factor` server configuration parameter to 2 or higher and retest the query.
   - If the skew occurs while joining a single fact table that is relatively small (less than 5000 rows), set the `gp_segments_for_planner` server configuration parameter to 1 and retest the query.
4. Check whether the filters applied in the query match distribution keys of the base tables. If the filters and distribution keys are the same, consider redistributing some of the base tables with different distribution keys.

5. Check the cardinality of the join keys. If they have low cardinality, try to rewrite the query with different joining columns or additional filters on the tables to reduce the number of rows. These changes could change the query semantics.

**Viewing Metadata Information about Database Objects**

Greenplum Database tracks various metadata information in its system catalogs about the objects stored in a database, such as tables, views, indexes and so on, as well as global objects such as roles and tablespaces.

**Viewing the Last Operation Performed**

You can use the system views `pg_stat_operations` and `pg_stat_partition_operations` to look up actions performed on an object, such as a table. For example, to see the actions performed on a table, such as when it was created and when it was last vacuumed and analyzed:

```sql
=> SELECT schemaname as schema, objname as table, usename 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)
```

**Viewing the Definition of an Object**

To see the definition of an object, such as a table or view, you can use the `\d+` meta-command when working in `psql`. For example, to see the definition of a table:

```sql
=> \d+ mytable
```

**Viewing Session Memory Usage Information**

You can create and use the `session_level_memory_consumption` view that provides information about the current memory utilization for sessions that are running queries on Greenplum Database. The view contains session information and information such as the database that the session is connected to, the query that the session is currently running, and memory consumed by the session processes.

**Creating the session_level_memory_consumption View**

To create the `session_level_memory_consumption` view in a Greenplum Database, run the script `$/GPHOME/share/postgresql/contrib/gp_session_state.sql` once for each database. For example, to install the view in the database `testdb`, use this command:

```bash
$ psql -d testdb -f $GPHOME/share/postgresql/contrib/gp_session_state.sql
```

**The session_level_memory_consumption View**

The `session_level_memory_consumption` view provides information about memory consumption and idle time for sessions that are running SQL queries.
In the view, the column `is_runaway` indicates whether Greenplum Database considers the session a runaway session based on the vmem memory consumption of the session's queries. When the queries consume an excessive amount of memory, Greenplum Database considers the session a runaway. The Greenplum Database server configuration parameter `runway_detector_activation_percent` controlling when Greenplum Database considers a session a runaway session.

For information about the parameter, see "Server Configuration Parameters" in the Greenplum Database Reference Guide.

**Table 26: session_level_memory_consumption**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datname</td>
<td>name</td>
<td></td>
<td>Name of the database that the session is connected to.</td>
</tr>
<tr>
<td>sess_id</td>
<td>integer</td>
<td></td>
<td>Session ID.</td>
</tr>
<tr>
<td>usename</td>
<td>name</td>
<td></td>
<td>Name of the session user.</td>
</tr>
<tr>
<td>current_query</td>
<td>text</td>
<td></td>
<td>Current SQL query that the session is running.</td>
</tr>
<tr>
<td>segid</td>
<td>integer</td>
<td></td>
<td>Segment ID.</td>
</tr>
<tr>
<td>vmem_mb</td>
<td>integer</td>
<td></td>
<td>Total vmem memory usage for the session in MB.</td>
</tr>
<tr>
<td>is_runaway</td>
<td>boolean</td>
<td></td>
<td>Session is marked as runaway on the segment.</td>
</tr>
<tr>
<td>qe_count</td>
<td>integer</td>
<td></td>
<td>Number of query processes for the session.</td>
</tr>
<tr>
<td>active_qe_count</td>
<td>integer</td>
<td></td>
<td>Number of active query processes for the session.</td>
</tr>
<tr>
<td>dirty_qe_count</td>
<td>integer</td>
<td></td>
<td>Number of query processes that have not yet released their memory.</td>
</tr>
<tr>
<td>runaway_vmem_mb</td>
<td>integer</td>
<td></td>
<td>Amount of vmem memory that the session was consuming when it was marked as</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>runaway session.</td>
</tr>
<tr>
<td>runaway_command_cnt</td>
<td>integer</td>
<td></td>
<td>Command count for the session when it was marked as a runaway session.</td>
</tr>
</tbody>
</table>
Viewing Query Workfile Usage Information

The Greenplum Database administrative schema gp_toolkit contains views that display information about Greenplum Database workfiles. Greenplum Database creates workfiles on disk if it does not have sufficient memory to execute the query in memory. This information can be used for troubleshooting and tuning queries. The information in the views can also be used to specify the values for the Greenplum Database configuration parameters gp_workfile_limit_per_query and gp_workfile_limit_per_segment.

These are the views in the schema gp_toolkit:

- The gp_workfile_entries view contains one row for each operator using disk space for workfiles on a segment at the current time.
- The gp_workfile_usage_per_query view contains one row for each query using disk space for workfiles on a segment at the current time.
- The gp_workfile_usage_per_segment view contains one row for each segment. Each row displays the total amount of disk space used for workfiles on the segment at the current time.

For information about using gp_toolkit, see Using gp_toolkit.

Viewing the Database Server Log Files

Every database instance in Greenplum Database (master and segments) runs a PostgreSQL database server with its own server log file. Daily log files are created in the pg_log directory of the master and each segment data directory.

Log File Format

The server log files are written in comma-separated values (CSV) format. Some log entries will not have values for all log fields. For example, only log entries associated with a query worker process will have the slice_id populated. You can identify related log entries of a particular query by the query's session identifier (gp_session_id) and command identifier (gp_command_count).

The following fields are written to the log:

Table 27: Greenplum Database Server Log Format

<table>
<thead>
<tr>
<th>#</th>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>event_time</td>
<td>timestamp with time zone</td>
<td>Time that the log entry was written to the log</td>
</tr>
<tr>
<td>2</td>
<td>user_name</td>
<td>varchar(100)</td>
<td>The database user name</td>
</tr>
<tr>
<td>3</td>
<td>database_name</td>
<td>varchar(100)</td>
<td>The database name</td>
</tr>
<tr>
<td>4</td>
<td>process_id</td>
<td>varchar(10)</td>
<td>The system process ID (prefixed with &quot;p&quot;)</td>
</tr>
<tr>
<td>5</td>
<td>thread_id</td>
<td>varchar(50)</td>
<td>The thread count (prefixed with &quot;th&quot;)</td>
</tr>
<tr>
<td>6</td>
<td>remote_host</td>
<td>varchar(100)</td>
<td>On the master, the hostname/address of the client machine. On the segment, the hostname/address of the master.</td>
</tr>
<tr>
<td>7</td>
<td>remote_port</td>
<td>varchar(10)</td>
<td>The segment or master port number</td>
</tr>
<tr>
<td>#</td>
<td>Field Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----</td>
<td>--------------------</td>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>8</td>
<td>session_start_time</td>
<td>timestamp with time zone</td>
<td>Time session connection was opened</td>
</tr>
<tr>
<td>9</td>
<td>transaction_id</td>
<td>int</td>
<td>Top-level transaction ID on the master. This ID is the parent of any subtransactions.</td>
</tr>
<tr>
<td>10</td>
<td>gp_session_id</td>
<td>text</td>
<td>Session identifier number (prefix with &quot;con&quot;)</td>
</tr>
<tr>
<td>11</td>
<td>gp_command_count</td>
<td>text</td>
<td>The command number within a session (prefix with &quot;cmd&quot;)</td>
</tr>
<tr>
<td>12</td>
<td>gp_segment</td>
<td>text</td>
<td>The segment content identifier (prefix with &quot;seg&quot; for primaries or &quot;mir&quot; for mirrors). The master always has a content ID of -1.</td>
</tr>
<tr>
<td>13</td>
<td>slice_id</td>
<td>text</td>
<td>The slice ID (portion of the query plan being executed)</td>
</tr>
<tr>
<td>14</td>
<td>distr_trx_id</td>
<td>text</td>
<td>Distributed transaction ID</td>
</tr>
<tr>
<td>15</td>
<td>local_trx_id</td>
<td>text</td>
<td>Local transaction ID</td>
</tr>
<tr>
<td>16</td>
<td>sub_trx_id</td>
<td>text</td>
<td>Subtransaction ID</td>
</tr>
<tr>
<td>17</td>
<td>event_severity</td>
<td>varchar(10)</td>
<td>Values include: LOG, ERROR, FATAL, PANIC, DEBUG1, DEBUG2</td>
</tr>
<tr>
<td>18</td>
<td>sql_state_code</td>
<td>varchar(10)</td>
<td>SQL state code associated with the log message</td>
</tr>
<tr>
<td>19</td>
<td>event_message</td>
<td>text</td>
<td>Log or error message text</td>
</tr>
<tr>
<td>20</td>
<td>event_detail</td>
<td>text</td>
<td>Detail message text associated with an error or warning message</td>
</tr>
<tr>
<td>21</td>
<td>event_hint</td>
<td>text</td>
<td>Hint message text associated with an error or warning message</td>
</tr>
<tr>
<td>22</td>
<td>internal_query</td>
<td>text</td>
<td>The internally-generated query text</td>
</tr>
<tr>
<td>23</td>
<td>internal_query_pos</td>
<td>int</td>
<td>The cursor index into the internally-generated query text</td>
</tr>
<tr>
<td>24</td>
<td>event_context</td>
<td>text</td>
<td>The context in which this message gets generated</td>
</tr>
<tr>
<td>25</td>
<td>debug_query_string</td>
<td>text</td>
<td>User-supplied query string with full detail for debugging. This string can be modified for internal use.</td>
</tr>
<tr>
<td>26</td>
<td>error_cursor_pos</td>
<td>int</td>
<td>The cursor index into the query string</td>
</tr>
<tr>
<td>27</td>
<td>func_name</td>
<td>text</td>
<td>The function in which this message is generated</td>
</tr>
<tr>
<td>28</td>
<td>file_name</td>
<td>text</td>
<td>The internal code file where the message originated</td>
</tr>
<tr>
<td>29</td>
<td>file_line</td>
<td>int</td>
<td>The line of the code file where the message originated</td>
</tr>
<tr>
<td>30</td>
<td>stack_trace</td>
<td>text</td>
<td>Stack trace text associated with this message</td>
</tr>
</tbody>
</table>

**Searching the Greenplum Server Log Files**

Greenplum Database provides a utility called `gplogfilter` can search through a Greenplum Database log file for entries matching the specified criteria. By default, this utility searches through the Greenplum
Database master log file in the default logging location. For example, to display the last three lines of the master log file:

```bash
$ gplogfilter -n 3
```

To search through all segment log files simultaneously, run `gplogfilter` through the `gpssh` utility. For example, to display the last three lines of each segment log file:

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

### Using gp_toolkit

Use the Greenplum Database administrative schema `gp_toolkit` to query the system catalogs, log files, and operating environment for system status information. The `gp_toolkit` schema contains several views you can access using SQL commands. The `gp_toolkit` schema is accessible to all database users. Some objects require superuser permissions. Use a command similar to the following to add the `gp_toolkit` schema to your schema search path:

```sql
=> ALTER ROLE myrole SET search_path TO myschema,gp_toolkit;
```

For a description of the available administrative schema views and their usages, see the *Greenplum Database Reference Guide*.

### Greenplum Database SNMP OIDs and Error Codes

When a Greenplum Database system is configured to trigger SNMP alerts or send email notifications to system administrators if certain database events occur, the alerts and notifications contain Object IDs (OIDs) and SQL error codes.

- *Greenplum Database SNMP OIDs*
- *SQL Standard Error Codes*

For information about enabling Greenplum Database to use SNMP, see *Enabling System Alerts and Notifications*.

#### Greenplum Database SNMP OIDs

This is the Greenplum Database OID hierarchy structure:

```plaintext
iso(1)
identified-organization(3)
dod(6)
internet(1)
private(4)
enterprises(1)
gpdbMIB(31327)
gpdbObjects(1)
gpdbAlertMsg(1)
```

`gdbAlertMsg`

```
1.3.6.1.4.1.31327.1.1: STRING: alert message text
```
gpdbAlertSeverity

1.3.6.1.4.1.31327.1.2: INTEGER: severity level

gpdbAlertSeverity can have one of the following values:

- gpdbSevUnknown (0)
- gpdbSevOk (1)
- gpdbSevWarning (2)
- gpdbSevError (3)
- gpdbSevFatal (4)
- gpdbSevPanic (5)
- gpdbSevSystemDegraded (6)
- gpdbSevSystemDown (7)

gpdbAlertSqlstate

1.3.6.1.4.1.31327.1.3: STRING: SQL standard error codes

For a list of codes, see SQL Standard Error Codes.

gpdbAlertDetail

1.3.6.1.4.1.31327.1.4: STRING: detailed alert message text

gpdbAlertSqlStmt

1.3.6.1.4.1.31327.1.5: STRING: SQL statement generating this alert if applicable

gpdbAlertSystemName

1.3.6.1.4.1.31327.1.6: STRING: hostname

SQL Standard Error Codes

The following table lists all the defined error codes. Some are not used, but are defined by the SQL standard. The error classes are also shown. For each error class there is a standard error code having the last three characters 000. This code is used only for error conditions that fall within the class but do not have any more-specific code assigned.

The PL/pgSQL condition name for each error code is the same as the phrase shown in the table, with underscores substituted for spaces. For example, code 22012, DIVISION BY ZERO, has condition name DIVISION_BY_ZERO. Condition names can be written in either upper or lower case.

**Note:** PL/pgSQL does not recognize warning, as opposed to error, condition names; those are classes 00, 01, and 02.

**Table 28: SQL Codes**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Meaning</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 00</td>
<td>Successful Completion</td>
<td></td>
</tr>
<tr>
<td>00000</td>
<td>SUCCESSFUL COMPLETION</td>
<td>successful_completion</td>
</tr>
<tr>
<td>Class 01</td>
<td>Warning</td>
<td></td>
</tr>
<tr>
<td>01000</td>
<td>WARNING</td>
<td>warning</td>
</tr>
<tr>
<td>0100C</td>
<td>DYNAMIC RESULT SETS RETURNED</td>
<td>dynamic_result_sets_returned</td>
</tr>
<tr>
<td>Error Code</td>
<td>Meaning</td>
<td>Constant</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>01008</td>
<td>IMPLICIT ZERO BIT Padding</td>
<td>implicit_zero_bit_padding</td>
</tr>
<tr>
<td>01003</td>
<td>NULL VALUE ELIMINATED IN SET FUNCTION</td>
<td>null_value_eliminated_in_set_function</td>
</tr>
<tr>
<td>01007</td>
<td>PRIVILEGE NOT GRANTED</td>
<td>privilege_not_granted</td>
</tr>
<tr>
<td>01006</td>
<td>PRIVILEGE NOT REVOKED</td>
<td>privilege_not_revoked</td>
</tr>
<tr>
<td>01004</td>
<td>STRING DATA RIGHT TRUNCATION</td>
<td>string_data_right_truncation</td>
</tr>
<tr>
<td>01P01</td>
<td>DEPRECATED FEATURE</td>
<td>deprecated_feature</td>
</tr>
<tr>
<td>02000</td>
<td>NO DATA</td>
<td>no_data</td>
</tr>
<tr>
<td>02001</td>
<td>NO ADDITIONAL DYNAMIC RESULT SETS RETURNED</td>
<td>no_additional_dynamic_result_sets_returned</td>
</tr>
<tr>
<td>03000</td>
<td>SQL STATEMENT NOT YET COMPLETE</td>
<td>sql_statement_not_yet_complete</td>
</tr>
<tr>
<td>08000</td>
<td>CONNECTION EXCEPTION</td>
<td>connection_exception</td>
</tr>
<tr>
<td>08003</td>
<td>CONNECTION DOES NOT EXIST</td>
<td>connection_does_not_exist</td>
</tr>
<tr>
<td>08006</td>
<td>CONNECTION FAILURE</td>
<td>connection_failure</td>
</tr>
<tr>
<td>08001</td>
<td>SQLCLIENT UNABLE TO ESTABLISH SQLCONNECTION</td>
<td>sqlclient_unable_to_establish_sqlconnection</td>
</tr>
<tr>
<td>08004</td>
<td>SQLSERVER REJECTED ESTABLISHMENT OF SQLCONNECTION</td>
<td>sqlserver_rejected_establishment_of_sqlconnection</td>
</tr>
<tr>
<td>08007</td>
<td>TRANSACTION RESOLUTION UNKNOWN</td>
<td>transaction_resolution_unknown</td>
</tr>
<tr>
<td>08P01</td>
<td>PROTOCOL VIOLATION</td>
<td>protocol_violation</td>
</tr>
<tr>
<td>09000</td>
<td>TRIGGERED ACTION EXCEPTION</td>
<td>triggered_action_exception</td>
</tr>
<tr>
<td>0A000</td>
<td>FEATURE NOT SUPPORTED</td>
<td>feature_not_supported</td>
</tr>
<tr>
<td>0B000</td>
<td>INVALID TRANSACTION INITIATION</td>
<td>invalid_transaction_initiation</td>
</tr>
<tr>
<td>0F000</td>
<td>LOCATOR EXCEPTION</td>
<td>locator_exception</td>
</tr>
<tr>
<td>0F001</td>
<td>INVALID LOCATOR SPECIFICATION</td>
<td>invalid_locator_specification</td>
</tr>
<tr>
<td>0L000</td>
<td>INVALID GRANTOR</td>
<td>invalid_grantor</td>
</tr>
<tr>
<td>Error Code</td>
<td>Meaning</td>
<td>Constant</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>0LP01</td>
<td>INVALID GRANT OPERATION</td>
<td>invalid_grant_operation</td>
</tr>
<tr>
<td>Class 0P</td>
<td>Invalid Role Specification</td>
<td></td>
</tr>
<tr>
<td>0P000</td>
<td>INVALID ROLE SPECIFICATION</td>
<td>invalid_role_specification</td>
</tr>
<tr>
<td>Class 21</td>
<td>Cardinality Violation</td>
<td></td>
</tr>
<tr>
<td>21000</td>
<td>CARDINALITY VIOLATION</td>
<td>cardinality_violation</td>
</tr>
<tr>
<td>Class 22</td>
<td>Data Exception</td>
<td></td>
</tr>
<tr>
<td>22000</td>
<td>DATA EXCEPTION</td>
<td>data_exception</td>
</tr>
<tr>
<td>2202E</td>
<td>ARRAY SUBSCRIPT ERROR</td>
<td>array_subscript_error</td>
</tr>
<tr>
<td>22021</td>
<td>CHARACTER NOT IN REPertoire</td>
<td>character_not_in_repertoire</td>
</tr>
<tr>
<td>22008</td>
<td>DATETIME FIELD OVERFLOW</td>
<td>datetime_field_overflow</td>
</tr>
<tr>
<td>22012</td>
<td>DIVISION BY ZERO</td>
<td>division_by_zero</td>
</tr>
<tr>
<td>22005</td>
<td>ERROR IN ASSIGNMENT</td>
<td>error_in_assignment</td>
</tr>
<tr>
<td>2200B</td>
<td>ESCAPE CHARACTER CONFLICT</td>
<td>escape_character_conflict</td>
</tr>
<tr>
<td>22022</td>
<td>INDICATOR OVERFLOW</td>
<td>indicator_overflow</td>
</tr>
<tr>
<td>22015</td>
<td>INTERVAL FIELD OVERFLOW</td>
<td>interval_field_overflow</td>
</tr>
<tr>
<td>2201E</td>
<td>INVALID ARGUMENT FOR LOGARITHM</td>
<td>invalid_argument_for_logarithm</td>
</tr>
<tr>
<td>2201F</td>
<td>INVALID ARGUMENT FOR POWER FUNCTION</td>
<td>invalid_argument_for_power_function</td>
</tr>
<tr>
<td>2201G</td>
<td>INVALID ARGUMENT FOR WIDTH BUCKET FUNCTION</td>
<td>invalid_argument_for_width_bucket_function</td>
</tr>
<tr>
<td>22018</td>
<td>INVALID CHARACTER VALUE FOR CAST</td>
<td>invalid_character_value_for_cast</td>
</tr>
<tr>
<td>22007</td>
<td>INVALID DATETIME FORMAT</td>
<td>invalid_datetime_format</td>
</tr>
<tr>
<td>22019</td>
<td>INVALID ESCAPE CHARACTER</td>
<td>invalid_escape_character</td>
</tr>
<tr>
<td>2200D</td>
<td>INVALID ESCAPE OCTET</td>
<td>invalid_escape_octet</td>
</tr>
<tr>
<td>22025</td>
<td>INVALID ESCAPE SEQUENCE</td>
<td>invalid_escape_sequence</td>
</tr>
<tr>
<td>22P06</td>
<td>NONSTANDARD USE OF ESCAPE CHARACTER</td>
<td>nonstandard_use_of_escape_character</td>
</tr>
<tr>
<td>22010</td>
<td>INVALID INDICATOR PARAMETER VALUE</td>
<td>invalid_indicator_parameter_value</td>
</tr>
<tr>
<td>22020</td>
<td>INVALID LIMIT VALUE</td>
<td>invalid_limit_value</td>
</tr>
<tr>
<td>22023</td>
<td>INVALID PARAMETER VALUE</td>
<td>invalid_parameter_value</td>
</tr>
<tr>
<td>2201B</td>
<td>INVALID REGULAR EXPRESSION</td>
<td>invalid_regular_expression</td>
</tr>
<tr>
<td>22009</td>
<td>INVALID TIME ZONE DISPLACEMENT VALUE</td>
<td>invalid_time_zone_displacement_value</td>
</tr>
<tr>
<td>Error Code</td>
<td>Meaning</td>
<td>Constant</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>2200C</td>
<td>INVALID USE OF ESCAPE CHARACTER</td>
<td>invalid_use_of_escape_character</td>
</tr>
<tr>
<td>2200G</td>
<td>MOST SPECIFIC TYPE MISMATCH</td>
<td>most_specific_type_mismatch</td>
</tr>
<tr>
<td>22004</td>
<td>NULL VALUE NOT ALLOWED</td>
<td>null_value_not_allowed</td>
</tr>
<tr>
<td>22002</td>
<td>NULL VALUE NO INDICATOR PARAMETER</td>
<td>null_value_no_indicator_parameter</td>
</tr>
<tr>
<td>22003</td>
<td>NUMERIC VALUE OUT OF RANGE</td>
<td>numeric_value_out_of_range</td>
</tr>
<tr>
<td>22026</td>
<td>STRING DATA LENGTH MISMATCH</td>
<td>string_data_length_mismatch</td>
</tr>
<tr>
<td>22001</td>
<td>STRING DATA RIGHT TRUNCATION</td>
<td>string_data_right_truncation</td>
</tr>
<tr>
<td>22011</td>
<td>SUBSTRING ERROR</td>
<td>substring_error</td>
</tr>
<tr>
<td>22027</td>
<td>TRIM ERROR</td>
<td>trim_error</td>
</tr>
<tr>
<td>22024</td>
<td>UNTERMINATED C STRING</td>
<td>unterminated_c_string</td>
</tr>
<tr>
<td>2200F</td>
<td>ZERO LENGTH CHARACTER STRING</td>
<td>zero_length_character_string</td>
</tr>
<tr>
<td>22P01</td>
<td>FLOATING POINT EXCEPTION</td>
<td>floating_point_exception</td>
</tr>
<tr>
<td>22P02</td>
<td>INVALID TEXT REPRESENTATION</td>
<td>invalid_text_representation</td>
</tr>
<tr>
<td>22P03</td>
<td>INVALID BINARY REPRESENTATION</td>
<td>invalid_binary_representation</td>
</tr>
<tr>
<td>22P04</td>
<td>BAD COPY FILE FORMAT</td>
<td>bad_copy_file_format</td>
</tr>
<tr>
<td>22P05</td>
<td>UNTRANSLATABLE CHARACTER</td>
<td>untranslatable_character</td>
</tr>
</tbody>
</table>

**Class 23 — Integrity Constraint Violation**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Meaning</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>23000</td>
<td>INTEGRITY CONSTRAINT VIOLATION</td>
<td>integrity_constraint_violation</td>
</tr>
<tr>
<td>23001</td>
<td>RESTRICT VIOLATION</td>
<td>restrict_violation</td>
</tr>
<tr>
<td>23502</td>
<td>NOT NULL VIOLATION</td>
<td>not_null_violation</td>
</tr>
<tr>
<td>23503</td>
<td>FOREIGN KEY VIOLATION</td>
<td>foreign_key_violation</td>
</tr>
<tr>
<td>23505</td>
<td>UNIQUE VIOLATION</td>
<td>unique_violation</td>
</tr>
<tr>
<td>23514</td>
<td>CHECK VIOLATION</td>
<td>check_violation</td>
</tr>
</tbody>
</table>

**Class 24 — Invalid Cursor State**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Meaning</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>24000</td>
<td>INVALID CURSOR STATE</td>
<td>invalid_cursor_state</td>
</tr>
</tbody>
</table>

**Class 25 — Invalid Transaction State**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Meaning</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>25000</td>
<td>INVALID TRANSACTION STATE</td>
<td>invalid_transaction_state</td>
</tr>
<tr>
<td>25001</td>
<td>ACTIVE SQL TRANSACTION</td>
<td>active_sql_transaction</td>
</tr>
<tr>
<td>25002</td>
<td>BRANCH TRANSACTION ALREADY ACTIVE</td>
<td>branch_transaction_already_active</td>
</tr>
<tr>
<td>25008</td>
<td>HELD CURSOR REQUIRES SAME ISOLATION LEVEL</td>
<td>held_cursor_requires_same_isolation_level</td>
</tr>
<tr>
<td>25003</td>
<td>INAPPROPRIATE ACCESS MODE FOR BRANCH TRANSACTION</td>
<td>inappropriate_access_mode_for_branch_transaction</td>
</tr>
<tr>
<td>Error Code</td>
<td>Meaning</td>
<td>Constant</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>25004</td>
<td>INAPPROPRIATE ISOLATION LEVEL FOR BRANCH TRANSACTION</td>
<td>inappropriate_isolation_level_for_branch_transaction</td>
</tr>
<tr>
<td>25005</td>
<td>NO ACTIVE SQL TRANSACTION FOR BRANCH TRANSACTION</td>
<td>no_active_sql_transaction_for_branch_transaction</td>
</tr>
<tr>
<td>25006</td>
<td>READ ONLY SQL TRANSACTION</td>
<td>read_only_sql_transaction</td>
</tr>
<tr>
<td>25007</td>
<td>SCHEMA AND DATA STATEMENT MIXING NOT SUPPORTED</td>
<td>schema_and_data_statement_mixing_not_supported</td>
</tr>
<tr>
<td>25P01</td>
<td>NO ACTIVE SQL TRANSACTION</td>
<td>no_active_sql_transaction</td>
</tr>
<tr>
<td>25P02</td>
<td>IN FAILED SQL TRANSACTION</td>
<td>in_failed_sql_transaction</td>
</tr>
<tr>
<td><strong>Class 26</strong></td>
<td>- Invalid SQL Statement Name</td>
<td></td>
</tr>
<tr>
<td>26000</td>
<td>INVALID SQL STATEMENT NAME</td>
<td>invalid_sql_statement_name</td>
</tr>
<tr>
<td><strong>Class 27</strong></td>
<td>- Triggered Data Change Violation</td>
<td></td>
</tr>
<tr>
<td>27000</td>
<td>TRIGGERED DATA CHANGE VIOLATION</td>
<td>triggered_data_change_violation</td>
</tr>
<tr>
<td><strong>Class 28</strong></td>
<td>- Invalid Authorization Specification</td>
<td></td>
</tr>
<tr>
<td>28000</td>
<td>INVALID AUTHORIZATION SPECIFICATION</td>
<td>invalid_authorization_specification</td>
</tr>
<tr>
<td><strong>Class 2B</strong></td>
<td>- Dependent Privilege Descriptors Still Exist</td>
<td></td>
</tr>
<tr>
<td>2B000</td>
<td>DEPENDENT PRIVILEGE DESCRIPTORS STILL EXIST</td>
<td>dependent_privilege_descriptors_still_exist</td>
</tr>
<tr>
<td>2BP01</td>
<td>DEPENDENT OBJECTS STILL EXIST</td>
<td>dependent_objects_still_exist</td>
</tr>
<tr>
<td><strong>Class 2D</strong></td>
<td>- Invalid Transaction Termination</td>
<td></td>
</tr>
<tr>
<td>2D000</td>
<td>INVALID TRANSACTION TERMINATION</td>
<td>invalid_transaction_termination</td>
</tr>
<tr>
<td><strong>Class 2F</strong></td>
<td>- SQL Routine Exception</td>
<td></td>
</tr>
<tr>
<td>2F000</td>
<td>SQL ROUTINE EXCEPTION</td>
<td>sql_routine_exception</td>
</tr>
<tr>
<td>2F005</td>
<td>FUNCTION EXECUTED NO RETURN STATEMENT</td>
<td>function_executed_no_return_statement</td>
</tr>
<tr>
<td>2F002</td>
<td>MODIFYING SQL DATA NOT PERMITTED</td>
<td>modifying_sql_data_not_permitted</td>
</tr>
<tr>
<td>2F003</td>
<td>PROHIBITED SQL STATEMENT ATTEMPTED</td>
<td>prohibited_sql_statement_attempted</td>
</tr>
<tr>
<td>2F004</td>
<td>READING SQL DATA NOT PERMITTED</td>
<td>reading_sql_data_not_permitted</td>
</tr>
<tr>
<td><strong>Class 34</strong></td>
<td>- Invalid Cursor Name</td>
<td></td>
</tr>
<tr>
<td>34000</td>
<td>INVALID CURSOR NAME</td>
<td>invalid_cursor_name</td>
</tr>
<tr>
<td><strong>Class 38</strong></td>
<td>- External Routine Exception</td>
<td></td>
</tr>
<tr>
<td>38000</td>
<td>EXTERNAL ROUTINE EXCEPTION</td>
<td>external_routine_exception</td>
</tr>
<tr>
<td>Error Code</td>
<td>Meaning</td>
<td>Constant</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>38001</td>
<td>CONTAINING SQL NOT PERMITTED</td>
<td>containing_sql_not_permitted</td>
</tr>
<tr>
<td>38002</td>
<td>MODIFYING SQL DATA NOT PERMITTED</td>
<td>modifying_sql_data_not_permitted</td>
</tr>
<tr>
<td>38003</td>
<td>PROHIBITED SQL STATEMENT ATTEMPTED</td>
<td>prohibited_sql_statement_attempted</td>
</tr>
<tr>
<td>38004</td>
<td>READING SQL DATA NOT PERMITTED</td>
<td>reading_sql_data_not_permitted</td>
</tr>
<tr>
<td><strong>Class 39</strong></td>
<td>External Routine Invocation Exception</td>
<td></td>
</tr>
<tr>
<td>39000</td>
<td>EXTERNAL ROUTINE INVOCATION EXCEPTION</td>
<td>external_routine_invocation_exception</td>
</tr>
<tr>
<td>39001</td>
<td>INVALID SQLSTATE RETURNED</td>
<td>invalid_sqlstate_returned</td>
</tr>
<tr>
<td>39004</td>
<td>NULL VALUE NOT ALLOWED</td>
<td>null_value_not_allowed</td>
</tr>
<tr>
<td>39P01</td>
<td>TRIGGER PROTOCOL VIOLATED</td>
<td>trigger_protocol_violated</td>
</tr>
<tr>
<td>39P02</td>
<td>SRF PROTOCOL VIOLATED</td>
<td>srf_protocol_violated</td>
</tr>
<tr>
<td><strong>Class 3B</strong></td>
<td>Savepoint Exception</td>
<td></td>
</tr>
<tr>
<td>3B000</td>
<td>SAVEPOINT EXCEPTION</td>
<td>savepoint_exception</td>
</tr>
<tr>
<td>3B001</td>
<td>INVALID SAVEPOINT SPECIFICATION</td>
<td>invalid_savepoint_specification</td>
</tr>
<tr>
<td><strong>Class 3D</strong></td>
<td>Invalid Catalog Name</td>
<td></td>
</tr>
<tr>
<td>3D000</td>
<td>INVALID CATALOG NAME</td>
<td>invalid_catalog_name</td>
</tr>
<tr>
<td><strong>Class 3F</strong></td>
<td>Invalid Schema Name</td>
<td></td>
</tr>
<tr>
<td>3F000</td>
<td>INVALID SCHEMA NAME</td>
<td>invalid_schema_name</td>
</tr>
<tr>
<td><strong>Class 40</strong></td>
<td>Transaction Rollback</td>
<td></td>
</tr>
<tr>
<td>40000</td>
<td>TRANSACTION ROLLBACK</td>
<td>transaction_rollback</td>
</tr>
<tr>
<td>40002</td>
<td>TRANSACTION INTEGRITY CONSTRAINT VIOLATION</td>
<td>transaction_integrity_constraint_violation</td>
</tr>
<tr>
<td>40001</td>
<td>SERIALIZATION FAILURE</td>
<td>serialization_failure</td>
</tr>
<tr>
<td>40003</td>
<td>STATEMENT COMPLETION UNKNOWN</td>
<td>statement_completion_unknown</td>
</tr>
<tr>
<td>40P01</td>
<td>DEADLOCK DETECTED</td>
<td>deadlock_detected</td>
</tr>
<tr>
<td><strong>Class 42</strong></td>
<td>Syntax Error or Access Rule Violation</td>
<td></td>
</tr>
<tr>
<td>42000</td>
<td>SYNTAX ERROR OR ACCESS RULE VIOLATION</td>
<td>syntax_error_or_access_rule_violation</td>
</tr>
<tr>
<td>42601</td>
<td>SYNTAX ERROR</td>
<td>syntax_error</td>
</tr>
<tr>
<td>42501</td>
<td>INSUFFICIENT PRIVILEGE</td>
<td>insufficient_privilege</td>
</tr>
<tr>
<td>42846</td>
<td>CANNOT COERCCE</td>
<td>cannot_coerce</td>
</tr>
<tr>
<td>42803</td>
<td>GROUPING ERROR</td>
<td>grouping_error</td>
</tr>
<tr>
<td>42830</td>
<td>INVALID FOREIGN KEY</td>
<td>invalid_foreign_key</td>
</tr>
<tr>
<td>Error Code</td>
<td>Meaning</td>
<td>Constant</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>42602</td>
<td>INVALID NAME</td>
<td>invalid_name</td>
</tr>
<tr>
<td>42622</td>
<td>NAME TOO LONG</td>
<td>name-too-long</td>
</tr>
<tr>
<td>42939</td>
<td>RESERVED NAME</td>
<td>reserved_name</td>
</tr>
<tr>
<td>42804</td>
<td>DATATYPE MISMATCH</td>
<td>datatype_mismatch</td>
</tr>
<tr>
<td>42P18</td>
<td>INDETERMINATE DATATYPE</td>
<td>indeterminate_datatype</td>
</tr>
<tr>
<td>42809</td>
<td>WRONG OBJECT TYPE</td>
<td>wrong_object_type</td>
</tr>
<tr>
<td>42703</td>
<td>UNDEFINED COLUMN</td>
<td>undefined_column</td>
</tr>
<tr>
<td>42883</td>
<td>UNDEFINED FUNCTION</td>
<td>undefined_function</td>
</tr>
<tr>
<td>42P01</td>
<td>UNDEFINED TABLE</td>
<td>undefined_table</td>
</tr>
<tr>
<td>42P02</td>
<td>UNDEFINED PARAMETER</td>
<td>undefined_parameter</td>
</tr>
<tr>
<td>42704</td>
<td>UNDEFINED OBJECT</td>
<td>undefined_object</td>
</tr>
<tr>
<td>42701</td>
<td>DUPLICATE COLUMN</td>
<td>duplicate_column</td>
</tr>
<tr>
<td>42P03</td>
<td>DUPLICATE CURSOR</td>
<td>duplicate_cursor</td>
</tr>
<tr>
<td>42P04</td>
<td>DUPLICATE DATABASE</td>
<td>duplicate_database</td>
</tr>
<tr>
<td>42723</td>
<td>DUPLICATE FUNCTION</td>
<td>duplicate_function</td>
</tr>
<tr>
<td>42P05</td>
<td>DUPLICATE PREPARED STATEMENT</td>
<td>duplicate_prepared_statement</td>
</tr>
<tr>
<td>42P06</td>
<td>DUPLICATE SCHEMA</td>
<td>duplicate_schema</td>
</tr>
<tr>
<td>42P07</td>
<td>DUPLICATE TABLE</td>
<td>duplicate_table</td>
</tr>
<tr>
<td>42712</td>
<td>DUPLICATE ALIAS</td>
<td>duplicate Alias</td>
</tr>
<tr>
<td>42710</td>
<td>DUPLICATE OBJECT</td>
<td>duplicate_object</td>
</tr>
<tr>
<td>42702</td>
<td>AMBIGUOUS COLUMN</td>
<td>ambiguous_column</td>
</tr>
<tr>
<td>42725</td>
<td>AMBIGUOUS FUNCTION</td>
<td>ambiguous_function</td>
</tr>
<tr>
<td>42P08</td>
<td>AMBIGUOUS PARAMETER</td>
<td>ambiguous_parameter</td>
</tr>
<tr>
<td>42P09</td>
<td>AMBIGUOUS ALIAS</td>
<td>ambiguous_alias</td>
</tr>
<tr>
<td>42P10</td>
<td>INVALID COLUMN REFERENCE</td>
<td>invalid_column_reference</td>
</tr>
<tr>
<td>42611</td>
<td>INVALID COLUMN DEFINITION</td>
<td>invalid_column_definition</td>
</tr>
<tr>
<td>42P11</td>
<td>INVALID CURSOR DEFINITION</td>
<td>invalid_cursor_definition</td>
</tr>
<tr>
<td>42P12</td>
<td>INVALID DATABASE DEFINITION</td>
<td>invalid_database_definition</td>
</tr>
<tr>
<td>42P13</td>
<td>INVALID FUNCTION DEFINITION</td>
<td>invalid_function_definition</td>
</tr>
<tr>
<td>42P14</td>
<td>INVALID PREPARED STATEMENT DEFINITION</td>
<td>invalid_prepared_statement_definition</td>
</tr>
<tr>
<td>42P15</td>
<td>INVALID SCHEMA DEFINITION</td>
<td>invalid_schema_definition</td>
</tr>
<tr>
<td>42P16</td>
<td>INVALID TABLE DEFINITION</td>
<td>invalid_table_definition</td>
</tr>
<tr>
<td>42P17</td>
<td>INVALID OBJECT DEFINITION</td>
<td>invalid_object_definition</td>
</tr>
</tbody>
</table>

Class 44 — WITH CHECK OPTION Violation
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Meaning</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>44000</td>
<td>WITH CHECK OPTION VIOLATION</td>
<td>with_check_option_violation</td>
</tr>
<tr>
<td></td>
<td><strong>Class 53</strong> — Insufficient Resources</td>
<td></td>
</tr>
<tr>
<td>53000</td>
<td>INSUFFICIENT RESOURCES</td>
<td>insufficient_resources</td>
</tr>
<tr>
<td>53100</td>
<td>DISK FULL</td>
<td>disk_full</td>
</tr>
<tr>
<td>53200</td>
<td>OUT OF MEMORY</td>
<td>out_of_memory</td>
</tr>
<tr>
<td>53300</td>
<td>TOO MANY CONNECTIONS</td>
<td>too_many_connections</td>
</tr>
<tr>
<td></td>
<td><strong>Class 54</strong> — Program Limit Exceeded</td>
<td></td>
</tr>
<tr>
<td>54000</td>
<td>PROGRAM LIMIT EXCEEDED</td>
<td>program_limit_exceeded</td>
</tr>
<tr>
<td>54001</td>
<td>STATEMENT TOO COMPLEX</td>
<td>statement_too_complex</td>
</tr>
<tr>
<td>54011</td>
<td>TOO MANY COLUMNS</td>
<td>too_many_columns</td>
</tr>
<tr>
<td>54023</td>
<td>TOO MANY ARGUMENTS</td>
<td>too_many_arguments</td>
</tr>
<tr>
<td></td>
<td><strong>Class 55</strong> — Object Not In Prerequisite State</td>
<td></td>
</tr>
<tr>
<td>55000</td>
<td>OBJECT NOT IN PREREQUISITE STATE</td>
<td>object_not_in_prerequisite_state</td>
</tr>
<tr>
<td>55006</td>
<td>OBJECT IN USE</td>
<td>object_in_use</td>
</tr>
<tr>
<td>55P02</td>
<td>CANT CHANGE RUNTIME PARAM</td>
<td>cant_change_runtime_param</td>
</tr>
<tr>
<td>55P03</td>
<td>LOCK NOT AVAILABLE</td>
<td>lock_not_available</td>
</tr>
<tr>
<td></td>
<td><strong>Class 57</strong> — Operator Intervention</td>
<td></td>
</tr>
<tr>
<td>57000</td>
<td>OPERATOR INTERVENTION</td>
<td>operator_intervention</td>
</tr>
<tr>
<td>5714</td>
<td>QUERY CANCELED</td>
<td>query_canceled</td>
</tr>
<tr>
<td>57P01</td>
<td>ADMIN SHUTDOWN</td>
<td>admin_shutdown</td>
</tr>
<tr>
<td>57P02</td>
<td>CRASH SHUTDOWN</td>
<td>crash_shutdown</td>
</tr>
<tr>
<td>57P03</td>
<td>CANNOT CONNECT NOW</td>
<td>cannot_connect_now</td>
</tr>
<tr>
<td></td>
<td><strong>Class 58</strong> — System Error (errors external to Greenplum Database )</td>
<td></td>
</tr>
<tr>
<td>58030</td>
<td>IO ERROR</td>
<td>io_error</td>
</tr>
<tr>
<td>58P01</td>
<td>UNDEFINED FILE</td>
<td>undefined_file</td>
</tr>
<tr>
<td>58P02</td>
<td>DUPLICATE FILE</td>
<td>duplicate_file</td>
</tr>
<tr>
<td></td>
<td><strong>Class F0</strong> — Configuration File Error</td>
<td></td>
</tr>
<tr>
<td>F0000</td>
<td>CONFIG FILE ERROR</td>
<td>config_file_error</td>
</tr>
<tr>
<td>F0001</td>
<td>LOCK FILE EXISTS</td>
<td>lock_file_exists</td>
</tr>
<tr>
<td></td>
<td><strong>Class P0</strong> — PL/pgSQL Error</td>
<td></td>
</tr>
<tr>
<td>P0000</td>
<td>PLPGSQL ERROR</td>
<td>plpgsql_error</td>
</tr>
<tr>
<td>P0001</td>
<td>RAISE EXCEPTION</td>
<td>raise_exception</td>
</tr>
<tr>
<td>P0002</td>
<td>NO DATA FOUND</td>
<td>no_data_found</td>
</tr>
<tr>
<td>P0003</td>
<td>TOO MANY ROWS</td>
<td>too_many_rows</td>
</tr>
<tr>
<td>Error Code</td>
<td>Meaning</td>
<td>Constant</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Class XX   — Internal Error</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XX000</td>
<td>INTERNAL ERROR</td>
<td>internal_error</td>
</tr>
<tr>
<td>XX001</td>
<td>DATA CORRUPTED</td>
<td>data_corrupted</td>
</tr>
<tr>
<td>XX002</td>
<td>INDEX CORRUPTED</td>
<td>index_corrupted</td>
</tr>
</tbody>
</table>

### Routine System Maintenance Tasks

To keep a Greenplum Database system running efficiently, the database must be regularly cleared of expired data and the table statistics must be updated so that the query optimizer has accurate information.

Greenplum Database requires that certain tasks be performed regularly to achieve optimal performance. The tasks discussed here are required, but database administrators can automate them using standard UNIX tools such as `cron` scripts. An administrator sets up the appropriate scripts and checks that they execute successfully. See *Recommended Monitoring and Maintenance Tasks* for additional suggested maintenance activities you can implement to keep your Greenplum system running optimally.

#### Routine Vacuum and Analyze

The design of the MVCC transaction concurrency model used in Greenplum Database means that deleted or updated data rows still occupy physical space on disk even though they are not visible to new transactions. If your database has many updates and deletes, many expired rows exist and the space they use must be reclaimed with the `VACUUM` command. The `VACUUM` command also collects table-level statistics, such as numbers of rows and pages, so it is also necessary to vacuum append-optimized tables, even when there is no space to reclaim from updated or deleted rows.

Vacuuming an append-optimized table follows a different process than vacuuming heap tables. On each segment, a new segment file is created and visible rows are copied into it from the current segment. When the segment file has been copied, the original is scheduled to be dropped and the new segment file is made available. This requires sufficient available disk space for a copy of the visible rows until the original segment file is dropped.

If the ratio of hidden rows to total rows in a segment file is less than a threshold value (10, by default), the segment file is not compacted. The threshold value can be configured with the `gp_appendonly_compaction_threshold` server configuration parameter. `VACUUM FULL` ignores the value of `gp_appendonly_compaction_threshold` and rewrites the segment file regardless of the ratio.

You can use the `_gp_aovisimap_compaction_info()` function in the the `gp_toolkit` schema to investigate the effectiveness of a `VACUUM` operation on append-optimized tables.

For information about the `_gp_aovisimap_compaction_info()` function see, "Checking Append-Optimized Tables" in the *Greenplum Database Reference Guide*.

`VACUUM` can be disabled for append-optimized tables using the `gp_appendonly_compaction` server configuration parameter.

For details about vacuuming a database, see *Vacuuming the Database*.

For information about the `gp_appendonly_compaction_threshold` server configuration parameter and the `VACUUM` command, see the *Greenplum Database Reference Guide*.

#### Transaction ID Management

Greenplum's MVCC transaction semantics depend on comparing transaction ID (XID) numbers to determine visibility to other transactions. Transaction ID numbers are compared using modulo $2^{32}$ arithmetic, so a Greenplum system that runs more than about two billion transactions can experience transaction ID wraparound, where past transactions appear to be in the future. This means past
transactions' outputs become invisible. Therefore, it is necessary to \texttt{VACUUM} every table in every database at least once per two billion transactions.

Greenplum Database assigns XID values only to transactions that involve DDL or DML operations, which are typically the only transactions that require an XID.

\begin{quote}
\textbf{Important:} Greenplum Database monitors transaction IDs. If you do not vacuum the database regularly, Greenplum Database will generate a warning and error.
\end{quote}

Greenplum Database issues the following warning when a significant portion of the transaction IDs are no longer available and before transaction ID wraparound occurs:

\texttt{WARNING: database "database\_name" must be vacuumed within number\_of\_transactions transactions}

When the warning is issued, a \texttt{VACUUM} operation is required. If a \texttt{VACUUM} operation is not performed, Greenplum Database stops creating transactions when it reaches a limit prior to when transaction ID wraparound occurs. Greenplum Database issues this error when it stops creating transactions to avoid possible data loss:

\texttt{FATAL: database is not accepting commands to avoid wraparround data loss in database "database\_name"}

The Greenplum Database configuration parameter \texttt{xid\_warn\_limit} controls when the warning is displayed. The parameter \texttt{xid\_stop\_limit} controls when Greenplum Database stops creating transactions.

\section*{Recovering from a Transaction ID Limit Error}

When Greenplum Database reaches the \texttt{xid\_stop\_limit} transaction ID limit due to infrequent \texttt{VACUUM} maintenance, it becomes unresponsive. To recover from this situation, perform the following steps as database administrator:

1. Shut down Greenplum Database.
2. Temporarily lower the \texttt{xid\_stop\_limit} by 10,000,000.
3. Start Greenplum Database.
4. Run \texttt{VACUUM FREEZE} on all affected databases.
5. Reset the \texttt{xid\_stop\_limit} to its original value.
6. Restart Greenplum Database.

For information about the configuration parameters, see the \textit{Greenplum Database Reference Guide}.

For information about transaction ID wraparound see the \textit{PostgreSQL documentation}.

\section*{System Catalog Maintenance}

Numerous database updates with \texttt{CREATE} and \texttt{DROP} commands increase the system catalog size and affect system performance. For example, running many \texttt{DROP TABLE} statements degrades the overall system performance due to excessive data scanning during metadata operations on catalog tables. The performance loss occurs between thousands to tens of thousands of \texttt{DROP TABLE} statements, depending on the system.

You should run a system catalog maintenance procedure regularly to reclaim the space occupied by deleted objects. If a regular procedure has not been run for a long time, you may need to run a more intensive procedure to clear the system catalog. This topic describes both procedures.

\subsection*{Regular System Catalog Maintenance}

It is recommended that you periodically run \texttt{VACUUM} and \texttt{REINDEX} on the system catalog to clear the space that deleted objects occupy in the system tables and indexes. If regular database operations include
numerous DROP statements, it is safe and appropriate to run a system catalog maintenance procedure with VACUUM daily at off-peak hours. You can do this while the system is available.

The following example script performs a VACUUM, REINDEX, and ANALYZE of the Greenplum Database system catalog:

```bash
#!/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'"
psql -tc "SELECT 'VACUUM' || $SYSTABLES" $DBNAME | psql -a $DBNAME
reindexdb --system -d $DBNAME
analyzedb --system -s pg_catalog -d $DBNAME
```

**Intensive System Catalog Maintenance**

If a system catalog maintenance procedure has not been performed in a long time, the catalog can become bloated with dead space; this causes excessively long wait times for simple metadata operations. A wait of more than two seconds to list user tables, such as with the \d metacommand from within psql, is an indication of catalog bloat.

If you see indications of system catalog bloat, you must perform an intensive system catalog maintenance procedure with VACUUM FULL during a scheduled downtime period. During this period, stop all catalog activity on the system; the FULL system catalog maintenance procedure takes exclusive locks against the system catalog.

Running regular system catalog maintenance procedures can prevent the need for this more costly procedure.

**Vacuum and Analyze for Query Optimization**

Greenplum Database uses a cost-based query optimizer that relies on database statistics. Accurate statistics allow the query optimizer to better estimate selectivity and the number of rows that a query operation retrieves. These estimates help it choose the most efficient query plan. The ANALYZE command collects column-level statistics for the query optimizer.

You can run both VACUUM and ANALYZE operations in the same command. For example:

```bash
=# VACUUM ANALYZE mytable;
```

Running the VACUUM ANALYZE command might produce incorrect statistics when the command is run on a table with a significant amount of bloat (a significant amount of table disk space is occupied by deleted or obsolete rows). For large tables, the ANALYZE command calculates statistics from a random sample of rows. It estimates the number rows in the table by multiplying the average number of rows per page in the sample by the number of actual pages in the table. If the sample contains many empty pages, the estimated row count can be inaccurate.

For a table, you can view information about the amount of unused disk space (space that is occupied by deleted or obsolete rows) in the gp_toolkit view gp_bloat_diag. If the bdidiag column for a table contains the value significant amount of bloat suspected, a significant amount of table disk space consists of unused space. Entries are added to the gp_bloat_diag view after a table has been vacuumed.

To remove unused disk space from the table, you can run the command VACUUM FULL on the table. Due to table lock requirements, VACUUM FULL might not be possible until a maintenance period.

As a temporary workaround, run ANALYZE to compute column statistics and then run VACUUM on the table to generate an accurate row count. This example runs ANALYZE and then VACUUM on the cust_info table.

```bash
ANALYZE cust_info;
VACUUM cust_info;
```
Important: If you intend to execute queries on partitioned tables with GPORCA enabled (the default), you must collect statistics on the partitioned table root partition with the `ANALYZE ROOTPARTITION` command. For information about GPORCA, see Overview of GPORCA.

Note: You can use the Greenplum Database utility `analyzedb` to update table statistics. Tables can be analyzed concurrently. For append optimized tables, `analyzedb` updates statistics only if the statistics are not current. For information about the `analyzedb` utility, see the Greenplum Database Utility Guide.

Routine Reindexing

For B-tree indexes, a freshly-constructed index is slightly faster to access than one that has been updated many times because logically adjacent pages are usually also physically adjacent in a newly built index. Reindexing older indexes periodically can improve access speed. If all but a few index keys on a page have been deleted, there will be wasted space on the index page. A reindex will reclaim that wasted space. In Greenplum Database it is often faster to drop an index (`DROP INDEX`) and then recreate it (`CREATE INDEX`) than it is to use the `REINDEX` command.

For table columns with indexes, some operations such as bulk updates or inserts to the table might perform more slowly because of the updates to the indexes. To enhance performance of bulk operations on tables with indexes, you can drop the indexes, perform the bulk operation, and then re-create the index.

Managing Greenplum Database Log Files

- Database Server Log Files
- Management Utility Log Files

Database Server Log Files

Greenplum Database log output tends to be voluminous, especially at higher debug levels, and you do not need to save it indefinitely. Administrators rotate the log files periodically so new log files are started and old ones are removed.

Greenplum Database has log file rotation enabled on the master and all segment instances. Daily log files are created in the `pg_log` subdirectory of the master and each segment data directory using the following naming convention: `gpdb-YYYY-MM-DD_hhmmss.csv`. Although log files are rolled over daily, they are not automatically truncated or deleted. Administrators need to implement scripts or programs to periodically clean up old log files in the `pg_log` directory of the master and each segment instance.

For information about viewing the database server log files, see Viewing the Database Server Log Files.

Management Utility Log Files

Log files for the Greenplum Database management utilities are written to `~/gpAdminLogs` by default. The naming convention for management log files is:

<table>
<thead>
<tr>
<th>script_name_date.log</th>
</tr>
</thead>
</table>

The log entry format is:

| timestamp:utility:host:user:[INFO|WARN|FATAL]:message |
|---------------------------------------------------------|

The log file for a particular utility execution is appended to its daily log file each time that utility is run.
**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.

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.

**Database State Monitoring Activities**

**Table 29: Database State Monitoring Activities**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Procedure</th>
<th>Corrective Actions</th>
</tr>
</thead>
</table>
| List segments that are currently down. If any rows are returned, this should generate a warning or alert.  
Recommended frequency: run every 5 to 10 minutes  
Severity: IMPORTANT | Run the following query in the postgres database:  
SELECT * FROM gp_segment_configuration  
WHERE status <> 'u'; | If the query returns any rows, follow these steps to correct the problem:  
1. Verify that the hosts with down segments are responsive.  
2. If hosts are OK, check the pg_log files for the primaries and mirrors of the down segments to discover the root cause of the segments going down.  
3. If no unexpected errors are found, run the gprecoverseg utility to bring the segments back online. |
| Check for segments that are currently in change tracking mode. If any rows are returned, this should generate a warning or alert.  
Recommended frequency: run every 5 to 10 minutes  
Severity: IMPORTANT | Execute the following query in the postgres database:  
SELECT * FROM gp_segment_configuration  
WHERE mode = 'c'; | If the query returns any rows, follow these steps to correct the problem:  
1. Verify that hosts with down segments are responsive.  
2. If hosts are OK, check the pg_log 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 gprecoverseg utility to bring the segments back online. |
<table>
<thead>
<tr>
<th>Activity</th>
<th>Procedure</th>
<th>Corrective Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check for segments that are currently re-syncing. If rows are returned, this should generate a warning or alert. Recommended frequency: run every 5 to 10 minutes Severity: IMPORTANT</td>
<td>Execute the following query in the postgres database: <code>SELECT * FROM gp_segment_configuration WHERE mode = 'r';</code></td>
<td>When this query returns rows, it implies that the segments are in the process of being re-synched. If the state does not change from 'r' to 's', then check the pg_log files from the primaries and mirrors of the affected segments for errors.</td>
</tr>
<tr>
<td>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. Recommended frequency: run every 5 to 10 minutes Severity: IMPORTANT</td>
<td>Execute the following query in the postgres database: <code>SELECT * FROM gp_segment_configuration WHERE preferred_role &lt;&gt; role;</code></td>
<td>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.</td>
</tr>
<tr>
<td>Run a distributed query to test that it runs on all segments. One row should be returned for each primary segment. Recommended frequency: run every 5 to 10 minutes Severity: CRITICAL</td>
<td>Execute the following query in the postgres database: <code>SELECT gp_segment_id, count(*) FROM gp_dist_random('pg_class') GROUP BY 1;</code></td>
<td>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.</td>
</tr>
<tr>
<td>Test the state of master mirroring on a Greenplum Database 4.2 or earlier cluster. If the value is &quot;Not Synchronized&quot;, raise an alert or warning. Recommended frequency: run every 5 to 10 minutes Severity: IMPORTANT</td>
<td>Execute the following query in the postgres database: <code>SELECT summary_state FROM gp_master_mirroring;</code></td>
<td>Check the pg_log from the master and standby master for errors. If there are no unexpected errors and the machines are up, run the gpinitstandby utility to bring the standby online. This requires a database restart on GPDB 4.2 and earlier.</td>
</tr>
<tr>
<td>Test the state of master mirroring on Greenplum Database. If the value is not &quot;STREAMING&quot;, raise an alert or warning. Recommended frequency: run every 5 to 10 minutes Severity: IMPORTANT</td>
<td>Run the following psql command: <code>psql dbname -c 'SELECT procpid, state FROM pg_stat_replication;'</code></td>
<td>Check the pg_log file from the master and standby master for errors. If there are no unexpected errors and the machines are up, run the gpinitstandby utility to bring the standby online.</td>
</tr>
</tbody>
</table>
### Activity
Perform a basic check to see if the master is up and functioning.

- **Recommended frequency:** run every 5 to 10 minutes
- **Severity:** CRITICAL

Run the following query in the `postgres` database:

```sql
SELECT count(*) FROM gp_segment_configuration;
```

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 trigger the activation of the standby master.

### Database Alert Log Monitoring

**Table 30: Database Alert Log Monitoring Activities**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Procedure</th>
<th>Corrective Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check for FATAL and ERROR log messages from the system.</td>
<td>Run the following query in the <code>gpperfmon</code> database:</td>
<td>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.</td>
</tr>
</tbody>
</table>
| - **Recommended frequency:** run every 15 minutes | ```sql
SELECT * FROM log_alert_history WHERE logseverity in ('FATAL', 'ERROR') AND logtime > (now() - interval '15 minutes');
``` | |
| - **Severity:** WARNING | | |
| **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.** | | |
| Set up server configuration parameters to send SNMP or email alerts. | Enable server configuration parameters to send alerts via SNMP or email: | DBA takes action based on the nature of the alert. |
| - **Recommended frequency:** N/A. Alerts are generated by the system. | - gp_email_smtp_server  
- gp_email_smtp_userid  
- gp_email_smtp_password  
- gp_snmp_monitor_address  
- gp_snmp_community  
- gp_snmp_use_inform_or_trap | |
| - **Severity:** WARNING | | |
| **This activity and the previous are two methods for monitoring messages in the log_alert_history table. It is only necessary to set up one or the other.** | | |
# Hardware and Operating System Monitoring

## Table 31: Hardware and Operating System Monitoring Activities

<table>
<thead>
<tr>
<th>Activity</th>
<th>Procedure</th>
<th>Corrective Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Underlying platform check for maintenance required or system down of the hardware. Recommended frequency: real-time, if possible, or every 15 minutes Severity: CRITICAL</td>
<td>Set up SNMP or other system check for hardware and OS errors.</td>
<td>If required, remove a machine from the Greenplum cluster to resolve hardware and OS issues, then, after add it back to the cluster and run <code>gprecoverseg</code>.</td>
</tr>
<tr>
<td>Check disk space usage on volumes used for Greenplum Database data storage and the OS. Recommended frequency: every 5 to 30 minutes Severity: CRITICAL</td>
<td>Set up a disk space check. - Set a threshold to raise an alert when a disk reaches a percentage of capacity. The recommended threshold is 75% full. - It is not recommended to run the system with capacities approaching 100%.</td>
<td>Free space on the system by removing some data or files.</td>
</tr>
<tr>
<td>Check for errors or dropped packets on the network interfaces. Recommended frequency: hourly Severity: IMPORTANT</td>
<td>Set up a network interface checks.</td>
<td>Work with network and OS teams to resolve errors.</td>
</tr>
<tr>
<td>Check for RAID errors or degraded RAID performance. Recommended frequency: every 5 minutes Severity: CRITICAL</td>
<td>Set up a RAID check. - Replace failed disks as soon as possible. - Work with system administration team to resolve other RAID or controller errors as soon as possible.</td>
<td></td>
</tr>
<tr>
<td>Run the Greenplum <code>gpcheck</code> utility to test that the configuration of the cluster complies with current recommendations. Recommended frequency: when creating a cluster or adding new machines to the cluster Severity: IMPORTANT</td>
<td>Run <code>gpcheck</code>.</td>
<td>Work with system administration team to update configuration according to the recommendations made by the <code>gpcheck</code> utility.</td>
</tr>
<tr>
<td>Activity</td>
<td>Procedure</td>
<td>Corrective Actions</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>----------------------------------------</td>
<td>------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Check for adequate I/O bandwidth and I/O skew. | Run the Greenplum gpcheckperf utility. | The cluster may be under-specified if data transfer rates are not similar to the following:  
• 2GB per second disk read  
• 1 GB per second disk write  
• 10 Gigabit per second network read and write  
If transfer rates are lower than expected, consult with your data architect regarding performance expectations.  
If the machines on the cluster display an uneven performance profile, work with the system administration team to fix faulty machines. |

### Catalog Monitoring

**Table 32: Catalog Monitoring Activities**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Procedure</th>
<th>Corrective Actions</th>
</tr>
</thead>
</table>
| Run catalog consistency checks to ensure the catalog on each host in the cluster is consistent and in a good state. | Run the Greenplum gpcheckcat utility in each database:  
```bash
gpcheckcat -O
``` | Run repair scripts for any issues detected.        |
| Recommended frequency: weekly  
Severity: IMPORTANT                          |                                        |                                    |
| Run a persistent table catalog check.  
Recommended frequency: monthly  
Severity: CRITICAL                          | During a downtime, with no users on the system, run the Greenplum gpcheckcat utility in each database:  
```bash
gpcheckcat -R persistent
``` | Run repair scripts for any issues detected.        |
| Check for `pg_class` entries that have no corresponding `pg_attribute` entry.  
Recommended frequency: monthly  
Severity: IMPORTANT                          | During a downtime, with no users on the system, run the Greenplum gpcheckcat utility in each database:  
```bash
gpcheckcat -R pgclass
``` | Run the repair scripts for any issues identified. |


### Activity
Check for leaked temporary schema and missing schema definition.
Recommended frequency: monthly
Severity: IMPORTANT

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Corrective Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>During a downtime, with no users on the system, run the Greenplum <code>gpcheckcat</code> utility in each database: <code>gpcheckcat -R namespace</code></td>
<td>Run the repair scripts for any issues identified.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Corrective Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>During a downtime, with no users on the system, run the Greenplum <code>gpcheckcat</code> utility in each database: <code>gpcheckcat -R distribution_policy</code></td>
<td>Run the repair scripts for any issues identified.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Corrective Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>During a downtime, with no users on the system, run the Greenplum <code>gpcheckcat</code> utility in each database: <code>gpcheckcat -R dependency</code></td>
<td>Run the repair scripts for any issues identified.</td>
</tr>
</tbody>
</table>

### Data Maintenance

#### Table 33: Data Maintenance Activities

<table>
<thead>
<tr>
<th>Activity</th>
<th>Procedure</th>
<th>Corrective Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check for missing statistics on tables.</td>
<td>Check the <code>gp_stats_missing</code> view in each database: <code>SELECT * FROM gp_toolkit.gp_stats_missing;</code></td>
<td>Run <code>ANALYZE</code> on tables that are missing statistics.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Activity</th>
<th>Procedure</th>
<th>Corrective Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check for tables that have bloat (dead space) in data files that cannot be recovered by a regular <code>VACUUM</code> command. Recommended frequency: weekly or monthly Severity: WARNING</td>
<td>Check the <code>gp_bloat_diag</code> view in each database: <code>SELECT * FROM gp_toolkit.gp_bloat_diag;</code></td>
<td>Execute a <code>VACUUM FULL</code> statement at a time when users are not accessing the table to remove bloat and compact the data.</td>
</tr>
</tbody>
</table>
# Database Maintenance

## Table 34: Database Maintenance Activities

<table>
<thead>
<tr>
<th>Activity</th>
<th>Procedure</th>
<th>Corrective Actions</th>
</tr>
</thead>
</table>
| Mark deleted rows in heap tables so that the space they occupy can be reused.  
Recommended frequency: daily  
Severity: CRITICAL | Vacuum user tables:  
\[\text{VACUUM <table>;}\] | Vacuum updated tables regularly to prevent bloating. |
| Update table statistics.  
Recommended frequency: after loading data and before executing queries  
Severity: CRITICAL | Analyze user tables. You can use the `analyzedb` management utility:  
\[\text{analyzedb -d <database> -a}\] | Analyze updated tables regularly so that the optimizer can produce efficient query execution plans. |
| Backup the database data.  
Recommended frequency: daily, or as required by your backup plan  
Severity: CRITICAL | Run the `gpcrondump` utility to create a backup of the master and segment databases in parallel. | Best practice is to have a current backup ready in case the database must be restored. |
| Vacuum, reindex, and analyze system catalogs to maintain an efficient catalog.  
Recommended frequency: weekly, or more often if database objects are created and dropped frequently | 1. VACUUM the system tables in each database.  
2. Run `REINDEX SYSTEM` in each database, or use the `reindexdb` command-line utility with the `-s` option:  
\[\text{reindexdb -s <database>}\]  
3. ANALYZE each of the system tables:  
\[\text{analyzedb -s pg_catalog -d <database>}\] | 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 `ANALYZE` after reindexing, because `REINDEX` leaves indexes with no statistics. |
# Patching and Upgrading

## Table 35: Patch and Upgrade Activities

<table>
<thead>
<tr>
<th>Activity</th>
<th>Procedure</th>
<th>Corrective Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ensure any bug fixes or enhancements are applied to the kernel.</td>
<td>Follow the vendor's instructions to update the Linux kernel.</td>
<td>Keep the kernel current to include bug fixes and security fixes, and to avoid difficult future upgrades.</td>
</tr>
<tr>
<td>Recommended frequency: at least every 6 months</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Severity: IMPORTANT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Install Greenplum Database minor releases, for example 5.0.x</td>
<td>Follow upgrade instructions in the Greenplum Database Release Notes. Always upgrade to the latest in the series.</td>
<td>Keep the Greenplum Database software current to incorporate bug fixes, performance enhancements, and feature enhancements into your Greenplum cluster.</td>
</tr>
<tr>
<td>Recommended frequency: quarterly</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Severity: IMPORTANT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Managing Greenplum Database Access

Securing Greenplum Database includes protecting access to the database through network configuration, database user authentication, and encryption.

This section contains the following topics:

• Configuring Client Authentication
  • Using LDAP Authentication with TLS/SSL
  • Using Kerberos Authentication
• Managing Roles and Privileges

Configuring Client Authentication

This topic explains how to configure client connections and authentication for Greenplum Database.

When a Greenplum Database system is first initialized, the system contains one predefined superuser role. This role will have the same name as the operating system user who initialized the Greenplum Database system. This role is referred to as gpadmin. By default, the system is configured to only allow local connections to the database from the gpadmin role. To allow any other roles to connect, or to allow connections from remote hosts, you configure Greenplum Database to allow such connections.

Note: The PgBouncer connection pooler is bundled with Greenplum Database. PgBouncer can be configured to support LDAP or Active Directory authentication for users connecting to Greenplum Database through the connection pooler client connections to Greenplum Database. See Using the PgBouncer Connection Pooler.

Allowing Connections to Greenplum Database

Client access and authentication is controlled by the standard PostgreSQL host-based authentication file, pg_hba.conf. In Greenplum Database, the pg_hba.conf file of the master instance controls client access and authentication to your Greenplum Database system. Greenplum Database segments have pg_hba.conf files that are configured to allow only client connections from the master host and never accept client connections. Do not alter the pg_hba.conf file on your segments.

See The pg_hba.conf File in the PostgreSQL documentation for more information.

The general format of the pg_hba.conf file is a set of records, one per line. Blank lines and any text after the # comment character are ignored. The first matching record is used for authentication. After the first match, the following records are not evaluated. If the client cannot be authenticated using the method specified in the first matching record, the connection is rejected. A record consists of a number of fields that are separated by spaces and/or tabs. Fields can contain white space if the field value is quoted. Records cannot be continued across multiple lines. Each remote client access record has the following format:

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

Each UNIX-domain socket access record has the following format:

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

The following table describes meaning of each field.
Table 36: pg_hba.conf Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>Matches connection attempts using UNIX-domain sockets. Without a record of this type, UNIX-domain socket connections are disallowed.</td>
</tr>
<tr>
<td>host</td>
<td>Matches connection attempts made using TCP/IP. Remote TCP/IP connections will not be possible unless the server is started with an appropriate value for the listen_addresses server configuration parameter.</td>
</tr>
<tr>
<td>hostssl</td>
<td>Matches connection attempts made using TCP/IP, but only when the connection is made with SSL encryption. SSL must be enabled at server start time by setting the ssl configuration parameter.</td>
</tr>
<tr>
<td>hostnossl</td>
<td>Matches connection attempts made over TCP/IP that do not use SSL.</td>
</tr>
<tr>
<td>database</td>
<td>Specifies which database names this record matches. The value all specifies that it matches all databases. Multiple database names can be supplied by separating them with commas. A separate file containing database names can be specified by preceding the file name with @.</td>
</tr>
<tr>
<td>role</td>
<td>Specifies which database role names this record matches. The value all specifies that it matches all roles. If the specified role is a group and you want all members of that group to be included, precede the role name with a +. Multiple role names can be supplied by separating them with commas. A separate file containing role names can be specified by preceding the file name with @.</td>
</tr>
<tr>
<td>CIDR-address</td>
<td>Specifies the client machine IP address range that this record matches. It contains an IP address in standard dotted decimal notation and a CIDR mask length. IP addresses can only be specified numerically, not as domain or host names. The mask length indicates the number of high-order bits of the client IP address that must match. Bits to the right of this must be zero in the given IP address. There must not be any white space between the IP address, the /, and the CIDR mask length. Typical examples of a CIDR-address are 192.0.2.89/32 for a single host, or 192.0.2.0/24 for a small network, or 10.6.0.0/16 for a larger one. To specify a single host, use a CIDR mask of 32 for IPv4 or 128 for IPv6. In a network address, do not omit trailing zeroes.</td>
</tr>
<tr>
<td>IP-address</td>
<td>These fields can be used as an alternative to the CIDR-address notation. Instead of specifying the mask length, the actual mask is specified in a separate column. For example, 255.255.255.255 represents a CIDR mask length of 32. These fields only apply to host, hostssl, and hostnossl records.</td>
</tr>
<tr>
<td>IP-mask</td>
<td></td>
</tr>
<tr>
<td>authentication-method</td>
<td>Specifies the authentication method to use when connecting. Greenplum supports the authentication methods supported by PostgreSQL 9.0.</td>
</tr>
</tbody>
</table>

Editing the pg_hba.conf File

This example shows how to edit the `pg_hba.conf` file of the master to allow remote client access to all databases from all roles using encrypted password authentication.

**Note:** For a more secure system, consider removing all connections that use trust authentication from your master `pg_hba.conf`. Trust authentication means the role is granted access without any authentication, therefore bypassing all security. Replace trust entries with ident authentication if your system has an ident service available.
Editing pg_hba.conf

1. Open the file \$MASTER_DATA_DIRECTORY/pg_hba.conf in a text editor.

2. Add a line to the file for each type of connection you want to allow. Records are read sequentially, so the order of the records is significant. Typically, earlier records will have tight connection match parameters and weaker authentication methods, while later records will have looser match parameters and stronger authentication methods. For example:

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

3. Save and close the file.

4. Reload the pg_hba.conf configuration file for your changes to take effect:

   $ gpstop -u

**Note:** Note that you can also control database access by setting object privileges as described in *Managing Object Privileges*. The pg_hba.conf file just controls who can initiate a database session and how those connections are authenticated.

Limiting Concurrent Connections

Greenplum Database allocates some resources on a per-connection basis, so setting the maximum number of connections allowed is recommended.

To limit the number of active concurrent sessions to your Greenplum Database system, you can configure the `max_connections` server configuration parameter. This is a local parameter, meaning that you must set it in the `postgresql.conf` file of the master, the standby master, and each segment instance (primary and mirror). The value of `max_connections` on segments should be 5-10 times the value on the master.

When you set `max_connections`, you must also set the dependent parameter `max_prepared_transactions`. This value must be at least as large as the value of `max_connections` on the master, and segment instances should be set to the same value as the master.

For example:

- In \$MASTER_DATA_DIRECTORY/postgresql.conf (including standby master):

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

- In \$SEGMENT_DATA_DIRECTORY/postgresql.conf for all segment instances:

```
max_connections=500
max_prepared_transactions=100
```
The following steps set the parameter values with the Greenplum Database utility `gpconfig`. For information about `gpconfig`, see the *Greenplum Database Utility Guide*.

**To change the number of allowed connections**

1. Log into the Greenplum Database master host as the Greenplum Database administrator and source the file `$GPHOME/greenplum_path.sh`.
2. Set the value of the `max_connections` parameter. This `gpconfig` command sets the value on the segments to 1000 and the value on the master to 200.

   ```bash
   $ gpconfig -c max_connections -v 1000 -m 200
   ```

   The value on the segments must be greater than the value on the master. The value of `max_connections` on segments should be 5-10 times the value on the master.

3. Set the value of the `max_prepared_transactions` parameter. This `gpconfig` command sets the value to 200 on the master and all segments.

   ```bash
   $ gpconfig -c max_prepared_transactions -v 200
   ```

   The value of `max_prepared_transactions` must be greater than or equal to `max_connections` on the master.

4. Stop and restart your Greenplum Database system.

   ```bash
   $ gpstop -r
   ```

5. You can check the value of parameters on the master and segments with the `gpconfig -s` option. This `gpconfig` command displays the values of the `max_connections` parameter.

   ```bash
   $ gpconfig -s max_connections
   ```

**Note:**

Raising the values of these parameters may cause Greenplum Database to request more shared memory. To mitigate this effect, consider decreasing other memory-related parameters such as `gp_cached_segworkers_threshold`.

**Encrypting Client/Server Connections**

Enable SSL for client connections to Greenplum Database to encrypt the data passed over the network between the client and the database.

Greenplum Database has native support for SSL connections between the client and the master server. SSL connections prevent third parties from snooping on the packets, and also prevent man-in-the-middle attacks. SSL should be used whenever the client connection goes through an insecure link, and must be used whenever client certificate authentication is used.

To enable SSL requires that OpenSSL be installed on both the client and the master server systems. Greenplum Database can be started with SSL enabled by setting the server configuration parameter `ssl=on` in the `master postgresql.conf`. When starting in SSL mode, the server will look for the files `server.key` (server private key) and `server.crt` (server certificate) in the master data directory. These files must be set up correctly before an SSL-enabled Greenplum Database system can start.

**Important:** Do not protect the private key with a passphrase. The server does not prompt for a passphrase for the private key, and the database startup fails with an error if one is required.

A self-signed certificate can be used for testing, but a certificate signed by a certificate authority (CA) should be used in production, so the client can verify the identity of the server. Either a global or local CA can be used. If all the clients are local to the organization, a local CA is recommended.
Creating a Self-signed Certificate without a Passphrase for Testing Only

To create a quick self-signed certificate for the server for testing, use the following OpenSSL command:

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

Enter the information requested by the prompts. Be sure to enter the local host name as *Common Name*. The challenge password can be left blank.

The program will generate a key that is passphrase protected, and does not accept a passphrase that is less than four characters long.

To use this certificate with Greenplum Database, remove the passphrase with the following commands:

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

Enter the old passphrase when prompted to unlock the existing key.

Then, enter the following command to turn the certificate into a self-signed certificate and to copy the key and certificate to a location where the server will look for them.

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

Finally, change the permissions on the key with the following command. The server will reject the file if the permissions are less restrictive than these.

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

For more details on how to create your server private key and certificate, refer to the *OpenSSL documentation*.

Using LDAP Authentication with TLS/SSL

You can control access to Greenplum Database with an LDAP server and, optionally, secure the connection with encryption by adding parameters to *pg_hba.conf* file entries.

Greenplum Database supports LDAP authentication with the TLS/SSL protocol to encrypt communication with an LDAP server:

- LDAP authentication with STARTTLS and TLS protocol – STARTTLS starts with a clear text connection (no encryption) and upgrades it to a secure connection (with encryption).
- LDAP authentication with a secure connection and TLS/SSL (LDAPS) – Greenplum Database uses the TLS or SSL protocol based on the protocol that is used by the LDAP server.

If no protocol is specified, Greenplum Database communicates with the LDAP server with a clear text connection.

To use LDAP authentication, the Greenplum Database master host must be configured as an LDAP client. See your LDAP documentation for information about configuring LDAP clients.

**Note:** The PgBouncer connection pooler bundled with Greenplum Database is modified to support LDAP or Active Directory authentication for users connecting to Greenplum Database through the connection pooler. See *Setting up LDAP Authentication with PgBouncer* for instructions.
Enabling LDAP Authentication with STARTTLS and TLS

To enable STARTTLS with the TLS protocol, in the pg_hba.conf file, add an ldap line and specify the ldaptls parameter with the value 1. The default port is 389. In this example, the authentication method parameters include the ldaptls parameter.

```
ldap ldapserver=myldap.com ldaptls=1 ldapprefix="uid=" ldapsuffix=",,ou=People,dc=example,dc=com"
```

Specify a non-default port with the ldapport parameter. In this example, the authentication method includes the ldaptls parameter and the ldapport parameter to specify the port 550.

```
ldap ldapserver=myldap.com ldaptls=1 ldapport=500 ldapprefix="uid=" ldapsuffix=",,ou=People,dc=example,dc=com"
```

Enabling LDAP Authentication with a Secure Connection and TLS/SSL

To enable a secure connection with TLS/SSL, add ldaps:// as the prefix to the LDAP server name specified in the ldapserver parameter. The default port is 636.

This example ldapserver parameter specifies a secure connection and the TLS/SSL protocol for the LDAP server myldap.com.

```
ldapserver=ldaps://myldap.com
```

To specify a non-default port, add a colon (:) and the port number after the LDAP server name. This example ldapserver parameter includes the ldaps:// prefix and the non-default port 550.

```
ldapserver=ldaps://myldap.com:550
```

Configuring Authentication with a System-wide OpenLDAP System

If you have a system-wide OpenLDAP system and logins are configured to use LDAP with TLS or SSL in the pg_hba.conf file, logins may fail with the following message:

```
could not start LDAP TLS session: error code '-11'
```

To use an existing OpenLDAP system for authentication, Greenplum Database must be set up to use the LDAP server’s CA certificate to validate user certificates. Follow these steps on both the master and standby hosts to configure Greenplum Database:

1. Copy the base64-encoded root CA chain file from the Active Directory or LDAP server to the Greenplum Database master and standby master hosts. This example uses the directory /etc/pki/tls/certs.
2. Change to the directory where you copied the CA certificate file and, as the root user, generate the hash for OpenLDAP:

   ```
   # cd /etc/pki/tls/certs
   # openssl x509 -noout -hash -in <ca-certificate-file>
   # ln -s <ca-certificate-file> <ca-certificate-file>.0
   ```
3. Configure an OpenLDAP configuration file for Greenplum Database with the CA certificate directory and certificate file specified.

   As the root user, edit the OpenLDAP configuration file /etc/openldap/ldap.conf:

   ```
   SASL_NOCANON on
   URL ldaps://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: For certificate validation to succeed, the hostname in the certificate must match a hostname in the URI property. Otherwise, you must also add TLS_REQCERT allow to the file.

4. As the gpadmin user, edit /usr/local/greenplum-db/greenplum_path.sh and add the following line.

   export LDAPCONF=/etc/openldap/ldap.conf

Notes

Greenplum Database logs an error if the following are specified in an pg_hba.conf file entry:

• If both the ldaps:// prefix and the ldaptls=1 parameter are specified.
• If both the ldaps:// prefix and the ldappasswords parameter are specified.

Enabling encrypted communication for LDAP authentication only encrypts the communication between Greenplum Database and the LDAP server.

See Encrypting Client/Server Connections for information about encrypting client connections.

Examples

These are example entries from an pg_hba.conf file.

This example specifies LDAP authentication with no encryption between Greenplum Database and the LDAP server.

   host all plainuser 0.0.0.0/0 ldap ldapserver=myldap.com ldapprefix="uid=" ldapsuffix="",ou=People,dc=example,dc=com"

This example specifies LDAP authentication with the STARTTLS and TLS protocol between Greenplum Database and the LDAP server.

   host all tlsuser 0.0.0.0/0 ldap ldapserver=myldap.com ldaptls=1 ldapprefix="uid=" ldapsuffix="",ou=People,dc=example,dc=com"

This example specifies LDAP authentication with a secure connection and TLS/SSL protocol between Greenplum Database and the LDAP server.

   host all ldapsuser 0.0.0.0/0 ldap ldaps://myldap.com ldapprefix="uid=" ldapsuffix="",ou=People,dc=example,dc=com"

Using Kerberos Authentication

You can control access to Greenplum Database with a Kerberos authentication server.

Greenplum Database supports the Generic Security Service Application Program Interface (GSSAPI) with Kerberos authentication. GSSAPI provides automatic authentication (single sign-on) for systems that support it. You specify the Greenplum Database users (roles) that require Kerberos authentication in the Greenplum Database configuration file pg_hba.conf. The login fails if Kerberos authentication is not available when a role attempts to log in to Greenplum Database.

Kerberos provides a secure, encrypted authentication service. It does not encrypt data exchanged between the client and database and provides no authorization services. To encrypt data exchanged over the network, you must use an SSL connection. To manage authorization for access to Greenplum databases and objects such as schemas and tables, you use settings in the pg_hba.conf file and privileges given to
Greenplum Database users and roles within the database. For information about managing authorization privileges, see *Managing Roles and Privileges*.

For more information about Kerberos, see [http://web.mit.edu/kerberos/](http://web.mit.edu/kerberos/).

**Requirements for Using Kerberos with Greenplum Database**

The following items are required for using Kerberos with Greenplum Database:

- Kerberos Key Distribution Center (KDC) server using the `krb5-server` library
- Kerberos version 5 `krb5-libs` and `krb5-workstation` packages installed on the Greenplum Database master host
- Greenplum Database version with support for Kerberos
- System time on the Kerberos server and Greenplum Database master host must be synchronized. (Install Linux `ntp` package on both servers.)
- Network connectivity between the Kerberos server and the Greenplum Database master
- Java 1.7.0_17 or later is required to use Kerberos-authenticated JDBC on Red Hat Enterprise Linux 6.x

**Enabling Kerberos Authentication for Greenplum Database**

Complete the following tasks to set up Kerberos authentication with Greenplum Database:

1. Verify your system satisfies the prerequisites for using Kerberos with Greenplum Database. See *Requirements for Using Kerberos with Greenplum Database*.
2. Set up, or identify, a Kerberos Key Distribution Center (KDC) server to use for authentication. See *Install and Configure a Kerberos KDC Server*.
3. In a Kerberos database on the KDC server, set up a Kerberos realm and principals on the server. For Greenplum Database, a principal is a Greenplum Database role that uses Kerberos authentication. In the Kerberos database, a realm groups together Kerberos principals that are Greenplum Database roles.
4. Create Kerberos `keytab` files for Greenplum Database. To access Greenplum Database, you create a service key known only by Kerberos and Greenplum Database. On the Kerberos server, the service key is stored in the Kerberos database. On the Greenplum Database master, the service key is stored in key tables, which are files known as keytabs. The service keys are usually stored in the keytab file `/etc/krb5.keytab`. This service key is the equivalent of the service's password, and must be kept secure. Data that is meant to be read-only by the service is encrypted using this key.
5. Install the Kerberos client packages and the keytab file on Greenplum Database master.
6. Create a Kerberos ticket for `gpadmin` on the Greenplum Database master node using the keytab file. The ticket contains the Kerberos authentication credentials that grant access to the Greenplum Database.

With Kerberos authentication configured on the Greenplum Database, you can use Kerberos for PSQL and JDBC.

- Set up Greenplum Database with Kerberos for PSQL
- Set up Greenplum Database with Kerberos for JDBC

You can also configure external authentication for clients running on a Microsoft Windows system.

- Configure Kerberos authentication for the Greenplum Database client utilities `gpload` and `psql` on a Microsoft Windows system. See *Configure Kerberos on Windows for Greenplum Database Clients*.
- Configure a Microsoft Windows user with a Microsoft Active Directory (AD) account for single-sign-on to a Greenplum Database system. See *Configure Client Authentication with Active Directory*.
Install and Configure a Kerberos KDC Server

Steps to set up a Kerberos Key Distribution Center (KDC) server on a Red Hat Enterprise Linux host for use with Greenplum Database.

Follow these steps to install and configure a Kerberos Key Distribution Center (KDC) server on a Red Hat Enterprise Linux host.

1. Install the Kerberos server packages:

   ```bash
   sudo yum install krb5-libs krb5-server krb5-workstation
   ```

2. Edit the `/etc/krb5.conf` configuration file. The following example shows a Kerberos server with a default `KRB.GREENPLUM.COM` realm.

   ```
   [logging]
   default = FILE:/var/log/krb5libs.log
   kdc = FILE:/var/log/krb5kdc.log
   admin_server = FILE:/var/log/kadmind.log

   [libdefaults]
   default_realm = KRB.GREENPLUM.COM
   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]
   KRB.GREENPLUM.COM = {
     kdc = kerberos-gpdb:88
     admin_server = kerberos-gpdb:749
     default_domain = kerberos-gpdb
   }

   [domain_realm]
   .kerberos-gpdb = KRB.GREENPLUM.COM
   kerberos-gpdb = KRB.GREENPLUM.COM

   [appdefaults]
   pam = {
     debug = false
     ticket_lifetime = 36000
     renew_lifetime = 36000
     forwardable = true
     krb4_convert = false
   }
   ```

   The `kdc` and `admin_server` keys in the `[realms]` section specify the host (`kerberos-gpdb`) and port where the Kerberos server is running. IP numbers can be used in place of host names.

   If your Kerberos server manages authentication for other realms, you would instead add the `KRB.GREENPLUM.COM` realm in the `[realms]` and `[domain_realm]` section of the `krb5.conf` file. See the Kerberos documentation for information about the `krb5.conf` file.

3. To create a Kerberos KDC database, run the `kdb5_util`.

   ```bash
   kdb5_util create -s
   ```
The `kdb5_util create` option creates the database to store keys for the Kerberos realms that are managed by this KDC server. The `-s` option creates a stash file. Without the stash file, every time the KDC server starts it requests a password.

4. Add an administrative user to the KDC database with the `kadmin.local` utility. Because it does not itself depend on Kerberos authentication, the `kadmin.local` utility allows you to add an initial administrative user to the local Kerberos server. To add the user `gpadmin` as an administrative user to the KDC database, run the following command:

```
/kadmin.local -q "addprinc gpadmin/admin"
```

Most users do not need administrative access to the Kerberos server. They can use `kadmin` to manage their own principals (for example, to change their own password). For information about `kadmin`, see the [Kerberos documentation](https://kerberos.org/).

5. If needed, edit the `/var/kerberos/krb5kdc/kadm5.acl` file to grant the appropriate permissions to `gpadmin`.

6. Start the Kerberos daemons:

```
/sbin/service krb5kdc start
/sbin/service kadmin start
```

7. To start Kerberos automatically upon restart:

```
/sbin/chkconfig krb5kdc on
/sbin/chkconfig kadmin on
```

### Create Greenplum Database Roles in the KDC Database

Add principals to the Kerberos realm for Greenplum Database.

Start `kadmin.local` in interactive mode, then add two principals to the Greenplum Database Realm.

1. Start `kadmin.local` in interactive mode:

```
kadmin.local
```

2. Add principals:

```
kadmin.local: addprinc gpadmin/kerberos-gpdb@KRB.EXAMPLE.COM
kadmin.local: addprinc postgres/master.test.com@KRB.EXAMPLE.COM
```

The `addprinc` commands prompt for passwords for each principal. The first `addprinc` creates a Greenplum Database user as a principal, `gpadmin/kerberos-gpdb`. The second `addprinc` command creates the `postgres` process on the Greenplum Database master host as a principal in the Kerberos KDC. This principal is required when using Kerberos authentication with Greenplum Database.

3. Create a Kerberos keytab file with `kadmin.local`. The following example creates a keytab file `gpdb-kerberos.keytab` in the current directory with authentication information for the two principals.

```
kadmin.local: xst -k gpdb-kerberos.keytab
gpadmin/kerberos-gpdb@KRB.EXAMPLE.COM
postgres/master.test.com@KRB.EXAMPLE.COM
```

You will copy this file to the Greenplum Database master host.

4. Exit `kadmin.local` interactive mode with the `quit` command:

```
kadmin.local: quit
```
Install and Configure the Kerberos Client
Steps to install the Kerberos client on the Greenplum Database master host.
Install the Kerberos client libraries on the Greenplum Database master and configure the Kerberos client.

1. Install the Kerberos packages on the Greenplum Database master.
   ```bash
   sudo yum install krb5-libs krb5-workstation
   ```

2. Ensure that the `/etc/krb5.conf` file is the same as the one that is on the Kerberos server.

3. Copy the `gpdb-kerberos.keytab` file that was generated on the Kerberos server to the Greenplum Database master host.

4. Remove any existing tickets with the Kerberos utility `kdestroy`. Run the utility as root.
   ```bash
   sudo kdestroy
   ```

5. Use the Kerberos utility `kinit` to request a ticket using the keytab file on the Greenplum Database master for `gpadmin/kerberos-gpdb@KRB.EXAMPLE.COM`. The `-t` option specifies the keytab file on the Greenplum Database master.
   ```bash
   # kinit -k -t gpdb-kerberos.keytab gpadmin/kerberos-gpdb@KRB.EXAMPLE.COM
   ```

6. Use the Kerberos utility `klist` to display the contents of the Kerberos ticket cache on the Greenplum Database master. The following is an example:
   ```bash
   # klist
   Ticket cache: FILE:/tmp/krb5cc_108061
   Default principal: gpadmin/kerberos-gpdb@KRB.EXAMPLE.COM
   Valid starting     Expires            Service principal
   03/28/13 14:50:26  03/29/13 14:50:26  krbtgt/KRB.GREENPLUM.COM
   @KRB.EXAMPLE.COM
   renew until 03/28/13 14:50:26
   ```

Set up Greenplum Database with Kerberos for PSQL
Configure a Greenplum Database to use Kerberos.

After you have set up Kerberos on the Greenplum Database master, you can configure Greenplum Database to use Kerberos. For information on setting up the Greenplum Database master, see Install and Configure the Kerberos Client.

1. Create a Greenplum Database administrator role in the database `postgres` for the Kerberos principal that is used as the database administrator. The following example uses `gpadmin/kerberos-gpdb`.
   ```bash
   psql postgres -c 'create role "gpadmin/kerberos-gpdb" login superuser;'
   ```
   The role you create in the database `postgres` will be available in any new Greenplum Database that you create.

2. Modify `postgresql.conf` to specify the location of the keytab file. For example, adding this line to the `postgresql.conf` specifies the folder `/home/gpadmin` as the location of the keytab file `gpdb-kerberos.keytab`.
   ```bash
   krb_server_keyfile = '/home/gpadmin/gpdb-kerberos.keytab'
   ```

3. Modify the Greenplum Database file `pg_hba.conf` to enable Kerberos support. Then restart Greenplum Database (`gpstop -ar`). For example, adding the following line to `pg_hba.conf` adds GSSAPI and Kerberos support. The value for `krb_realm` is the Kerberos realm that is used for authentication to Greenplum Database.
   ```bash
   host all all 0.0.0.0/0 gss include_realm=0 krb_realm=KRB.GREENPLUM.COM
   ```
For information about the pg_hba.conf file, see *The pg_hba.conf file* in the Postgres documentation.

4. Create a ticket using `kinit` and show the tickets in the Kerberos ticket cache with `klist`.

5. As a test, log in to the database as the `gpadmin` role with the Kerberos credentials `gpadmin/kerberos-gpdb`:

   ```sql
   psql -U "gpadmin/kerberos-gpdb" -h master.test postgres
   ```

   A username map can be defined in the `pg_ident.conf` file and specified in the `pg_hba.conf` file to simplify logging into Greenplum Database. For example, this `psql` command logs into the default Greenplum Database on `mdw.proddb` as the Kerberos principal `adminuser/mdw.proddb`:

   ```
   $ psql -U "adminuser/mdw.proddb" -h mdw.proddb
   ```

   If the default user is `adminuser`, the `pg_ident.conf` file and the `pg_hba.conf` file can be configured so that the `adminuser` can log in to the database as the Kerberos principal `adminuser/mdw.proddb` without specifying the `-U` option:

   ```
   $ psql -h mdw.proddb
   ```

   The following username map is defined in the Greenplum Database file `$MASTER_DATA_DIRECTORY/pg_ident.conf`:

   ```
   # MAPNAME   SYSTEM-USERNAME        GP-USERNAME
   mymap       /^(.*)mdw\.*$    adminuser
   ```

   The map can be specified in the `pg_hba.conf` file as part of the line that enables Kerberos support:

   ```
   host all all 0.0.0.0/0 krb5 include_realm=0 krb_realm=proddb map=mymap
   ```

   For more information about specifying username maps see *Username maps* in the Postgres documentation.

6. If a Kerberos principal is not a Greenplum Database user, a message similar to the following is displayed from the `psql` command line when the user attempts to log in to the database:

   ```
   psql: krb5_sendauth: Bad response
   ```

   The principal must be added as a Greenplum Database user.

---

**Set up Greenplum Database with Kerberos for JDBC**

Enable Kerberos-authenticated JDBC access to Greenplum Database.

You can configure Greenplum Database to use Kerberos to run user-defined Java functions.

1. Ensure that Kerberos is installed and configured on the Greenplum Database master. See *Install and Configure the Kerberos Client*.

2. Create the file `.java.login.config` in the folder `/home/gpadmin` and add the following text to the file:

   ```
   pgjdbc {
     com.sun.security.auth.module.Krb5LoginModule required
doNotPrompt=true
useTicketCache=true
debog=true
client=true;
};
   ```
3. Create a Java application that connects to Greenplum Database using Kerberos authentication. The following example database connection URL uses a PostgreSQL JDBC driver and specifies parameters for Kerberos authentication:

```
jdbc:postgresql://mdw:5432/mytest?kerberosServerName=postgres
&jaasApplicationName=pgjdbc&user=gpadmin/kerberos-gpdb
```

The parameter names and values specified depend on how the Java application performs Kerberos authentication.

4. Test the Kerberos login by running a sample Java application from Greenplum Database.

### Configuring Kerberos for Linux Clients

You can configure Linux client applications to connect to a Greenplum Database system that is configured to authenticate with Kerberos.

If your JDBC application on RedHat Enterprise Linux uses Kerberos authentication when it connects to your Greenplum Database, your client system must be configured to use Kerberos authentication. If you are not using Kerberos authentication to connect to a Greenplum Database, Kerberos is not needed on your client system.

- **Requirements**
- **Setting Up Client System with Kerberos Authentication**
- **Running a Java Application**

For information about enabling Kerberos authentication with Greenplum Database, see the chapter "Setting Up Kerberos Authentication" in the *Greenplum Database Administrator Guide*.

### Requirements

The following are requirements to connect to a Greenplum Database that is enabled with Kerberos authentication from a client system with a JDBC application.

- **Prerequisites**
- **Required Software on the Client Machine**

#### Prerequisites

- Kerberos must be installed and configured on the Greenplum Database master host.

  **Important:** Greenplum Database must be configured so that a remote user can connect to Greenplum Database with Kerberos authentication. Authorization to access Greenplum Database is controlled by the `pg_hba.conf` file. For details, see "Editing the pg_hba.conf File" in the *Greenplum Database Administration Guide*, and also see the *Greenplum Database Security Configuration Guide*.

- The client system requires the Kerberos configuration file `krb5.conf` from the Greenplum Database master.

- The client system requires a Kerberos keytab file that contains the authentication credentials for the Greenplum Database user that is used to log into the database.

- The client machine must be able to connect to Greenplum Database master host.

  If necessary, add the Greenplum Database master host name and IP address to the system `hosts` file. On Linux systems, the `hosts` file is in `/etc`.

#### Required Software on the Client Machine

- The Kerberos `kinit` utility is required on the client machine. The `kinit` utility is available when you install the Kerberos packages:

  - `krb5-libs`
Note: When you install the Kerberos packages, you can use other Kerberos utilities such as `klist` to display Kerberos ticket information.

Java applications require this additional software:

- **Java JDK**
  
  Java JDK 1.7.0_17 is supported on Red Hat Enterprise Linux 6.x.

- Ensure that JAVA_HOME is set to the installation directory of the supported Java JDK.

### Setting Up Client System with Kerberos Authentication

To connect to Greenplum Database with Kerberos authentication requires a Kerberos ticket. On client systems, tickets are generated from Kerberos keytab files with the `kinit` utility and are stored in a cache file.

1. Install a copy of the Kerberos configuration file `krb5.conf` from the Greenplum Database master. The file is used by the Greenplum Database client software and the Kerberos utilities.

   Install `krb5.conf` in the directory `/etc`.

   If needed, add the parameter `default_ccache_name` to the `[libdefaults]` section of the `krb5.ini` file and specify location of the Kerberos ticket cache file on the client system.

2. Obtain a Kerberos keytab file that contains the authentication credentials for the Greenplum Database user.

3. Run `kinit` specifying the keytab file to create a ticket on the client machine. For this example, the keytab file `gpdb-kerberos.keytab` is in the current directory. The ticket cache file is in the `gpadmin` user home directory.

   ```
   > kinit -k -t gpdb-kerberos.keytab -c /home/gpadmin/cache.txt
   gpadmin/kerberos-gpdb@KRB.EXAMPLE.COM
   ```

### Running psql

From a remote system, you can access a Greenplum Database that has Kerberos authentication enabled.

**To connect to Greenplum Database with psql**

1. As the gpadmin user, open a command window.

2. Start `psql` from the command window and specify a connection to the Greenplum Database specifying the user that is configured with Kerberos authentication.

   The following example logs into the Greenplum Database on the machine `kerberos-gpdb` as the `gpadmin` user with the Kerberos credentials `gpadmin/kerberos-gpdb`:

   ```
   $ psql -U "gpadmin/kerberos-gpdb" -h kerberos-gpdb postgres
   ```

### Running a Java Application

Accessing Greenplum Database from a Java application with Kerberos authentication uses the Java Authentication and Authorization Service (JAAS)

1. Create the file `.java.login.config` in the user home folder.

   For example, on a Linux system, the home folder is similar to `/home/gpadmin`.

   Add the following text to the file:

   ```java
   pgjdbc {
       com.sun.security.auth.module.Krb5LoginModule required
   ```
doNotPrompt=true
useTicketCache=true
ticketCache = "/home/gpadmin/cache.txt"
debug=true
client=true;
}

2. Create a Java application that connects to Greenplum Database using Kerberos authentication and run the application as the user.

This example database connection URL uses a PostgreSQL JDBC driver and specifies parameters for Kerberos authentication.

```sql
jdbc:postgresql://kerberos-gpdb:5432/mytest?
    kerberosServerName=postgres&jaasApplicationName=pgjdbc&
    user=gpadmin/kerberos-gpdb
```

The parameter names and values specified depend on how the Java application performs Kerberos authentication.

**Configuring Kerberos For Windows Clients**

You can configure Microsoft Windows client applications to connect to a Greenplum Database system that is configured to authenticate with Kerberos.

- Configure Kerberos on Windows for Greenplum Database Clients
- Configure Client Authentication with Active Directory

For information about configuring Greenplum Database with Kerberos authentication, see *Using Kerberos Authentication*.

**Configure Kerberos on Windows for Greenplum Database Clients**

When a Greenplum Database system is configured to authenticate with Kerberos, you can configure Kerberos authentication for the Greenplum Database client utilities `gpload` and `psql` on a Microsoft Windows system. The Greenplum Database clients authenticate with Kerberos directly, not with Microsoft Active Directory (AD).

This section contains the following information.

- Installing Kerberos on a Windows System
- Running the psql Utility
- Example gpload YAML File
- Creating a Kerberos Keytab File
- Issues and Possible Solutions

These topics assume that the Greenplum Database system is configured to authenticate with Kerberos and Microsoft Active Directory. See *Configure Client Authentication with Active Directory*.

**Installing Kerberos on a Windows System**

To use Kerberos authentication with the Greenplum Database clients on a Windows system, the MIT Kerberos Windows client must be installed on the system. For the clients you can install MIT Kerberos for Windows 4.0.1 (for krb5) that is available at [http://web.mit.edu/kerberos/dist/index.html](http://web.mit.edu/kerberos/dist/index.html).

On the Windows system, you manage Kerberos tickets with the Kerberos `kinit` utility.

The automatic start up of the Kerberos service is not enabled. The service cannot be used to authenticate with Greenplum Database.

Create a copy of the Kerberos configuration file `/etc/krb5.conf` from the Greenplum Database master and place it in the default Kerberos location on the Windows system C:\ProgramData\MIT
In the file section [libdefaults], remove the location of the Kerberos ticket cache default_ccache_name.

On the Windows system, use the environment variable KRB5CCNAME to specify the location of the Kerberos ticket. The value for the environment variable is a file, not a directory and should be unique to each login on the server.

This is an example configuration file with default_ccache_name removed. Also, the section [logging] is removed.

```plaintext
[libdefaults]
  debug = true
  default_etypes = aes256-cts-hmac-sha1-96
  default_realm = EXAMPLE.LOCAL
  dns_lookups_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
```

When specifying a Kerberos ticket with KRB5CCNAME, you can specify the value in either a local user environment or within a session. These commands set KRB5CCNAME, runs kinit, and runs the batch file to set the environment variables for the Greenplum Database clients.

```plaintext
set KRB5CCNAME=%USERPROFILE%\krb5cache
kinit
"c:\Program Files (x86)\Greenplum\greenplum-clients-<version>\greenplum_clients_path.bat"
```

Running the psql Utility

After installing and configuring Kerberos and the Kerberos ticket on a Windows system, you can run the Greenplum Database command line client psql.

If you get warnings indicating that the Console code page differs from Windows code page, you can run the Windows utility chcp to change the code page. This is an example of the warning and fix.

```plaintext
psql -h prod1.example.local warehouse
psql (8.3.23)
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 (8.3.23)
```
Creating a Kerberos Keytab File

You can create and use a Kerberos keytab file to avoid entering a password at the command line or the listing a password in a script file when connecting to a Greenplum Database system. You can create a keytab file with these utilities:

- Windows Kerberos utility ktpass
- Java JRE keytab utility ktab

If you use AES256-CTS-HMAC-SHA1-96 encryption, you need to download and install the Java extension Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files for JDK/JRE from Oracle. This command creates the keytab file svcPostgresProd1.keytab.

You run the ktpass utility as an AD Domain Administrator. The utility expects a user account to have a Service Principal Name (SPN) defined as an AD user attribute, however, it does not appear to be required. You can specify it as a parameter to ktpass and ignore the warning that it cannot be set.

The Java JRE ktab utility does not require an AD Domain Administrator and does not require an SPN.

**Note:** When you enter the password to create the keytab file, the password is visible on screen.

This example runs the ktpass utility to create the keytab dev1.keytab.

```
ktpass -out dev1.keytab -princ dev1@EXAMPLE.LOCAL -mapUser dev1 -pass your_password -crypto all -ptype KRB5_NT_PRINCIPAL
```

It works despite the warning message Unable to set SPN mapping data.

This example runs the Java ktab.exe to create a keytab file (-a option) and list the keytab name and entries (-l -e -t options).

```
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)
```

You can then use a keytab with the following:

```
kinit -kt dev1.keytab dev1
or
kinit -kt %USERPROFILE%\krb5.keytab dev1
```

Example gpload YAML File

This is an example of running a gpload job with the user dev1 logged onto a Windows desktop with the AD domain.

In the example test.yaml control file, the USER: line has been removed. Kerberos authentication is used.

---
These commands run `kinit` and then `gpload` with the `test.yaml` file and displays successful `gpload` output.

```
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
```

### Issues and Possible Solutions

- This message indicates that Kerberos cannot find your cache file:

```
Credentials cache I/O operation failed XXX
(Kerberos error 193)
krb5_cc_default() failed
```

To ensure that Kerberos can find the file set the environment variable `KR5CCNAME` and run `kinit`.

```
set KR5CCNAME=%USERPROFILE%\krb5cache
kinit
```

- This `kinit` message indicates that the `kinit -k -t` command could not find the keytab.

```
kinit: Generic preauthentication failure while getting initial credentials
```

Confirm the full path and filename for the Kerberos keytab file is correct.

### Configure Client Authentication with Active Directory

You can configure a Microsoft Windows user with a Microsoft Active Directory (AD) account for single sign-on to a Greenplum Database system.
You configure an AD user account to support logging in with Kerberos authentication.

With AD single sign-on, a Windows user can use Active Directory credentials with a Windows client application to log into a Greenplum Database system. For Windows applications that use ODBC, the ODBC driver can use Active Directory credentials to connect to a Greenplum Database system.

**Note:** Greenplum Database clients that run on Windows, like `gpload`, connect with Greenplum Database directly and do not use Active Directory. For information about connecting Greenplum Database clients on Windows to a Greenplum Database system with Kerberos authentication, see [Configure Kerberos on Windows for Greenplum Database Clients](#).

This section contains the following information.

- **Prerequisites**
- **Active Directory Setup**
- **Greenplum Database Setup for Active Directory**
- **Single Sign-On Examples**
- **Issues and Possible Solutions for Active Directory**

### Prerequisites

These items are required enable AD single sign-on to a Greenplum Database system.

- The Greenplum Database system must be configured to support Kerberos authentication. For information about configuring Greenplum Database with Kerberos authentication, see [Configuring Kerberos For Windows Clients](#).
- You must know the fully-qualified domain name (FQDN) of the Greenplum Database master host. Also, the Greenplum Database master host name must have a domain portion. If the system does not have a domain, you must configure the system to use a domain.

  This Linux `hostname` command displays the FQDN.

  ```bash
date
```

- You must confirm that the Greenplum Database system has the same date and time as the Active Directory domain. For example, you could set the Greenplum Database system NTP time source to be an AD Domain Controller, or configure the master host to use the same external time source as the AD Domain Controller.
- To support single sign-on, you configure an AD user account as a Managed Service Account in AD. These are requirements for Kerberos authentication.

  - You need to add the Service Principal Name (SPN) attribute to the user account information because the Kerberos utilities require the information during Kerberos authentication.
  - Also, as Greenplum database has unattended startups, you must also provide the account login details in a Kerberos keytab file.

  **Note:** Setting the SPN and creating the keytab requires AD administrative permissions.

### Active Directory Setup

The AD naming convention should support multiple Greenplum Database systems. In this example, we create a new AD Managed Service Account `svcPostresProd1` for our `prod1` Greenplum Database system master host.

The Active Directory domain is `example.local`.

The fully qualified domain name for the Greenplum Database master host is `prod1.example.local`.

We will add the SPN `postgres/prod1.example.local` to this account. Service accounts for other Greenplum Database systems will all be in the form `postgres/fully.qualified.hostname`. 
In this example, the AD password is set to never expire and cannot be changed by the user. The AD account password is only used when creating the Kerberos keytab file. There is no requirement to provide it to a database administrator.
An AD administrator must add the Service Principal Name attribute to the account from the command line with the Windows `setspn` command. This example command set the SPN attribute value to `postgres/prod1.example.local` for the AD user `svcPostgresProd1`:

```
setspn -A postgres/prod1.example.local svcPostgresProd1
```

You can see the SPN if Advanced Features are set in the Active Directory Users and Computers view. Find `servicePrincipalName` in the Attribute Editor tab and edit it if necessary.
The next step is to create a Kerberos keytab file.

You can select a specific cryptography method if your security requirements require it, but in the absence of that, it is best to get it to work first and then remove any cryptography methods you do not want.

As an AD Domain Administrator, you can list the types of encryption that your AD domain controller supports with this `ktpass` command:

```
ktpass /?
```

As an AD Domain Administrator, you can run the `ktpass` command to create a keytab file. This example command creates the file `svcPostgresProd1.keytab` with this information:

- **ServicePrincipalName (SPN):** `postgres/prod1.example.local@EXAMPLE.LOCAL`
- **AD user:** `svcPostgresProd1`
- **Encryption methods:** ALL available on AD
- **Principal Type:** `KRB5_NT_PRINCIPAL`

```
ktpass -out svcPostgresProd1.keytab -princ postgres/prod1.example.local@EXAMPLE.LOCAL -mapUser svcPostgresProd1 -pass your_password -crypto all -ptype KRB5_NT_PRINCIPAL
```

**Note:** The AD domain `@EXAMPLE.LOCAL` is appended to the SPN.
You copy the keytab file `svcPostgresProd1.keytab` to the Greenplum Database master host.

As an alternative to running `ktpass` as an AD Domain Administrator, you can run the Java `ktab.exe` utility to generate a keytab file if you have the Java JRE installed on your desktop. When you enter the password using either `ktpass` or `ktab.exe`, the password will be visible on the screen as a command line argument.

This example command creates the keytab file `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:** If you use AES256-CTS-HMAC-SHA1-96 encryption, you must download and install the Java extension Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files for JDK/JRE from Oracle.

### Greenplum Database Setup for Active Directory

These instructions assume that the Kerberos workstation utilities `krb5-workstation` are installed on the Greenplum Database master host.

Update `/etc/krb5.conf` with the AD domain name details and the location of an AD domain controller. This is an example configuration.

```
[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
```

Copy the Kerberos keytab file that contains the AD user information to the Greenplum Database master directory. This example copies the `svcPostgresProd1.keytab` that was created in *Active Directory Setup.*

```
my_svcPostgresProd1.keytab $MASTER_DATA_DIRECTORY
chown gpadmin:gpadmin $MASTER_DATA_DIRECTORY/svcPostgresProd1.keytab
chmod 600 $MASTER_DATA_DIRECTORY/svcPostgresProd1.keytab
```
Add this line as the last line in the Greenplum Database `pg_hba.conf` file. This line configures Greenplum Database authentication to use Active Directory for authentication for connection any attempt that is not matched by a previous line.

```
host all all 0.0.0.0/0 gss include_realm=0
```

Update the Greenplum Database `postgresql.conf` file with the location details for the keytab file and the principal name to use. The fully qualified host name and the default realm from `/etc/krb5.conf` forms the full service principal name.

```
krb_server_keyfile = '/data/master/gpseg-1/svcPostgresProd1.keytab'
krb_srvname = 'postgres'
```

Create a database role for the AD user. This example logs into the default database and runs the `CREATE ROLE` command. The user `dev1` was the user specified when creating the keytab file in `Active Directory Setup`.

```
psql
create role dev1 with login superuser;
```

Restart the database to use the updated authentication information:

```
gpstop -a
gpstart
```

**Note:** The Greenplum Database libraries might conflict with the Kerberos workstation utilities such as `kinit`. If you are using these utilities on the Greenplum Database master, you can either run a `gpadmin` shell that does not source the `$GPHOME/greenplum_path.sh` script, or unset the `LD_LIBRARY_PATH` environment variable similar to this example:

```
unset LD_LIBRARY_PATH
kinit
source $GPHOME/greenplum_path.sh
```

Confirm Greenplum Database access with Kerberos authentication:

```
kinit dev1
psql -h prod1.example.local -U dev1
```

**Single Sign-On Examples**

These single sign-on examples that use AD and Kerberos assume that the AD user `dev1` configured for single sign-on is logged into the Windows desktop.

This example configures Aginity Workbench for Greenplum Database. When using single sign-on, you enable Use Integrated Security.
This example configures an ODBC source. When setting up the ODBC source, do not enter a User Name or Password. This DSN can then be used by applications as an ODBC data source.
You can use the DSN `testdata` with an R client. This example configures R to access the DSN.

```r
library("RODBC")
conn <- odbcDriverConnect("testdata")
sql <- "select * from public.data1"
my_data <- sqlQuery(conn,sql)
print(my_data)
```

**Issues and Possible Solutions for Active Directory**

- Kerberos tickets contain a version number that must match the version number for AD.
  
  To display the version number in your keytab file, use the `klist -ket` command. For example:

  ```bash
  klist -ket svcPostgresProd1.keytab
  ```

  To get the corresponding value from AD domain controller, run this command as an AD Administrator:

  ```bash
  kvno postgres/prod1.example.local@EXAMPLE.LOCAL
  ```

- This login error can occur when there is a mismatch between the Windows ID and the Greenplum Database user role ID. This log file entry shows the login error. A user `dev22` is attempting to login from a Windows desktop where the user is logged in as a different Windows user.

  ```plaintext
  PDT,1917,con32,,seg-1,,x1917,sx1, "FATAL","28000","authentication failed for user ""dev22"": valid
  ```
The error can also occur when the user can be authenticated, but does not have a Greenplum Database user role.

Ensure that the user is using the correct Windows ID and that a Greenplum Database user role is configured for the user ID.

• This error can occur when the Kerberos keytab does not contain a matching cryptographic type to a client attempting to connect.

```sql
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
```

The resolution is to add the additional encryption types to the keytab using `ktutil` or recreating the postgres keytab with all crypto systems from AD.

**Managing Roles and Privileges**

The Greenplum Database authorization mechanism stores roles and permissions to access database objects in the database and is administered using SQL statements or command-line utilities.

Greenplum Database manages database access permissions using *roles*. The concept of roles subsumes the concepts of *users* and *groups*. A role can be a database user, a group, or both. Roles can own database objects (for example, tables) and can assign privileges on those objects to other roles to control access to the objects. Roles can be members of other roles, thus a member role can inherit the object privileges of its parent role.

Every Greenplum Database system contains a set of database roles (users and groups). Those roles are separate from the users and groups managed by the operating system on which the server runs. However, for convenience you may want to maintain a relationship between operating system user names and Greenplum Database role names, since many of the client applications use the current operating system user name as the default.

In Greenplum Database, users log in and connect through the master instance, which then verifies their role and access privileges. The master then issues commands to the segment instances behind the scenes as the currently logged in role.

Roles are defined at the system level, meaning they are valid for all databases in the system.

In order to bootstrap the Greenplum Database system, a freshly initialized system always contains one predefined *superuser* role (also referred to as the system user). This role will have the same name as the operating system user that initialized the Greenplum Database system. Customarily, this role is named `gpadmin`. In order to create more roles you first have to connect as this initial role.

**Security Best Practices for Roles and Privileges**

• **Secure the gpadmin system user.** Greenplum requires a UNIX user id to install and initialize the Greenplum Database system. This system user is referred to as `gpadmin` in the Greenplum documentation. This `gpadmin` user is the default database superuser in Greenplum Database, as well as the file system owner of the Greenplum installation and its underlying data files. This default administrator account is fundamental to the design of Greenplum Database. The system cannot run without it, and there is no way to limit the access of this gpadmin user id. Use roles to manage who has access to the database for specific purposes. You should only use the `gpadmin` account for system maintenance tasks such as expansion and upgrade. Anyone who logs on to a Greenplum host as this user id can read, alter or delete any data; including system catalog data and database access rights. Therefore, it is very important to secure the gpadmin user id and only provide access to essential system administrators. Administrators should only log in to Greenplum as `gpadmin` when performing

```sql
until timestamp expired",",",",",",",",",","auth.c",628,
```
certain system maintenance tasks (such as upgrade or expansion). Database users should never log on as gpadmin, and ETL or production workloads should never run as gpadmin.

- **Assign a distinct role to each user that logs in.** For logging and auditing purposes, each user that is allowed to log in to Greenplum Database should be given their own database role. For applications or web services, consider creating a distinct role for each application or service. See Creating New Roles (Users).

- **Use groups to manage access privileges.** See Role Membership.

- **Limit users who have the SUPERUSER role attribute.** Roles that are superusers bypass all access privilege checks in Greenplum Database, as well as resource queuing. Only system administrators should be given superuser rights. See Altering Role Attributes.

### Creating New Roles (Users)

A user-level role is considered to be a database role that can log in to the database and initiate a database session. Therefore, when you create a new user-level role using the `CREATE ROLE` command, you must specify the `LOGIN` privilege. For example:

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

A database role may have a number of attributes that define what sort of tasks that role can perform in the database. You can set these attributes when you create the role, or later using the `ALTER ROLE` command. See Table 37: Role Attributes for a description of the role attributes you can set.

### Altering Role Attributes

A database role may have a number of attributes that define what sort of tasks that role can perform in the database.

#### Table 37: Role Attributes

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUPERUSER</td>
<td>NOSUPERUSER</td>
</tr>
<tr>
<td>CREATEDB</td>
<td>NOCREATEDB</td>
</tr>
<tr>
<td>CREATEROLE</td>
<td>NOCREATEROLE</td>
</tr>
<tr>
<td>INHERIT</td>
<td>NOINHERIT</td>
</tr>
<tr>
<td>LOGIN</td>
<td>NOLOGIN</td>
</tr>
<tr>
<td>CONNECTION LIMIT connlimit</td>
<td>If role can log in, this specifies how many concurrent connections the role can make. -1 (the default) means no limit.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CREATEEXTTABLE</td>
<td>NOCREATEEXTTABLE</td>
</tr>
<tr>
<td>PASSWORD 'password'</td>
<td>Sets the role’s password. If you do not plan to use password authentication you can omit this option. If no password is specified, the password will be set to null and password authentication will always fail for that user. A null password can optionally be written explicitly as PASSWORD NULL.</td>
</tr>
<tr>
<td>ENCRYPTED</td>
<td>UNENCRYPTED</td>
</tr>
<tr>
<td>VALID UNTIL 'timestamp'</td>
<td>Sets a date and time after which the role’s password is no longer valid. If omitted the password will be valid for all time.</td>
</tr>
<tr>
<td>RESOURCE QUEUE queue_name</td>
<td>Assigns the role to the named resource queue for workload management. Any statement that role issues is then subject to the resource queue’s limits. Note that the RESOURCE QUEUE attribute is not inherited; it must be set on each user-level (LOGIN) role.</td>
</tr>
<tr>
<td>DENY {deny_interval</td>
<td>deny_point}</td>
</tr>
</tbody>
</table>

You can set these attributes when you create the role, or later using the ALTER ROLE command. For example:

```sql
=# 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';
```

A role can also have role-specific defaults for many of the server configuration settings. For example, to set the default schema search path for a role:

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

**Role Membership**

It is frequently convenient to group users together to ease management of object privileges: that way, privileges can be granted to, or revoked from, a group as a whole. In Greenplum Database this is done by creating a role that represents the group, and then granting membership in the group role to individual user roles.
Use the `CREATE ROLE` SQL command to create a new group role. For example:

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

Once the group role exists, you can add and remove members (user roles) using the `GRANT` and `REVOKE` commands. For example:

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

For managing object privileges, you would then grant the appropriate permissions to the group-level role only (see Table 38: Object Privileges). The member user roles then inherit the object privileges of the group role. For example:

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

The role attributes `LOGIN`, `SUPERUSER`, `CREATEDB`, `CREATEROLE`, `CREATEEXTTABLE`, and `RESOURCE QUEUE` are never inherited as ordinary privileges on database objects are. User members must actually `SET ROLE` to a specific role having one of these attributes in order to make use of the attribute. In the above example, we gave `CREATEDB` and `CREATEROLE` to the `admin` role. If `sally` is a member of `admin`, she could issue the following command to assume the role attributes of the parent role:

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

### Managing Object Privileges

When an object (table, view, sequence, database, function, language, schema, or tablespace) is created, it is assigned an owner. The owner is normally the role that executed the creation statement. For most kinds of objects, the initial state is that only the owner (or a superuser) can do anything with the object. To allow other roles to use it, privileges must be granted. Greenplum Database supports the following privileges for each object type:

**Table 38: Object Privileges**

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables, Views, Sequences</td>
<td>SELECT, INSERT, UPDATE, DELETE, RULE, ALL</td>
</tr>
<tr>
<td>External Tables</td>
<td>SELECT, RULE, ALL</td>
</tr>
</tbody>
</table>
### Object Type

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Databases</td>
<td>CONNECT, CREATE, TEMPORARY, TEMP, ALL</td>
</tr>
<tr>
<td>Functions</td>
<td>EXECUTE</td>
</tr>
<tr>
<td>Procedural Languages</td>
<td>USAGE</td>
</tr>
<tr>
<td>Schemas</td>
<td>CREATE, USAGE, ALL</td>
</tr>
<tr>
<td>Custom Protocol</td>
<td>SELECT, INSERT, UPDATE, DELETE, RULE, ALL</td>
</tr>
</tbody>
</table>

**Note:** Privileges must be granted for each object individually. For example, granting ALL on a database does not grant full access to the objects within that database. It only grants all of the database-level privileges (CONNECT, CREATE, TEMPORARY) to the database itself.

Use the `GRANT` SQL command to give a specified role privileges on an object. For example:

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

To revoke privileges, use the `REVOKE` command. For example:

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

You can also use the `DROP OWNED` and `REASSIGN OWNED` commands for managing objects owned by deprecated roles (Note: only an object's owner or a superuser can drop an object or reassign ownership). For example:

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

**Simulating Row and Column Level Access Control**

Row-level or column-level access is not supported, nor is labeled security. Row-level and column-level access can be simulated using views to restrict the columns and/or rows that are selected. Row-level labels can be simulated by adding an extra column to the table to store sensitivity information, and then using views to control row-level access based on this column. Roles can then be granted access to the views rather than the base table.

**Encrypting Data**

Greenplum Database includes an optional package of encryption/decryption functions called `pgcrypto`. The `pgcrypto` functions allow database administrators to store certain columns of data in encrypted...
form. This adds an extra layer of protection for sensitive data, as data stored in Greenplum Database in encrypted form cannot be read by anyone who does not have the encryption key, nor can it be read directly from the disks.

Note: The `pgcrypto` functions run inside the database server, which means that all the data and passwords move between `pgcrypto` and the client application in clear-text. For optimal security, consider also using SSL connections between the client and the Greenplum master server.

To use `pgcrypto` functions, run the installation script `$GPHOME/share/postgresql/contrib/pgcrypto.sql` in each database where you want the ability to query other databases:

```
$ psql -d testdb -f $GPHOME/share/postgresql/contrib/pgcrypto.sql
```

See `pgcrypto` in the PostgreSQL documentation for more information about individual functions.

Protecting Passwords in Greenplum Database

In its default configuration, Greenplum Database saves MD5 hashes of login users' passwords in the `pg_authid` system catalog rather than saving clear text passwords. Anyone who is able to view the `pg_authid` table can see hash strings, but no passwords. This also ensures that passwords are obscured when the database is dumped to backup files.

The hash function executes when the password is set by using any of the following commands:

- `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'`

The `ENCRYPTED` keyword may be omitted when the `password_encryption` system configuration parameter is on, which is the default value. The `password_encryption` configuration parameter determines whether clear text or hashed passwords are saved when the `ENCRYPTED` or `UNENCRYPTED` keyword is not present in the command.

Note: The SQL command syntax and `password_encryption` configuration variable include the term `encrypt`, but the passwords are not technically encrypted. They are hashed and therefore cannot be decrypted.

The hash is calculated on the concatenated clear text password and role name. The MD5 hash produces a 32-byte hexadecimal string prefixed with the characters `md5`. The hashed password is saved in the `rolpassword` column of the `pg_authid` system table.

Although it is not recommended, passwords may be saved in clear text in the database by including the `UNENCRYPTED` keyword in the command or by setting the `password_encryption` configuration variable to `off`. Note that changing the configuration value has no effect on existing passwords, only newly created or updated passwords.

To set `password_encryption` globally, execute these commands in a shell as the `gpadmin` user:

```
$ gpconfig -c password_encryption -v 'off'
$ gpstop -u
```

To set `password_encryption` in a session, use the SQL `SET` command:

```
=# SET password_encryption = 'on';
```

Passwords may be hashed using the SHA-256 hash algorithm instead of the default MD5 hash algorithm. The algorithm produces a 64-byte hexadecimal string prefixed with the characters `sha256`.

Note:
Although SHA-256 uses a stronger cryptographic algorithm and produces a longer hash string, it cannot be used with the MD5 authentication method. To use SHA-256 password hashing the authentication method must be set to `password` in the `pg_hba.conf` configuration file so that clear text passwords are sent to Greenplum Database. Because clear text passwords are sent over the network, it is very important to use SSL for client connections when you use SHA-256. The default `md5` authentication method, on the other hand, hashes the password twice before sending it to Greenplum Database, once on the password and role name and then again with a salt value shared between the client and server, so the clear text password is never sent on the network.

To enable SHA-256 hashing, change the `password_hash_algorithm` configuration parameter from its default value, `md5`, to `sha-256`. The parameter can be set either globally or at the session level. To set `password_hash_algorithm` globally, execute these commands in a shell as the `gpadmin` user:

```
$ gpconfig -c password_hash_algorithm -v 'sha-256'
$ gpstop -u
```

To set `password_hash_algorithm` in a session, use the SQL `SET` command:

```
=# SET password_hash_algorithm = 'sha-256';
```

**Time-based Authentication**

Greenplum Database enables the administrator to restrict access to certain times by role. Use the `CREATE ROLE` or `ALTER ROLE` commands to specify time-based constraints.

For details, refer to the *Greenplum Database Security Configuration Guide*. 
Defining Database Objects

This section covers data definition language (DDL) in Greenplum Database and how to create and manage database objects.

Creating objects in a Greenplum Database includes making up-front choices about data distribution, storage options, data loading, and other Greenplum features that will affect the ongoing performance of your database system. Understanding the options that are available and how the database will be used will help you make the right decisions.

Most of the advanced Greenplum features are enabled with extensions to the SQL CREATE DDL statements.

Creating and Managing Databases

A Greenplum Database system is a single instance of Greenplum Database. There can be several separate Greenplum Database systems installed, but usually just one is selected by environment variable settings. See your Greenplum administrator for details.

There can be multiple databases in a Greenplum Database system. This is different from some database management systems (such as Oracle) where the database instance is the database. Although you can create many databases in a Greenplum system, client programs can connect to and access only one database at a time — you cannot cross-query between databases.

About Template Databases

Each new database you create is based on a template. Greenplum provides a default database, template1. Use postgres to connect to Greenplum Database for the first time. Greenplum Database uses template1 to create databases unless you specify another template. Do not create any objects in template1 unless you want those objects to be in every database you create.

Greenplum Database uses another database template, template0, internally. Do not drop or modify template0. You can use template0 to create a completely clean database containing only the standard objects predefined by Greenplum Database at initialization, especially if you modified template1.

Creating a Database

The CREATE DATABASE command creates a new database. For example:

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

To create a database, you must have privileges to create a database or be a Greenplum Database superuser. If you do not have the correct privileges, you cannot create a database. Contact your Greenplum Database administrator to either give you the necessary privilege or to create a database for you.

You can also use the client program createdb to create a database. For example, running the following command in a command line terminal connects to Greenplum Database using the provided host name and port and creates a database named mydatabase:

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

The host name and port must match the host name and port of the installed Greenplum Database system.

Some objects, such as roles, are shared by all the databases in a Greenplum Database system. Other objects, such as tables that you create, are known only in the database in which you create them.

**Warning:** The CREATE DATABASE command is not transactional.
Cloning a Database

By default, a new database is created by cloning the standard system database template, `template1`. Any database can be used as a template when creating a new database, thereby providing the capability to 'clone' or copy an existing database and all objects and data within that database. For example:

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

Creating a Database with a Different Owner

Another database owner can be assigned when a database is created:

```sql
=> CREATE DATABASE new_dbname WITH owner=new_user;
```

Viewing the List of Databases

If you are working in the `psql` client program, you can use the `\l` meta-command to show the list of databases and templates in your Greenplum Database system. If using another client program and you are a superuser, you can query the list of databases from the `pg_database` system catalog table. For example:

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

Altering a Database

The `ALTER DATABASE` command changes database attributes such as owner, name, or default configuration attributes. For example, the following command alters a database by setting its default schema search path (the `search_path` configuration parameter):

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

To alter a database, you must be the owner of the database or a superuser.

Dropping a Database

The `DROP DATABASE` command drops (or deletes) a database. It removes the system catalog entries for the database and deletes the database directory on disk that contains the data. You must be the database owner or a superuser to drop a database, and you cannot drop a database while you or anyone else is connected to it. Connect to `postgres` (or another database) before dropping a database. For example:

```sql
=> \c postgres
=> DROP DATABASE mydatabase;
```

You can also use the client program `dropdb` to drop a database. For example, the following command connects to Greenplum Database using the provided host name and port and drops the database `mydatabase`:

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

**Warning:** Dropping a database cannot be undone.

The `DROP DATABASE` command is not transactional.
Creating and Managing Tablespaces

Tablespaces allow database administrators to have multiple file systems per machine and decide how to best use physical storage to store database objects. They are named locations within a filespace in which you can create objects. Tablespaces allow you to assign different storage for frequently and infrequently used database objects or to control the I/O performance on certain database objects. For example, place frequently-used tables on file systems that use high performance solid-state drives (SSD), and place other tables on standard hard drives.

A tablespace requires a file system location to store its database files. In Greenplum Database, the master and each segment (primary and mirror) require a distinct storage location. The collection of file system locations for all components in a Greenplum system is a filespace. Filespaces can be used by one or more tablespaces.

Creating a Filespace

A filespace sets aside storage for your Greenplum system. A filespace is a symbolic storage identifier that maps onto a set of locations in your Greenplum hosts’ file systems. To create a filespace, prepare the logical file systems on all of your Greenplum hosts, then use the `gpfilespace` utility to define the filespace. You must be a database superuser to create a filespace.

**Note:** Greenplum Database is not directly aware of the file system boundaries on your underlying systems. It stores files in the directories that you tell it to use. You cannot control the location on disk of individual files within a logical file system.

To create a filespace using gpfilespace

1. Log in to the Greenplum Database master as the `gpadmin` user.

   ```sh
   $ su - gpadmin
   ```

2. Create a filespace configuration file:

   ```sh
   $ gpfilespace -o gpfilespace_config
   ```

3. At the prompt, enter a name for the filespace, the primary segment file system locations, the mirror segment file system locations, and a master file system location. Primary and mirror locations refer to directories on segment hosts; the master location refers to a directory on the master host and standby master, if configured. For example, if your configuration has 2 primary and 2 mirror segments per host:

   ```plaintext
   Enter a name for this filespace> fastdisk
   primary location 1> /gpfs1/seg1
   primary location 2> /gpfs1/seg2
   mirror location 1> /gpfs2/mir1
   mirror location 2> /gpfs2/mir2
   master location> /gpfs1/master
   ```

4. `gpfilespace` creates a configuration file. Examine the file to verify that the gpfilespace configuration is correct.

5. Run `gpfilespace` again to create the filespace based on the configuration file:

   ```sh
   $ gpfilespace -c gpfilespace_config
   ```

Moving the Location of Temporary or Transaction Files

You can move temporary or transaction files to a specific filespace to improve database performance when running queries, creating backups, and to store data more sequentially.
The dedicated filespace for temporary and transaction files is tracked in two separate flat files called `gp_temporary_files_filespace` and `gp_transaction_files_filespace`. These are located in the `pg_system` directory on each primary and mirror segment, and on master and standby. You must be a superuser to move temporary or transaction files. Only the `gpfs` utility can write to this file.

### About Temporary and Transaction Files

Unless otherwise specified, temporary and transaction files are stored together with all user data. The default location of temporary files, `<filespace_directory>/<tablespace_oid>/<database_oid>/pgsql_tmp` is changed when you use `gpfs --movetempfiles` for the first time.

Also note the following information about temporary or transaction files:

- You can dedicate only one filespace for temporary or transaction files, although you can use the same filespace to store other types of files.
- You cannot drop a filespace if it used by temporary files.
- You must create the filespace in advance. See [Creating a Filespace](#).

#### To move temporary files using `gpfs`

1. Check that the filespace exists and is different from the filespace used to store all other user data.
2. Issue smart shutdown to bring the Greenplum Database offline.
3. Bring Greenplum Database online with no active session and run the following command:

```bash
gpfs --movetempfilespace filespace_name
```

The location of the temporary files is stored in the segment configuration shared memory (PMMState) and used whenever temporary files are created, opened, or dropped.

#### To move transaction files using `gpfs`

1. Check that the filespace exists and is different from the filespace used to store all other user data.
2. Issue smart shutdown to bring the Greenplum Database offline.
3. Bring Greenplum Database online with no active session and run the following command:

```bash
gpfs --movetransfilespace filespace_name
```

The location of the transaction files is stored in the segment configuration shared memory (PMMState) and used whenever transaction files are created, opened, or dropped.

### Creating a Tablespace

After you create a filespace, use the `CREATE TABLESPACE` command to define a tablespace that uses that filespace. For example:

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

Database superusers define tablespaces and grant access to database users with the `GRANT CREATE` command. For example:

```sql
=# GRANT CREATE ON TABLESPACE fastspace TO admin;
```
Using a Tablespace to Store Database Objects

Users with the `CREATE` privilege on a tablespace can create database objects in that tablespace, such as tables, indexes, and databases. The command is:

```
CREATE TABLE tablename(options) TABLESPACE spacename
```

For example, the following command creates a table in the tablespace `space1`:

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

You can also use the `default_tablespace` parameter to specify the default tablespace for `CREATE TABLE` and `CREATE INDEX` commands that do not specify a tablespace:

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

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. If you do not specify a tablespace when you create a database, the database uses the same tablespace used by its template database.

You can use a tablespace from any database if you have appropriate privileges.

Viewing Existing Tablespaces and Filespaces

Every Greenplum Database system has the following default tablespaces.

- `pg_global` for shared system catalogs.
- `pg_default`, the default tablespace. Used by the `template1` and `template0` databases.

These tablespaces use the system default filespace, `pg_system`, the data directory location created at system initialization.

To see filespace information, look in the `pg_filespace` and `pg_filespace_entry` catalog tables. You can join these tables with `pg_tablespace` to see the full definition of a tablespace. For example:

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

Dropping Tablespaces and Filespaces

To drop a tablespace, you must be the tablespace owner or a superuser. You cannot drop a tablespace until all objects in all databases using the tablespace are removed.

Only a superuser can drop a filespace. A filespace cannot be dropped until all tablespaces using that filespace are removed.

The `DROP TABLESPACE` command removes an empty tablespace.

The `DROP FILESPACE` command removes an empty filespace.

**Note:** You cannot drop a filespace if it stores temporary or transaction files.
Creating and Managing Schemas

Schemas logically organize objects and data in a database. Schemas allow you to have more than one object (such as tables) with the same name in the database without conflict if the objects are in different schemas.

The Default "Public" Schema

Every database has a default schema named public. If you do not create any schemas, objects are created in the public schema. All database roles (users) have CREATE and USAGE privileges in the public schema. When you create a schema, you grant privileges to your users to allow access to the schema.

Creating a Schema

Use the CREATE SCHEMA command to create a new schema. For example:

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

To create or access objects in a schema, write a qualified name consisting of the schema name and table name separated by a period. For example:

```sql
myschema.table
```

See Schema Search Paths for information about accessing a schema.

You can create a schema owned by someone else, for example, to restrict the activities of your users to well-defined namespaces. The syntax is:

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

Schema Search Paths

To specify an object's location in a database, use the schema-qualified name. For example:

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

You can set the search_path configuration parameter to specify the order in which to search the available schemas for objects. The schema listed first in the search path becomes the default schema. If a schema is not specified, objects are created in the default schema.

Setting the Schema Search Path

The search_path configuration parameter sets the schema search order. The ALTER DATABASE command sets the search path. For example:

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

You can also set search_path for a particular role (user) using the ALTER ROLE command. For example:

```sql
=> ALTER ROLE sally SET search_path TO myschema, public, pg_catalog;
```
**Viewing the Current Schema**

Use the `current_schema()` function to view the current schema. For example:

```sql
=> SELECT current_schema();
```

Use the `SHOW` command to view the current search path. For example:

```sql
=> SHOW search_path;
```

**Dropping a Schema**

Use the `DROP SCHEMA` command to drop (delete) a schema. For example:

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

By default, the schema must be empty before you can drop it. To drop a schema and all of its objects (tables, data, functions, and so on) use:

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

**System Schemas**

The following system-level schemas exist in every database:

- **pg_catalog** contains the system catalog tables, built-in data types, functions, and operators. It is always part of the schema search path, even if it is not explicitly named in the search path.
- **information_schema** consists of a standardized set of views that contain information about the objects in the database. These views get system information from the system catalog tables in a standardized way.
- **pg_toast** stores large objects such as records that exceed the page size. This schema is used internally by the Greenplum Database system.
- **pg_bitmapindex** stores bitmap index objects such as lists of values. This schema is used internally by the Greenplum Database system.
- **pg_aoseg** stores append-optimized table objects. This schema is used internally by the Greenplum Database system.
- **gp_toolkit** is an administrative schema that contains external tables, views, and functions that you can access with SQL commands. All database users can access `gp_toolkit` to view and query the system log files and other system metrics.

**Creating and Managing Tables**

Greenplum Database tables are similar to tables in any relational database, except that table rows are distributed across the different segments in the system. When you create a table, you specify the table’s distribution policy.

**Creating a Table**

The `CREATE TABLE` command creates a table and defines its structure. When you create a table, you define:

- The columns of the table and their associated data types. See *Choosing Column Data Types*.
- Any table or column constraints to limit the data that a column or table can contain. See *Setting Table and Column Constraints*. 
• The distribution policy of the table, which determines how Greenplum Database divides data is across the segments. See Choosing the Table Distribution Policy.

• The way the table is stored on disk. See Choosing the Table Storage Model.

• The table partitioning strategy for large tables. See Creating and Managing Databases.

Choosing Column Data Types

The data type of a column determines the types of data values the column can contain. Choose the data type that uses the least possible space but can still accommodate your data and that best constrains the data. For example, use character data types for strings, date or timestamp data types for dates, and numeric data types for numbers.

For table columns that contain textual data, specify the data type VARCHAR or TEXT. Specifying the data type CHAR is not recommended. In Greenplum Database, the data types VARCHAR or TEXT handles padding added to the data (space characters added after the last non-space character) as significant characters, the data type CHAR does not. For information on the character data types, see the CREATE TABLE command in the Greenplum Database Reference Guide.

Use the smallest numeric data type that will accommodate your numeric data and allow for future expansion. For example, using BIGINT for data that fits in INT or SMALLINT wastes storage space. If you expect that your data values will expand over time, consider that changing from a smaller datatype to a larger datatype after loading large amounts of data is costly. For example, if your current data values fit in a SMALLINT but it is likely that the values will expand, INT is the better long-term choice.

Use the same data types for columns that you plan to use in cross-table joins. Cross-table joins usually use the primary key in one table and a foreign key in the other table. When the data types are different, the database must convert one of them so that the data values can be compared correctly, which adds unnecessary overhead.

Greenplum Database has a rich set of native data types available to users. See the Greenplum Database Reference Guide for information about the built-in data types.

Setting Table and Column Constraints

You can define constraints on columns and tables to restrict the data in your tables. Greenplum Database support for constraints is the same as PostgreSQL with some limitations, including:

• CHECK constraints can refer only to the table on which they are defined.

• UNIQUE and PRIMARY KEY constraints must be compatible with their table#s distribution key and partitioning key, if any.

  Note: UNIQUE and PRIMARY KEY constraints are not allowed on append-optimized tables because the UNIQUE indexes that are created by the constraints are not allowed on append-optimized tables.

• FOREIGN KEY constraints are allowed, but not enforced.

• Constraints that you define on partitioned tables apply to the partitioned table as a whole. You cannot define constraints on the individual parts of the table.

Check Constraints

Check constraints allow you to specify that the value in a certain column must satisfy a Boolean (truth-value) expression. For example, to require positive product prices:

```sql
=> CREATE TABLE products
    ( product_no integer,
      name text,
      price numeric CHECK (price > 0) );
```
**Not-Null Constraints**

Not-null constraints specify that a column must not assume the null value. A not-null constraint is always written as a column constraint. For example:

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

**Unique Constraints**

Unique constraints ensure that the data contained in a column or a group of columns is unique with respect to all the rows in the table. The table must be hash-distributed (not DISTRIBUTED RANDOMLY), and the constraint columns must be the same as (or a superset of) the table's distribution key columns. For example:

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

**Primary Keys**

A primary key constraint is a combination of a UNIQUE constraint and a NOT NULL constraint. The table must be hash-distributed (not DISTRIBUTED RANDOMLY), and the primary key columns must be the same as (or a superset of) the table's distribution key columns. If a table has a primary key, this column (or group of columns) is chosen as the distribution key for the table by default. For example:

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

**Foreign Keys**

Foreign keys are not supported. You can declare them, but referential integrity is not enforced.

Foreign key constraints specify that the values in a column or a group of columns must match the values appearing in some row of another table to maintain referential integrity between two related tables. Referential integrity checks cannot be enforced between the distributed table segments of a Greenplum database.

**Choosing the Table Distribution Policy**

All Greenplum Database tables are distributed. When you create or alter a table, you optionally specify DISTRIBUTED BY (hash distribution) or DISTRIBUTED RANDOMLY (round-robin distribution) to determine the table row distribution.

**Note:** The Greenplum Database server configuration parameter `gp_create_table_random_default_distribution` controls the table distribution policy if the DISTRIBUTED BY clause is not specified when you create a table.

For information about the parameter, see "Server Configuration Parameters" of the Greenplum Database Reference Guide.

Consider the following points when deciding on a table distribution policy.
• **Even Data Distribution** — For the best possible performance, all segments should contain equal portions of data. If the data is unbalanced or skewed, the segments with more data must work harder to perform their portion of the query processing. Choose a distribution key that is unique for each record, such as the primary key.

• **Local and Distributed Operations** — Local operations are faster than distributed operations. Query processing is fastest if the work associated with join, sort, or aggregation operations is done locally, at the segment level. Work done at the system level requires distributing tuples across the segments, which is less efficient. When tables share a common distribution key, the work of joining or sorting on their shared distribution key columns is done locally. With a random distribution policy, local join operations are not an option.

• **Even Query Processing** — For best performance, all segments should handle an equal share of the query workload. Query workload can be skewed if a table's data distribution policy and the query predicates are not well matched. For example, suppose that a sales transactions table is distributed on the customer ID column (the distribution key). If a predicate in a query references a single customer ID, the query processing work is concentrated on just one segment.

**Declaring Distribution Keys**

CREATE TABLE's optional clauses DISTRIBUTED BY and DISTRIBUTED RANDOMLY specify the distribution policy for a table. The default is a hash distribution policy that uses either the PRIMARY KEY (if the table has one) or the first column of the table as the distribution key. Columns with geometric or user-defined data types are not eligible as Greenplum distribution key columns. If a table does not have an eligible column, Greenplum distributes the rows randomly or in round-robin fashion.

To ensure even distribution of data, choose a distribution key that is unique for each record. If that is not possible, choose DISTRIBUTED RANDOMLY. For example:

```sql
=> CREATE TABLE products
   (name varchar(40),
    prod_id integer,
    supplier_id integer)
   DISTRIBUTED BY (prod_id);
```

```sql
=> CREATE TABLE random_stuff
   (things text,
    doodads text,
    etc text)
   DISTRIBUTED RANDOMLY;
```

**Important**: A Primary Key is always the Distribution Key for a table. If no Primary Key exists, but a Unique Key exists, this is the Distribution Key for the table.

**Choosing the Table Storage Model**

Greenplum Database supports several storage models and a mix of storage models. When you create a table, you choose how to store its data. This topic explains the options for table storage and how to choose the best storage model for your workload.

• **Heap Storage**
• **Append-Optimized Storage**
• **Choosing Row or Column-Oriented Storage**
• **Using Compression (Append-Optimized Tables Only)**
• **Checking the Compression and Distribution of an Append-Optimized Table**
• **Altering a Table**
• **Dropping a Table**
**Note:** To simplify the creation of database tables, you can specify the default values for some table storage options with the Greenplum Database server configuration parameter `gp_default_storage_options`.

For information about the parameter, see "Server Configuration Parameters" in the Greenplum Database Reference Guide.

## Heap Storage

By default, Greenplum Database uses the same heap storage model as PostgreSQL. Heap table storage works best with OLTP-type workloads where the data is often modified after it is initially loaded. UPDATE and DELETE operations require storing row-level versioning information to ensure reliable database transaction processing. Heap tables are best suited for smaller tables, such as dimension tables, that are often updated after they are initially loaded.

## Append-Optimized Storage

Append-optimized table storage works best with denormalized fact tables in a data warehouse environment. Denormalized fact tables are typically the largest tables in the system. Fact tables are usually loaded in batches and accessed by read-only queries. Moving large fact tables to an append-optimized storage model eliminates the storage overhead of the per-row update visibility information, saving about 20 bytes per row. This allows for a leaner and easier-to-optimize page structure. The storage model of append-optimized tables is optimized for bulk data loading. Single row INSERT statements are not recommended.

### To create a heap table

Row-oriented heap tables are the default storage type.

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

### To create an append-optimized table

Use the WITH clause of the CREATE TABLE command to declare the table storage options. The default is to create the table as a regular row-oriented heap-storage table. For example, to create an append-optimized table with no compression:

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

UPDATE and DELETE are not allowed on append-optimized tables in a serializable transaction and will cause the transaction to abort. CLUSTER, DECLARE...FOR UPDATE, and triggers are not supported with append-optimized tables.

## Choosing Row or Column-Oriented Storage

Greenplum provides a choice of storage orientation models: row, column, or a combination of both. This topic provides general guidelines for choosing the optimum storage orientation for a table. Evaluate performance using your own data and query workloads.

- **Row-oriented storage:** good for OLTP types of workloads with many iterative transactions and many columns of a single row needed all at once, so retrieving is efficient.
- **Column-oriented storage:** good for data warehouse workloads with aggregations of data computed over a small number of columns, or for single columns that require regular updates without modifying other column data.
For most general purpose or mixed workloads, row-oriented storage offers the best combination of flexibility and performance. However, there are use cases where a column-oriented storage model provides more efficient I/O and storage. Consider the following requirements when deciding on the storage orientation model for a table:

- **Updates of table data.** If you load and update the table data frequently, choose a row-oriented heap table. Column-oriented table storage is only available on append-optimized tables. See *Heap Storage* for more information.

- **Frequent INSERTs.** If rows are frequently inserted into the table, consider a row-oriented model. Column-oriented tables are not optimized for write operations, as column values for a row must be written to different places on disk.

- **Number of columns requested in queries.** If you typically request all or the majority of columns in the `SELECT` list or `WHERE` clause of your queries, consider a row-oriented model. Column-oriented tables are best suited to queries that aggregate many values of a single column where the `WHERE` or `HAVING` predicate is also on the aggregate column. For example:

  ```sql
  SELECT SUM(salary) ...
  ```

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

  Or where the `WHERE` predicate is on a single column and returns a relatively small number of rows. For example:

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

- **Number of columns in the table.** Row-oriented storage is more efficient when many columns are required at the same time, or when the row-size of a table is relatively small. Column-oriented tables can offer better query performance on tables with many columns where you access a small subset of columns in your queries.

- **Compression.** Column data has the same data type, so storage size optimizations are available in column-oriented data that are not available in row-oriented data. For example, many compression schemes use the similarity of adjacent data to compress. However, the greater adjacent compression achieved, the more difficult random access can become, as data must be uncompressed to be read.

**To create a column-oriented table**

The `WITH` clause of the `CREATE TABLE` command specifies the table’s storage options. The default is a row-oriented heap table. Tables that use column-oriented storage must be append-optimized tables. For example, to create a column-oriented table:

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

**Using Compression (Append-Optimized Tables Only)**

There are two types of in-database compression available in the Greenplum Database for append-optimized tables:

- **Table-level compression** is applied to an entire table.
- **Column-level compression** is applied to a specific column. You can apply different column-level compression algorithms to different columns.

The following table summarizes the available compression algorithms.
Table 39: Compression Algorithms for Append-Optimized Tables

<table>
<thead>
<tr>
<th>Table Orientation</th>
<th>Available Compression Types</th>
<th>Supported Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row</td>
<td>Table</td>
<td>ZLIB and QUICKLZ (^1)</td>
</tr>
<tr>
<td>Column</td>
<td>Column and Table</td>
<td>RLE_TYPE, ZLIB, and QUICKLZ (^1)</td>
</tr>
</tbody>
</table>

**Note:** \(^1\)QuickLZ compression is not available in the open source version of Greenplum Database.

When choosing a compression type and level for append-optimized tables, consider these factors:

- **CPU usage.** Your segment systems must have the available CPU power to compress and uncompress the data.
- **Compression ratio/disk size.** Minimizing disk size is one factor, but also consider the time and CPU capacity required to compress and scan data. Find the optimal settings for efficiently compressing data without causing excessively long compression times or slow scan rates.
- **Speed of compression.** QuickLZ compression generally uses less CPU capacity and compresses data faster at a lower compression ratio than zlib. zlib provides higher compression ratios at lower speeds.

For example, at compression level 1 (compresslevel=1), QuickLZ and zlib have comparable compression ratios, though at different speeds. Using zlib with compresslevel=6 can significantly increase the compression ratio compared to QuickLZ, though with lower compression speed.

- **Speed of decompression/scan rate.** Performance with compressed append-optimized tables depends on hardware, query tuning settings, and other factors. Perform comparison testing to determine the actual performance in your environment.

**Note:** Do not create compressed append-optimized tables on file systems that use compression. If the file system on which your segment data directory resides is a compressed file system, your append-optimized table must not use compression.

Performance with compressed append-optimized tables depends on hardware, query tuning settings, and other factors. You should perform comparison testing to determine the actual performance in your environment.

**Note:** QuickLZ compression level can only be set to level 1; no other options are available. Compression level with zlib can be set at values from 1 - 9. Compression level with RLE can be set at values from 1 - 4.

An **ENCODING** clause specifies compression type and level for individual columns. When an **ENCODING** clause conflicts with a **WITH** clause, the **ENCODING** clause has higher precedence than the **WITH** clause.

### To create a compressed table

The **WITH** clause of the **CREATE TABLE** command declares the table storage options. Tables that use compression must be append-optimized tables. For example, to create an append-optimized table with zlib compression at a compression level of 5:

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

### Checking the Compression and Distribution of an Append-Optimized Table

Greenplum provides built-in functions to check the compression ratio and the distribution of an append-optimized table. The functions take either the object ID or a table name. You can qualify the table name with a schema name.
Table 40: Functions for compressed append-optimized table metadata

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_ao_distribution(name)</td>
<td></td>
<td>Shows the distribution of an append-optimized table's rows across the array. Returns a set of rows, each of which includes a segment dbid and the number of tuples stored on the segment.</td>
</tr>
<tr>
<td>get_ao_distribution(oid)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>get_ao_compression_ratio(name)</td>
<td>float8</td>
<td>Calculates the compression ratio for a compressed append-optimized table. If information is not available, this function returns a value of -1.</td>
</tr>
<tr>
<td>get_ao_compression_ratio(oid)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The compression ratio is returned as a common ratio. For example, a returned value of 3.19, or 3.19:1, means that the uncompressed table is slightly larger than three times the size of the compressed table.

The distribution of the table is returned as a set of rows that indicate how many tuples are stored on each segment. For example, in a system with four primary segments with dbid values ranging from 0 - 3, the function returns four rows similar to the following:

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

Support for Run-length Encoding

Greenplum Database supports Run-length Encoding (RLE) for column-level compression. RLE data compression stores repeated data as a single data value and a count. For example, in a table with two columns, a date and a description, that contains 200,000 entries containing the value date1 and 400,000 entries containing the value date2, RLE compression for the date field is similar to date1 200000 date2 400000. RLE is not useful with files that do not have large sets of repeated data as it can greatly increase the file size.

There are four levels of RLE compression available. The levels progressively increase the compression ratio, but decrease the compression speed.

Greenplum Database versions 4.2.1 and later support column-oriented RLE compression. To backup a table with RLE compression that you intend to restore to an earlier version of Greenplum Database, alter the table to have no compression or a compression type supported in the earlier version (ZLIB or QUICKlz) before you start the backup operation.

Greenplum Database combines delta compression with RLE compression for data in columns of type BIGINT, INTEGER, DATE, TIME, or TIMESTAMP. The delta compression algorithm is based on the change between consecutive column values and is designed to improve compression when data is loaded in sorted order or when the compression is applied to data in sorted order.

Adding Column-level Compression

You can add the following storage directives to a column for append-optimized tables with column orientation:
- Compression type
- Compression level
- Block size for a column

Add storage directives using the `CREATE TABLE`, `ALTER TABLE`, and `CREATE TYPE` commands.

The following table details the types of storage directives and possible values for each.

### Table 41: Storage Directives for Column-level Compression

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Values</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPRESSTYPE</td>
<td>Type of compression.</td>
<td>zlib: deflate algorithm, quicklz: fast compression, RLE_TYPE: run-length encoding, none: no compression</td>
<td>Values are not case-sensitive.</td>
</tr>
<tr>
<td>COMPRESSLEVEL</td>
<td>Compression level.</td>
<td>zlib compression: 1-9, QuickLZ compression: 1 – use compression</td>
<td>1 is the fastest method with the least compression. 1 is the default. 9 is the slowest method with the most compression.</td>
</tr>
<tr>
<td>BLOCKSIZE</td>
<td>The size in bytes for each block in the table</td>
<td>8192 – 2097152</td>
<td>The value must be a multiple of 8192.</td>
</tr>
</tbody>
</table>

The following is the format for adding storage directives.

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

where the word ENCODING is required and the storage directive has three parts:
- The name of the directive
- An equals sign
- The specification
Separate multiple storage directives with a comma. Apply a storage directive to a single column or designate it as the default for all columns, as shown in the following CREATE TABLE clauses.

**General Usage:**

```
column_name data_type ENCODING ( storage_directive [, ... ] ), ...
```

```
COLUMN column_name ENCODING ( storage_directive [, ... ] ), ...
```

```
DEFAULT COLUMN ENCODING ( storage_directive [, ... ] )
```

**Example:**

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

```
COLUMN C1 ENCODING (compresstype=zlib, compresslevel=6, blocksize=65536)
```

```
DEFAULT COLUMN ENCODING (compresstype=quicklz)
```

**Default Compression Values**

If the compression type, compression level and block size are not defined, the default is no compression, and the block size is set to the Server Configuration Parameter `block_size`.

**Precedence of Compression Settings**

Column compression settings are inherited from the table level to the partition level to the subpartition level. The lowest-level settings have priority.

- Column compression settings specified at the table level override any compression settings for the entire table.
- Column compression settings specified for partitions override any compression settings at the column or table levels.
- Column compression settings specified for subpartitions override any compression settings at the partition, column or table levels.
- When an **ENCODING** clause conflicts with a **WITH** clause, the **ENCODING** clause has higher precedence than the **WITH** clause.

**Note:** The **INHERITS** clause is not allowed in a table that contains a storage directive or a column reference storage directive.

Tables created using the **LIKE** clause ignore storage directive and column reference storage directives.

**Optimal Location for Column Compression Settings**

The best practice is to set the column compression settings at the level where the data resides. See Example 5, which shows a table with a partition depth of 2. **RLE_TYPE** compression is added to a column at the subpartition level.

**Storage Directives Examples**

The following examples show the use of storage directives in **CREATE TABLE** statements.
**Example 1**

In this example, column c1 is compressed using zlib and uses the block size defined by the system. Column c2 is compressed with quicklz, and uses a block size of 65536. Column c3 is not compressed and uses the block size defined by the system.

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

**Example 2**

In this example, column c1 is compressed using zlib and uses the block size defined by the system. Column c2 is compressed with quicklz, and uses a block size of 65536. Column c3 is compressed using RLE_TYPE and uses the block size defined by the system.

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

**Example 3**

In this example, column c1 is compressed using zlib and uses the block size defined by the system. Column c2 is compressed with quicklz, and uses a block size of 65536. Column c3 is compressed using zlib and uses the block size defined by the system. Note that column c3 uses zlib (not RLE_TYPE) in the partitions, because the column storage in the partition clause has precedence over the storage directive in the column definition for the table.

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

**Example 4**

In this example, CREATE TABLE assigns the zlib compressstype storage directive to c1. Column c2 has no storage directive and inherits the compression type (quicklz) and block size (65536) from the DEFAULT COLUMN ENCODING clause.

Column c3's ENCODING clause defines its compression type, RLE_TYPE. The DEFAULT COLUMN ENCODING clause defines c3's block size, 65536.

The ENCODING clause defined for a specific column overrides the DEFAULT ENCODING clause, so column c4 has a compress type of none and the default block size.

```sql
CREATE TABLE T4 (c1 int ENCODING (compresstype=zlib),
    c2 char,
    c4 smallint ENCODING (compresstype=none),
    DEFAULT COLUMN ENCODING (compresstype=quicklz, blocksize=65536),
    COLUMN c3 ENCODING (compresstype=RLE_TYPE)
)
**Example 5**

This example creates an append-optimized, column-oriented table, T5. T5 has two partitions, pl and p2, each of which has subpartitions. Each subpartition has `ENCODING` clauses:

- The `ENCODING` clause for partition pl’s subpartition sp1 defines column i’s compression type as `zlib` and block size as 65536.
- The `ENCODING` clauses for partition p2’s subpartition sp1 defines column i’s compression type as `rle_type` and block size is the default value. Column k uses the default compression and its block size is 8192.

```sql
CREATE TABLE T5(i int, j int, k int, l int)
WITH (appendonly=true, orientation=column)
PARTITION BY range(i) SUBPARTITION BY range(j)
(
  pl start(1) end(2)
  ( subpartition sp1 start(1) end(2)
    column i encoding(compresstype=zlib, blocksize=65536)
  ),
  partition p2 start(2) end(3)
  ( subpartition sp1 start(1) end(2)
    column i encoding(compresstype=rle_type)
    column k encoding(blocksize=8192)
  )
);
```

For an example showing how to add a compressed column to an existing table with the `ALTER TABLE` command, see [Adding a Compressed Column to Table](#).

**Adding Compression in a TYPE Command**

You can define a compression type to simplify column compression statements. For example, the following `CREATE TYPE` command defines a compression type, `comptype`, that specifies `quicklz` compression.

where `comptype` is defined as:

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

You can then use `comptype` in a `CREATE TABLE` command to specify `quicklz` compression for a column:

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

For information about creating and adding compression parameters to a type, see `CREATE TYPE`. For information about changing compression specifications in a type, see `ALTER TYPE`. 
Choosing Block Size

The blocksize is the size, in bytes, for each block in a table. Block sizes must be between 8192 and 2097152 bytes, and be a multiple of 8192. The default is 32768.

Specifying large block sizes can consume large amounts of memory. Block size determines buffering in the storage layer. Greenplum maintains a buffer per partition, and per column in column-oriented tables. Tables with many partitions or columns consume large amounts of memory.

Altering a Table

The `ALTER TABLE` command changes the definition of a table. Use `ALTER TABLE` to change table attributes such as column definitions, distribution policy, storage model, and partition structure (see also Maintaining Partitioned Tables). For example, to add a not-null constraint to a table column:

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

Altering Table Distribution

`ALTER TABLE` provides options to change a table's distribution policy. When the table distribution options change, the table data is redistributed on disk, which can be resource intensive. You can also redistribute table data using the existing distribution policy.

Changing the Distribution Policy

For partitioned tables, changes to the distribution policy apply recursively to the child partitions. This operation preserves the ownership and all other attributes of the table. For example, the following command redistributes the table sales across all segments using the customer_id column as the distribution key:

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

When you change the hash distribution of a table, table data is automatically redistributed. Changing the distribution policy to a random distribution does not cause the data to be redistributed. For example, the following `ALTER TABLE` command has no immediate effect:

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

Redistributing Table Data

To redistribute table data for tables with a random distribution policy (or when the hash distribution policy has not changed) use `REORGANIZE=TRUE`. Reorganizing data may be necessary to correct a data skew problem, or when segment resources are added to the system. For example, the following command redistributes table data across all segments using the current distribution policy, including random distribution.

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

Altering the Table Storage Model

Table storage, compression, and orientation can be declared only at creation. To change the storage model, you must create a table with the correct storage options, load the original table data into the new table, drop the original table, and rename the new table with the original table's name. You must also re-grant any table permissions. For example:

```sql
CREATE TABLE sales2 (LIKE sales)
```
WITH (appendonly=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;

See *Splitting a Partition* to learn how to change the storage model of a partitioned table.

**Adding a Compressed Column to Table**

Use `ALTER TABLE` command to add a compressed column to a table. All of the options and constraints for compressed columns described in *Adding Column-level Compression* apply to columns added with the `ALTER TABLE` command.

The following example shows how to add a column with zlib compression to a table, T1.

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

**Inheritance of Compression Settings**

A partition that is added to a table that has subpartitions with compression settings inherits the compression settings from the subpartition. The following example shows how to create a table with subpartition encodings, then alter it to add a partition.

```
CREATE TABLE ccddl (i int, j int, k int, l int)
  WITH (APPENDONLY = TRUE, ORIENTATION=COLUMN)
  PARTITION BY range(j)
  SUBPARTITION BY list (k)
  SUBPARTITION template{
    SUBPARTITION sp1 values(1, 2, 3, 4, 5),
    COLUMN i ENCODING(COMPRESSTYPE=ZLIB),
    COLUMN j ENCODING(COMPRESSTYPE=QUICKLZ),
    COLUMN k ENCODING(COMPRESSTYPE=ZLIB),
    COLUMN l ENCODING(COMPRESSTYPE=ZLIB))
  (PARTITION p1 START(1) END(10),
   PARTITION p2 START(10) END(20))
;
ALTER TABLE ccddl
  ADD PARTITION p3 START(20) END(30)
;
```

Running the `ALTER TABLE` command creates partitions of table `ccddl` named `ccddl_1_prt_p3` and `ccddl_1_prt_p3_2_prt_sp1`. Partition `ccddl_1_prt_p3` inherits the different compression encodings of subpartition `sp1`.

**Dropping a Table**

The `DROP TABLE` command removes tables from the database. For example:

```
DROP TABLE mytable;
```

To empty a table of rows without removing the table definition, use `DELETE` or `TRUNCATE`. For example:

```
DELETE FROM mytable;
```
TRUNCATE mytable;

DROP TABLE always removes any indexes, rules, triggers, and constraints that exist for the target table. Specify CASCADE to drop a table that is referenced by a view. CASCADE removes dependent views.

**Partitioning Large Tables**

Table partitioning enables supporting very large tables, such as fact tables, by logically dividing them into smaller, more manageable pieces. Partitioned tables can improve query performance by allowing the Greenplum Database query optimizer to scan only the data needed to satisfy a given query instead of scanning all the contents of a large table.

- About Table Partitioning
- Deciding on a Table Partitioning Strategy
- Creating Partitioned Tables
- Loading Partitioned Tables
- Verifying Your Partition Strategy
- Viewing Your Partition Design
- Maintaining Partitioned Tables

**About Table Partitioning**

Partitioning does not change the physical distribution of table data across the segments. Table distribution is physical: Greenplum Database physically divides partitioned tables and non-partitioned tables across segments to enable parallel query processing. Table partitioning is logical: Greenplum Database logically divides big tables to improve query performance and facilitate data warehouse maintenance tasks, such as rolling old data out of the data warehouse.

Greenplum Database supports:

- *range partitioning:* division of data based on a numerical range, such as date or price.
- *list partitioning:* division of data based on a list of values, such as sales territory or product line.
- A combination of both types.

![Figure 24: Example Multi-level Partition Design](image)

**Table Partitioning in Greenplum Database**

Greenplum Database divides tables into parts (also known as partitions) to enable massively parallel processing. Tables are partitioned during `CREATE TABLE` using the `PARTITION BY` (and optionally the
**Subpartitions by)** clause. Partitioning creates a top-level (or parent) table with one or more levels of sub-tables (or child tables). Internally, Greenplum Database creates an inheritance relationship between the top-level table and its underlying partitions, similar to the functionality of the **INHERITS** clause of PostgreSQL.

Greenplum uses the partition criteria defined during table creation to create each partition with a distinct **CHECK** constraint, which limits the data that table can contain. The query optimizer uses **CHECK** constraints to determine which table partitions to scan to satisfy a given query predicate.

The Greenplum system catalog stores partition hierarchy information so that rows inserted into the top-level parent table propagate correctly to the child table partitions. To change the partition design or table structure, alter the parent table using **ALTER TABLE** with the **PARTITION** clause.

To insert data into a partitioned table, you specify the root partitioned table, the table created with the **CREATE TABLE** command. You also can specify a leaf child table of the partitioned table in an **INSERT** command. An error is returned if the data is not valid for the specified leaf child table. Specifying a non-leaf or a non-root partition table in the DML command is not supported.

### Deciding on a Table Partitioning Strategy

Not all tables are good candidates for partitioning. If the answer is **yes** to all or most of the following questions, table partitioning is a viable database design strategy for improving query performance. If the answer is **no** to most of the following questions, table partitioning is not the right solution for that table. Test your design strategy to ensure that query performance improves as expected.

- **Is the table large enough?** Large fact tables are good candidates for table partitioning. If you have millions or billions of records in a table, you may see performance benefits from logically breaking that data up into smaller chunks. For smaller tables with only a few thousand rows or less, the administrative overhead of maintaining the partitions will outweigh any performance benefits you might see.

- **Are you experiencing unsatisfactory performance?** As with any performance tuning initiative, a table should be partitioned only if queries against that table are producing slower response times than desired.

- **Do your query predicates have identifiable access patterns?** Examine the **WHERE** clauses of your query workload and look for table columns that are consistently used to access data. For example, if most of your queries tend to look up records by date, then a monthly or weekly date-partitioning design might be beneficial. Or if you tend to access records by region, consider a list-partitioning design to divide the table by region.

- **Does your data warehouse maintain a window of historical data?** Another consideration for partition design is your organization's business requirements for maintaining historical data. For example, your data warehouse may require that you keep data for the past twelve months. If the data is partitioned by month, you can easily drop the oldest monthly partition from the warehouse and load current data into the most recent monthly partition.

- **Can the data be divided into somewhat equal parts based on some defining criteria?** Choose partitioning criteria that will divide your data as evenly as possible. If the partitions contain a relatively equal number of records, query performance improves based on the number of partitions created. For example, by dividing a large table into 10 partitions, a query will execute 10 times faster than it would against the unpartitioned table, provided that the partitions are designed to support the query's criteria.

Do not create more partitions than are needed. Creating too many partitions can slow down management and maintenance jobs, such as vacuuming, recovering segments, expanding the cluster, checking disk usage, and others.

Partitioning does not improve query performance unless the query optimizer can eliminate partitions based on the query predicates. Queries that scan every partition run slower than if the table were not partitioned, so avoid partitioning if few of your queries achieve partition elimination. Check the explain plan for queries to make sure that partitions are eliminated. See **Query Profiling** for more about partition elimination.

**Warning:** Be very careful with multi-level partitioning because the number of partition files can grow very quickly. For example, if a table is partitioned by both day and city, and there are 1,000
days of data and 1,000 cities, the total number of partitions is one million. Column-oriented tables store each column in a physical table, so if this table has 100 columns, the system would be required to manage 100 million files for the table.

Before settling on a multi-level partitioning strategy, consider a single level partition with bitmap indexes. Indexes slow down data loads, so performance testing with your data and schema is recommended to decide on the best strategy.

Creating Partitioned Tables

You partition tables when you create them with `CREATE TABLE`. This topic provides examples of SQL syntax for creating a table with various partition designs.

To partition a table:

1. Decide on the partition design: date range, numeric range, or list of values.
2. Choose the column(s) on which to partition the table.
3. Decide how many levels of partitions you want. For example, you can create a date range partition table by month and then subpartition the monthly partitions by sales region.

- Defining Date Range Table Partitions
- Defining Numeric Range Table Partitions
- Defining List Table Partitions
- Defining Multi-level Partitions
- Partitioning an Existing Table

Defining Date Range Table Partitions

A date range partitioned table uses a single `date` or `timestamp` column as the partition key column. You can use the same partition key column to create subpartitions if necessary, for example, to partition by month and then subpartition by day. Consider partitioning by the most granular level. For example, for a table partitioned by date, you can partition by day and have 365 daily partitions, rather than partition by year then subpartition by month then subpartition by day. A multi-level design can reduce query planning time, but a flat partition design runs faster.

You can have Greenplum Database automatically generate partitions by giving a `START` value, an `END` value, and an `EVERY` clause that defines the partition increment value. By default, `START` values are always inclusive and `END` values are always exclusive. For example:

```sql
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') );
```

You can also declare and name each partition individually. For example:

```sql
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 ,
```
You do not have to declare an `END` value for each partition, only the last one. In this example, `Jan16` ends where `Feb16` starts.

**Defining Numeric Range Table Partitions**

A numeric range partitioned table uses a single numeric data type column as the partition key column. For example:

```sql
CREATE TABLE rank (id int, rank int, year int, gender char(1), count int)
DISTRIBUTED BY (id)
PARTITION BY RANGE (year)
  DEFAULT PARTITION extra );
```

For more information about default partitions, see [Adding a Default Partition](#).

**Defining List Table Partitions**

A list partitioned table can use any data type column that allows equality comparisons as its partition key column. A list partition can also have a multi-column (composite) partition key, whereas a range partition only allows a single column as the partition key. For list partitions, you must declare a partition specification for every partition (list value) you want to create. For example:

```sql
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:** The current Greenplum Database legacy optimizer allows list partitions with multi-column (composite) partition keys. A range partition only allows a single column as the partition key. The Greenplum Query Optimizer does not support composite keys, so you should not use composite partition keys.

For more information about default partitions, see [Adding a Default Partition](#).

**Defining Multi-level Partitions**

You can create a multi-level partition design with subpartitions of partitions. Using a `subpartition template` ensures that every partition has the same subpartition design, including partitions that you add later. For example, the following SQL creates the two-level partition design shown in [Figure 24: Example Multi-level Partition Design](#):

```sql
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)
```
The following example shows a three-level partition design where the sales table is partitioned by year, then month, then region. The SUBPARTITION TEMPLATE clauses ensure that each yearly partition has the same subpartition structure. The example declares a DEFAULT partition at each level of the hierarchy.

```
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 usa VALUES ('usa'),
            SUBPARTITION europe VALUES ('europe'),
            SUBPARTITION asia VALUES ('asia'),
            DEFAULT SUBPARTITION other_regions )
        DEFAULT PARTITION outlying_years );
```

Caution: When you create multi-level partitions on ranges, it is easy to create a large number of subpartitions, some containing little or no data. This can add many entries to the system tables, which increases the time and memory required to optimize and execute queries. Increase the range interval or choose a different partitioning strategy to reduce the number of subpartitions created.

Partitioning an Existing Table

Tables can be partitioned only at creation. If you have a table that you want to partition, you must create a partitioned table, load the data from the original table into the new table, drop the original table, and rename the partitioned table with the original table's name. You must also re-grant any table permissions. For example:

```
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;
```

Limitations of Partitioned Tables

For each partition level, a partitioned table can have a maximum of 32,767 partitions.

A primary key or unique constraint on a partitioned table must contain all the partitioning columns. A unique index can omit the partitioning columns; however, it is enforced only on the parts of the partitioned table, not on the partitioned table as a whole.

GPORCA, the Greenplum next generation query optimizer, supports uniform multi-level partitioned tables. If GPORCA is enabled (the default) and the multi-level partitioned table is not uniform, Greenplum Database executes queries against the table with the legacy query optimizer. For information about uniform multi-level partitioned tables, see About Uniform Multi-level Partitioned Tables.
Exchanging a leaf child partition with an external table is not supported if the partitioned table is created with the `SUBPARTITION` clause or if a partition has a subpartition. For information about exchanging a leaf child partition with an external table, see *Exchanging a Leaf Child Partition with an External Table*.

These are limitations for partitioned tables when a leaf child partition of the table is an external table:

- Queries that run against partitioned tables that contain external table partitions are executed with the legacy query optimizer.
- The external table partition is a read only external table. Commands that attempt to access or modify data in the external table partition return an error. For example:
  - `INSERT`, `DELETE`, and `UPDATE` commands that attempt to change data in the external table partition return an error.
  - `TRUNCATE` commands return an error.
  - `COPY` commands cannot copy data to a partitioned table that updates an external table partition.
  - `COPY` commands that attempt to copy from an external table partition return an error unless you specify the `IGNORE EXTERNAL PARTITIONS` clause with `COPY` command. If you specify the clause, data is not copied from external table partitions.

To use the `COPY` command against a partitioned table with a leaf child table that is an external table, use an SQL query to copy the data. For example, if the table `my_sales` contains a with a leaf child table that is an external table, this command sends the data to `stdout`:

```
COPY (SELECT * from my_sales ) TO stdout
```

- `VACUUM` commands skip external table partitions.
- The following operations are supported if no data is changed on the external table partition. Otherwise, an error is returned.
  - Adding or dropping a column.
  - Changing the data type of column.
- These `ALTER PARTITION` operations are not supported if the partitioned table contains an external table partition:
  - 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.
- The Greenplum Database utility `gpcrondump` does not back up data from a leaf child partition of a partitioned table if the leaf child partition is a readable external table.

### Loading Partitioned Tables

After you create the partitioned table structure, top-level parent tables are empty. Data is routed to the bottom-level child table partitions. In a multi-level partition design, only the subpartitions at the bottom of the hierarchy can contain data.

Rows that cannot be mapped to a child table partition are rejected and the load fails. To avoid unmapped rows being rejected at load time, define your partition hierarchy with a `DEFAULT` partition. Any rows that do not match a partition's `CHECK` constraints load into the `DEFAULT` partition. See *Adding a Default Partition*.

At runtime, the query optimizer scans the entire table inheritance hierarchy and uses the `CHECK` table constraints to determine which of the child table partitions to scan to satisfy the query's conditions. The `DEFAULT` partition (if your hierarchy has one) is always scanned. `DEFAULT` partitions that contain data slow down the overall scan time.
When you use COPY or INSERT to load data into a parent table, the data is automatically rerouted to the correct partition, just like a regular table.

Best practice for loading data into partitioned tables is to create an intermediate staging table, load it, and then exchange it into your partition design. See Exchanging a Partition.

**Verifying Your Partition Strategy**

When a table is partitioned based on the query predicate, you can use EXPLAIN to verify that the query optimizer scans only the relevant data to examine the query plan.

For example, suppose a sales table is date-range partitioned by month and subpartitioned by region as shown in Figure 24: Example Multi-level Partition Design. For the following query:

```
EXPLAIN SELECT * FROM sales WHERE date='01-07-12' AND region='usa';
```

The query plan for this query should show a table scan of only the following tables:

- the default partition returning 0-1 rows (if your partition design has one)
- the January 2012 partition (sales_1_prt_1) returning 0-1 rows
- the USA region subpartition (sales_1_2_prt_usa) returning some number of rows.

The following example shows the relevant portion of the query plan.

```
->  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)
```

Ensure that the query optimizer does not scan unnecessary partitions or subpartitions (for example, scans of months or regions not specified in the query predicate), and that scans of the top-level tables return 0-1 rows.

**Troubleshooting Selective Partition Scanning**

The following limitations can result in a query plan that shows a non-selective scan of your partition hierarchy.

- The query optimizer can selectively scan partitioned tables only when the query contains a direct and simple restriction of the table using immutable operators such as:
  
  `=, < , <= , >, >= , and <>`

- Selective scanning recognizes STABLE and IMMUTABLE functions, but does not recognize VOLATILE functions within a query. For example, WHERE clauses such as `date > CURRENT_DATE` cause the query optimizer to selectively scan partitioned tables, but `time > TIMEOFDAY` does not.

**Viewing Your Partition Design**

You can look up information about your partition design using the `pg_partitions` view. For example, to see the partition design of the sales table:

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

The following table and views show information about partitioned tables.
Maintaining Partitioned Tables

To maintain a partitioned table, use the `ALTER TABLE` command against the top-level parent table. The most common scenario is to drop old partitions and add new ones to maintain a rolling window of data in a range partition design. You can convert (exchange) older partitions to the append-optimized compressed storage format to save space. If you have a default partition in your partition design, you add a partition by splitting the default partition.

- Adding a Partition
- Renaming a Partition
- Adding a Default Partition
- Dropping a Partition
- Truncating a Partition
- Exchanging a Partition
- Splitting a Partition
- Modifying a Subpartition Template
- Exchanging a Leaf Child Partition with an External Table

**Important:** When defining and altering partition designs, use the given partition name, not the table object name. Although you can query and load any table (including partitioned tables) directly using SQL commands, you can only modify the structure of a partitioned table using the `ALTER TABLE...PARTITION` clauses.

Partitions are not required to have names. If a partition does not have a name, use one of the following expressions to specify a part: `PARTITION FOR (value)` or `PARTITION FOR(RANK(number))`.

### Adding a Partition

You can add a partition to a partition design with the `ALTER TABLE` command. If the original partition design included subpartitions defined by a `subpartition template`, the newly added partition is subpartitioned according to that template. For example:

```
ALTER TABLE sales ADD PARTITION
  START (date '2017-02-01') INCLUSIVE
  END (date '2017-03-01') EXCLUSIVE;
```

If you did not use a subpartition template when you created the table, you define subpartitions when adding a partition:

```
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') );
```

When you add a subpartition to an existing partition, you can specify the partition to alter. For example:

```
ALTER TABLE sales ALTER PARTITION FOR (RANK(12))
  ADD PARTITION africa VALUES ('africa');
```
Note: You cannot add a partition to a partition design that has a default partition. You must split the default partition to add a partition. See Splitting a Partition.

**Renaming a Partition**

Partitioned tables use the following naming convention. Partitioned subtable names are subject to uniqueness requirements and length limitations.

\[ <\text{parentname}>_<\text{level}>_prt_<\text{partition_name}> \]

For example:

```
sales_1_prt_jan16
```

For auto-generated range partitions, where a number is assigned when no name is given):

```
sales_1_prt_1
```

To rename a partitioned child table, rename the top-level parent table. The \(<\text{parentname}>\) changes in the table names of all associated child table partitions. For example, the following command:

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

Changes the associated table names:

```
globalsales_1_prt_1
```

You can change the name of a partition to make it easier to identify. For example:

```
ALTER TABLE sales RENAME PARTITION FOR ('2016-01-01') TO jan16;
```

Changes the associated table name as follows:

```
sales_1_prt_jan16
```

When altering partitioned tables with the `ALTER TABLE` command, always refer to the tables by their partition name (`jan16`) and not their full table name (``sales_1_prt_jan16``).

Note: The table name cannot be a partition name in an `ALTER TABLE` statement. For example, `ALTER TABLE sales...` is correct, `ALTER TABLE sales_1_part_jan16...` is not allowed.

**Adding a Default Partition**

You can add a default partition to a partition design with the `ALTER TABLE` command.

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

If your partition design is multi-level, each level in the hierarchy must have a default partition. For example:

```
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;
```
If incoming data does not match a partition's CHECK constraint and there is no default partition, the data is rejected. Default partitions ensure that incoming data that does not match a partition is inserted into the default partition.

**Dropping a Partition**

You can drop a partition from your partition design using the `ALTER TABLE` command. When you drop a partition that has subpartitions, the subpartitions (and all data in them) are automatically dropped as well. For range partitions, it is common to drop the older partitions from the range as old data is rolled out of the data warehouse. For example:

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

**Truncating a Partition**

You can truncate a partition using the `ALTER TABLE` command. When you truncate a partition that has subpartitions, the subpartitions are automatically truncated as well.

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

**Exchanging a Partition**

You can exchange a partition using the `ALTER TABLE` command. Exchanging a partition swaps one table in place of an existing partition. You can exchange partitions only at the lowest level of your partition hierarchy (only partitions that contain data can be exchanged).

Partition exchange can be useful for data loading. For example, load a staging table and swap the loaded table into your partition design. You can use partition exchange to change the storage type of older partitions to append-optimized tables. For example:

```
CREATE TABLE jan12 (LIKE sales) WITH (appendonly=true);
INSERT INTO jan12 SELECT * FROM sales_1_prt_1 ;
ALTER TABLE sales EXCHANGE PARTITION FOR (DATE '2012-01-01')
WITH TABLE jan12;
```

**Note:** This example refers to the single-level definition of the table `sales`, before partitions were added and altered in the previous examples.

**Warning:** If you specify the WITHOUT VALIDATION clause, you must ensure that the data in table that you are exchanging for an existing partition is valid against the constraints on the partition. Otherwise, queries against the partitioned table might return incorrect results.

The Greenplum Database server configuration parameter `gp_enable_exchange_default_partition` controls availability of the EXCHANGE DEFAULT PARTITION clause. The default value for the parameter is off, the clause is not available and Greenplum Database returns an error if the clause is specified in an ALTER TABLE command.

For information about the parameter, see “Server Configuration Parameters” in the Greenplum Database Reference Guide.

**Warning:** Before you exchange the default partition, you must ensure the data in the table to be exchanged, the new default partition, is valid for the default partition. For example, the data in the new default partition must not contain data that would be valid in other leaf child partitions of the partitioned table. Otherwise, queries against the partitioned table with the exchanged default partition that are executed by GPORCA might return incorrect results.
**Splitting a Partition**

Splitting a partition divides a partition into two partitions. You can split a partition using the `ALTER TABLE` command. You can split partitions only at the lowest level of your partition hierarchy (partitions that contain data) and only range partitions can be split, not list partitions. The split value you specify goes into the latter partition.

For example, to split a monthly partition into two with the first partition containing dates January 1-15 and the second partition containing dates January 16-31:

```sql
ALTER TABLE sales SPLIT PARTITION FOR ('2017-01-01')
AT ('2017-01-16')
INTO (PARTITION jan171to15, PARTITION jan1716to31);
```

If your partition design has a default partition, you must split the default partition to add a partition.

When using the `INTO` clause, specify the current default partition as the second partition name. For example, to split a default range partition to add a new monthly partition for January 2017:

```sql
ALTER TABLE sales SPLIT DEFAULT PARTITION
START ('2017-01-01') INCLUSIVE
END ('2017-02-01') EXCLUSIVE
INTO (PARTITION jan17, default partition);
```

**Modifying a Subpartition Template**

Use `ALTER TABLE SET SUBPARTITION TEMPLATE` to modify the subpartition template of a partitioned table. Partitions added after you set a new subpartition template have the new partition design. Existing partitions are not modified.

The following example alters the subpartition template of this partitioned table:

```sql
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') );
```

This `ALTER TABLE` command, modifies the subpartition template.

```sql
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 );
```

When you add a date-range partition of the table sales, it includes the new regional list subpartition for Africa. For example, the following command creates the subpartitions `usa`, `asia`, `europe`, `africa`, and a default partition named `other`:

```sql
ALTER TABLE sales ADD PARTITION "4"
```
To view the tables created for the partitioned table sales, you can use the command \dt sales* from the psql command line.

To remove a subpartition template, use SET SUBPARTITION TEMPLATE with empty parentheses. For example, to clear the sales table subpartition template:

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

### Exchanging a Leaf Child Partition with an External Table

You can exchange a leaf child partition of a partitioned table with a readable external table. The external table data can reside on a host file system, an NFS mount, or a Hadoop file system (HDFS).

For example, if you have a partitioned table that is created with monthly partitions and most of the queries against the table only access the newer data, you can copy the older, less accessed data to external tables and exchange older partitions with the external tables. For queries that only access the newer data, you could create queries that use partition elimination to prevent scanning the older, unneeded partitions.

Exchanging a leaf child partition with an external table is not supported in these cases:

- The partitioned table is created with the SUBPARTITION clause or if a partition has a subpartition.
- The partitioned table contains a column with a check constraint or a NOT NULL constraint.

For information about exchanging and altering a leaf child partition, see the ALTER TABLE command in the Greenplum Database Command Reference.

For information about limitations of partitioned tables that contain a external table partition, see Limitations of Partitioned Tables.

### Example Exchanging a Partition with an External Table

This is a simple example that exchanges a leaf child partition of this partitioned table for an external table. The partitioned table contains data for the years 2010 through 2013.

```
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) ) ;
```

There are four leaf child partitions for the partitioned table. Each leaf child partition contains the data for a single year. The leaf child partition table sales_1_prt_yr_1 contains the data for the year 2010. These steps exchange the table sales_1_prt_yr_1 with an external table the uses the gpfdist protocol:

1. Ensure that the external table protocol is enabled for the Greenplum Database system.
   
   This example uses the gpfdist protocol. This command starts the gpfdist protocol.
   
   `$ gpfdist`

2. Create a writable external table.

   This CREATE WRITABLE EXTERNAL TABLE command creates a writable external table with the same columns as the partitioned table.

   ```
   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. Create a readable external table that reads the data from that destination of the writable external table created in the previous step.

   This CREATE EXTERNAL TABLE command creates a readable external that uses the same external data as the writable external data.

   ```sql
   CREATE EXTERNAL TABLE sales_2010_ext ( LIKE sales_1_prt_yr_1)
   LOCATION ( 'gpfdist://gpdb_test/sales_2010' )
   FORMAT 'csv';
   ```

4. Copy the data from the leaf child partition into the writable external table.

   This INSERT command copies the data from the child leaf partition table of the partitioned table into the external table.

   ```sql
   INSERT INTO my_sales_ext SELECT * FROM sales_1_prt_yr_1 ;
   ```

5. Exchange the existing leaf child partition with the external table.

   This ALTER TABLE command specifies the EXCHANGE PARTITION clause to switch the readable external table and the leaf child partition.

   ```sql
   ALTER TABLE sales ALTER PARTITION yr_1
   EXCHANGE PARTITION yr_1
   WITH TABLE sales_2010_ext WITHOUT VALIDATION;
   ```

   The external table becomes the leaf child partition with the table name `sales_1_prt_yr_1` and the old leaf child partition becomes the table `sales_2010_ext`.

   **Warning:** In order to ensure queries against the partitioned table return the correct results, the external table data must be valid against the CHECK constraints on the leaf child partition. In this case, the data was taken from the child leaf partition table on which the CHECK constraints were defined.

6. Drop the table that was rolled out of the partitioned table.

   ```sql
   DROP TABLE sales_2010_ext ;
   ```

   You can rename the name of the leaf child partition to indicate that `sales_1_prt_yr_1` is an external table.

   This example command changes the partitionname to `yr_1_ext` and the name of the child leaf partition table to `sales_1_prt_yr_1_ext`.

   ```sql
   ALTER TABLE sales RENAME PARTITION yr_1 TO yr_1_ext ;
   ```

---

**Creating and Using Sequences**

A Greenplum Database sequence object is a special single row table that functions as a number generator. You can use a sequence to generate unique integer identifiers for a row that you add to a table. Declaring a column of type `SERIAL` implicitly creates a sequence counter for use in that table column.

Greenplum Database provides commands to create, alter, and drop a sequence. Greenplum Database also provides built-in functions to return the next value in the sequence (`nextval()`) or to set the sequence to a specific start value (`setval()`).

**Note:** The PostgreSQL `currval()` and `lastval()` sequence functions are not supported in Greenplum Database.

Attributes of a sequence object include the name of the sequence, its increment value, and the last, minimum, and maximum values of the sequence counter. Sequences also have a special boolean attribute named `is_called` that governs the auto-increment behavior of a `nextval()` operation on the sequence.
counter. When a sequence's is_called attribute is true, nextval() increments the sequence counter before returning the value. When the is_called attribute value of a sequence is false, nextval() does not increment the counter before returning the value.

Creating a Sequence

The CREATE SEQUENCE command creates and initializes a sequence with the given sequence name and optional start value. The sequence name must be distinct from the name of any other sequence, table, index, or view in the same schema. For example:

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

When you create a new sequence, Greenplum Database sets the sequence is_called attribute to false. Invoking nextval() on a newly-created sequence does not increment the sequence counter, but returns the sequence start value and sets is_called to true.

Using a Sequence

After you create a sequence with the CREATE SEQUENCE command, you can examine the sequence and use the sequence built-in functions.

Examining Sequence Attributes

To examine the current attributes of a sequence, query the sequence directly. For example, to examine a sequence named myserial:

```sql
SELECT * FROM myserial;
```

Returning the Next Sequence Counter Value

You can invoke the nextval() built-in function to return and use the next value in a sequence. The following command inserts the next value of the sequence named myserial into the first column of a table named vendors:

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

nextval() uses the sequence's is_called attribute value to determine whether or not to increment the sequence counter before returning the value. nextval() advances the counter when is_called is true. nextval() sets the sequence is_called attribute to true before returning.

A nextval() operation is never rolled back. A fetched value is considered used, even if the transaction that performed the nextval() fails. This means that failed transactions can leave unused holes in the sequence of assigned values.

**Note:** You cannot use the nextval() function in UPDATE or DELETE statements if mirroring is enabled in Greenplum Database.

Setting the Sequence Counter Value

You can use the Greenplum Database setval() built-in function to set the counter value for a sequence. For example, the following command sets the counter value of the sequence named myserial to 201:

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

setval() has two function signatures: setval(sequence, start_val) and setval(sequence, start_val, is_called). The default behaviour of setval( sequence, start_val) sets the sequence is_called attribute value to true.
If you do not want the sequence counter advanced on the next `nextval()` call, use the `setval(sequence, start_val, is_called)` function signature, passing a `false` argument:

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

`setval()` operations are never rolled back.

---

### Altering a Sequence

The `ALTER SEQUENCE` command changes the attributes of an existing sequence. You can alter the sequence minimum, maximum, and increment values. You can also restart the sequence at a specific value.

Any parameters not set in the `ALTER SEQUENCE` command retain their prior settings.

The following command restarts the sequence named `myserial` at value 105:

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

When you alter a sequence start value with the `ALTER SEQUENCE` command, Greenplum Database sets the sequence's `is_called` attribute to `false`. The first `nextval()` invoked after restarting a sequence does not advance the sequence counter, but returns the sequence restart value and sets `is_called` to `true`.

---

### Dropping a Sequence

The `DROP SEQUENCE` command removes a sequence. For example, the following command removes the sequence named `myserial`:

```
DROP SEQUENCE myserial;
```

---

### Specifying a Sequence as the Default Value for a Column

You can reference a sequence directly in the `CREATE TABLE` command in addition to using the `SERIAL` or `BIGSERIAL` types. For example:

```
CREATE TABLE tablename ( id INT4 DEFAULT nextval('myserial'), name text );
```

You can also alter a table column to set its default value to a sequence counter:

```
ALTER TABLE tablename ALTER COLUMN id SET DEFAULT nextval('myserial');
```

---

### Sequence Wraparound

By default, a sequence does not wrap around. That is, when a sequence reaches the max value ( +32767 for `SMALLSERIAL`, +2147483647 for `SERIAL`, +9223372036854775807 for `BIGSERIAL`), every subsequent `nextval()` call produces an error. You can alter a sequence to make it cycle around and start at 1 again:

```
ALTER SEQUENCE myserial CYCLE;
```

You can also specify the wraparound behaviour when you create the sequence:

```
CREATE SEQUENCE myserial CYCLE;
```
Using Indexes in Greenplum Database

In most traditional databases, indexes can greatly improve data access times. However, in a distributed database such as Greenplum, indexes should be used more sparingly. Greenplum Database performs very fast sequential scans; indexes use a random seek pattern to locate records on disk. Greenplum data is distributed across the segments, so each segment scans a smaller portion of the overall data to get the result. With table partitioning, the total data to scan may be even smaller. Because business intelligence (BI) query workloads generally return very large data sets, using indexes is not efficient.

First try your query workload without adding indexes. Indexes are more likely to improve performance for OLTP workloads, where the query is returning a single record or a small subset of data. Indexes can also improve performance on compressed append-optimized tables for queries that return a targeted set of rows, as the optimizer can use an index access method rather than a full table scan when appropriate. For compressed data, an index access method means only the necessary rows are uncompressed.

Greenplum Database automatically creates primary key constraints for tables with primary keys. To create an index on a partitioned table, create an index on the partitioned table that you created. The index is propagated to all the child tables created by Greenplum Database. Creating an index on a table that is created by Greenplum Database for use by a partitioned table is not supported.

Note that a unique constraint (such as a primary key constraint) implicitly creates a unique index that must include all the columns of the distribution key and any partitioning key. The unique constraint is enforced across the entire table, including all table partitions (if any).

Indexes add some database overhead — they use storage space and must be maintained when the table is updated. Ensure that the query workload uses the indexes that you create, and check that the indexes you add improve query performance (as compared to a sequential scan of the table). To determine whether indexes are being used, examine the query explain plans. See Query Profiling.

Consider the following points when you create indexes.

- Your Query Workload. Indexes improve performance for workloads where queries return a single record or a very small data set, such as OLTP workloads.
- Compressed Tables. Indexes can improve performance on compressed append-optimized tables for queries that return a targeted set of rows. For compressed data, an index access method means only the necessary rows are uncompressed.
- Avoid indexes on frequently updated columns. Creating an index on a column that is frequently updated increases the number of writes required when the column is updated.
- Create selective B-tree indexes. Index selectivity is a ratio of the number of distinct values a column has divided by the number of rows in a table. For example, if a table has 1000 rows and a column has 800 distinct values, the selectivity of the index is 0.8, which is considered good. Unique indexes always have a selectivity ratio of 1.0, which is the best possible. Greenplum Database allows unique indexes only on distribution key columns.
- Use Bitmap indexes for low selectivity columns. The Greenplum Database Bitmap index type is not available in regular PostgreSQL. See About Bitmap Indexes.
- Index columns used in joins. An index on a column used for frequent joins (such as a foreign key column) can improve join performance by enabling more join methods for the query optimizer to use.
- Index columns frequently used in predicates. Columns that are frequently referenced in WHERE clauses are good candidates for indexes.
- Avoid overlapping indexes. Indexes that have the same leading column are redundant.
- Drop indexes for bulk loads. For mass loads of data into a table, consider dropping the indexes and re-creating them after the load completes. This is often faster than updating the indexes.
- Consider a clustered index. Clustering an index means that the records are physically ordered on disk according to the index. If the records you need are distributed randomly on disk, the database has to seek across the disk to fetch the records requested. If the records are stored close together, the fetching operation is more efficient. For example, a clustered index on a date column where the data is
ordered sequentially by date. A query against a specific date range results in an ordered fetch from the
disk, which leverages fast sequential access.

To cluster an index in Greenplum Database

Using the `CLUSTER` command to physically reorder a table based on an index can take a long time with
very large tables. To achieve the same results much faster, you can manually reorder the data on disk by
creating an intermediate table and loading the data in the desired order. For example:

```
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;
```

Index Types

Greenplum Database supports the Postgres index types B-tree and GiST. Hash and GIN indexes are not
supported. Each index type uses a different algorithm that is best suited to different types of queries. B-tree
indexes fit the most common situations and are the default index type. See Index Types in the PostgreSQL
documentation for a description of these types.

Note: Greenplum Database allows unique indexes only if the columns of the index key are the
same as (or a superset of) the Greenplum distribution key. Unique indexes are not supported on
append-optimized tables. On partitioned tables, a unique index cannot be enforced across all child
table partitions of a partitioned table. A unique index is supported only within a partition.

About Bitmap Indexes

Greenplum Database provides the Bitmap index type. Bitmap indexes are best suited to data warehousing
applications and decision support systems with large amounts of data, many ad hoc queries, and few data
modification (DML) transactions.

An index provides pointers to the rows in a table that contain a given key value. A regular index stores a list
of tuple IDs for each key corresponding to the rows with that key value. Bitmap indexes store a bitmap for
each key value. Regular indexes can be several times larger than the data in the table, but bitmap indexes
provide the same functionality as a regular index and use a fraction of the size of the indexed data.

Each bit in the bitmap corresponds to a possible tuple ID. If the bit is set, the row with the corresponding
tuple ID contains the key value. A mapping function converts the bit position to a tuple ID. Bitmaps are
compressed for storage. If the number of distinct key values is small, bitmap indexes are much smaller,
compress better, and save considerable space compared with a regular index. The size of a bitmap index
is proportional to the number of rows in the table times the number of distinct values in the indexed column.

Bitmap indexes are most effective for queries that contain multiple conditions in the `WHERE` clause. Rows
that satisfy some, but not all, conditions are filtered out before the table is accessed. This improves
response time, often dramatically.

When to Use Bitmap Indexes

Bitmap indexes are best suited to data warehousing applications where users query the data rather than
update it. Bitmap indexes perform best for columns that have between 100 and 100,000 distinct values
and when the indexed column is often queried in conjunction with other indexed columns. Columns with
fewer than 100 distinct values, such as a gender column with two distinct values (male and female), usually
do not benefit much from any type of index. On a column with more than 100,000 distinct values, the
performance and space efficiency of a bitmap index decline.

Bitmap indexes can improve query performance for ad hoc queries. `AND` and `OR` conditions in the `WHERE`
clause of a query can be resolved quickly by performing the corresponding Boolean operations directly on
the bitmaps before converting the resulting bitmap to tuple ids. If the resulting number of rows is small, the query can be answered quickly without resorting to a full table scan.

**When Not to Use Bitmap Indexes**

Do not use bitmap indexes for unique columns or columns with high cardinality data, such as customer names or phone numbers. The performance gains and disk space advantages of bitmap indexes start to diminish on columns with 100,000 or more unique values, regardless of the number of rows in the table.

Bitmap indexes are not suitable for OLTP applications with large numbers of concurrent transactions modifying the data.

Use bitmap indexes sparingly. Test and compare query performance with and without an index. Add an index only if query performance improves with indexed columns.

**Creating an Index**

The `CREATE INDEX` command defines an index on a table. A B-tree index is the default index type. For example, to create a B-tree index on the column `gender` in the table `employee`:

```
CREATE INDEX gender_idx ON employee (gender);
```

To create a bitmap index on the column `title` in the table `films`:

```
CREATE INDEX title_bmp_idx ON films USING bitmap (title);
```

**Examining Index Usage**

Greenplum Database indexes do not require maintenance and tuning. You can check which indexes are used by the real-life query workload. Use the `EXPLAIN` command to examine index usage for a query.

The query plan shows the steps or **plan nodes** that the database will take to answer a query and time estimates for each plan node. To examine the use of indexes, look for the following query plan node types in your `EXPLAIN` output:

- **Index Scan** - A scan of an index.
- **Bitmap Heap Scan** - Retrieves all from the bitmap generated by BitmapAnd, BitmapOr, or BitmapIndexScan and accesses the heap to retrieve the relevant rows.
- **Bitmap Index Scan** - Compute a bitmap by OR-ing all bitmaps that satisfy the query predicates from the underlying index.
- **BitmapAnd** or **BitmapOr** - Takes the bitmaps generated from multiple BitmapIndexScan nodes, ANDs or ORs them together, and generates a new bitmap as its output.

You have to experiment to determine the indexes to create. Consider the following points.

- Run `ANALYZE` after you create or update an index. `ANALYZE` collects table statistics. The query optimizer uses table statistics to estimate the number of rows returned by a query and to assign realistic costs to each possible query plan.
- Use real data for experimentation. Using test data for setting up indexes tells you what indexes you need for the test data, but that is all.
- Do not use very small test data sets as the results can be unrealistic or skewed.
- Be careful when developing test data. Values that are similar, completely random, or inserted in sorted order will skew the statistics away from the distribution that real data would have.
- You can force the use of indexes for testing purposes by using run-time parameters to turn off specific plan types. For example, turn off sequential scans (`enable_seqscan`) and nested-loop joins (`enable_nestloop`), the most basic plans, to force the system to use a different plan. Time your query with and without indexes and use the `EXPLAIN ANALYZE` command to compare the results.
Managing Indexes

Use the `REINDEX` command to rebuild a poorly-performing index. `REINDEX` rebuilds an index using the data stored in the index's table, replacing the old copy of the index.

To rebuild all indexes on a table

```sql
REINDEX my_table;
```

To rebuild a particular index

```sql
REINDEX my_index;
```

Dropping an Index

The `DROP INDEX` command removes an index. For example:

```sql
DROP INDEX title_idx;
```

When loading data, it can be faster to drop all indexes, load, then recreate the indexes.

Creating and Managing Views

Views enable you to save frequently used or complex queries, then access them in a `SELECT` statement as if they were a table. A view is not physically materialized on disk: the query runs as a subquery when you access the view.

If a subquery is associated with a single query, consider using the `WITH` clause of the `SELECT` command instead of creating a seldom-used view.

Creating Views

The `CREATE VIEW` command defines a view of a query. For example:

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

Views ignore `ORDER BY` and `SORT` operations stored in the view.

Dropping Views

The `DROP VIEW` command removes a view. For example:

```sql
DROP VIEW topten;
```

The `DROP VIEW ... CASCADE` command also removes all dependent objects. As example, if another view depends on the view which is about to be dropped, the other view will be dropped as well. Without the `CASCADE` option, the `DROP VIEW` command will fail.
Inserting, Updating, and Deleting Data

This section provides information about manipulating data and concurrent access in Greenplum Database.

This topic includes the following subtopics:

- About Concurrency Control in Greenplum Database
- Inserting Rows
- Updating Existing Rows
- Deleting Rows
- Working With Transactions
- Vacuuming the Database

About Concurrency Control in Greenplum Database

Greenplum Database and PostgreSQL do not use locks for concurrency control. They maintain data consistency using a multiversion model, Multiversion Concurrency Control (MVCC). MVCC achieves transaction isolation for each database session, and each query transaction sees a snapshot of data. This ensures the transaction sees consistent data that is not affected by other concurrent transactions.

Because MVCC does not use explicit locks for concurrency control, lock contention is minimized and Greenplum Database maintains reasonable performance in multiuser environments. Locks acquired for querying (reading) data do not conflict with locks acquired for writing data.

Greenplum Database provides multiple lock modes to control concurrent access to data in tables. Most Greenplum Database SQL commands automatically acquire the appropriate locks to ensure that referenced tables are not dropped or modified in incompatible ways while a command executes. For applications that cannot adapt easily to MVCC behavior, you can use the `LOCK` command to acquire explicit locks. However, proper use of MVCC generally provides better performance.

Table 42: Lock Modes in Greenplum Database

<table>
<thead>
<tr>
<th>Lock Mode</th>
<th>Associated SQL Commands</th>
<th>Conflicts With</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS SHARE</td>
<td>SELECT</td>
<td>ACCESS EXCLUSIVE</td>
</tr>
<tr>
<td>ROW SHARE</td>
<td>SELECT FOR SHARE, SELECT FOR UPDATE</td>
<td>EXCLUSIVE, ACCESS EXCLUSIVE</td>
</tr>
<tr>
<td>ROW EXCLUSIVE</td>
<td>INSERT, COPY</td>
<td>SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE</td>
</tr>
<tr>
<td>SHARE UPDATE EXCLUSIVE</td>
<td>VACUUM (without FULL), ANALYZE</td>
<td>SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE</td>
</tr>
<tr>
<td>SHARE</td>
<td>CREATE INDEX</td>
<td>ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE</td>
</tr>
<tr>
<td>SHARE ROW EXCLUSIVE</td>
<td></td>
<td>ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE</td>
</tr>
<tr>
<td>Lock Mode</td>
<td>Associated SQL Commands</td>
<td>Conflicts With</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EXCLUSIVE</td>
<td>DELETE, UPDATE, SELECT FOR UPDATE, See Note</td>
<td>ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE</td>
</tr>
<tr>
<td>ACCESS EXCLUSIVE</td>
<td>ALTER TABLE, DROP TABLE, TRUNCATE, REINDEX, CLUSTER, VACUUM FULL</td>
<td>ACCESS SHARE, ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, ACCESS EXCLUSIVE</td>
</tr>
</tbody>
</table>

**Note:** Greenplum Database acquires the more restrictive EXCLUSIVE lock (rather than ROW EXCLUSIVE in PostgreSQL) for UPDATE, DELETE, and SELECT FOR UPDATE.

**Inserting Rows**

Use the **INSERT** command to create rows in a table. This command requires the table name and a value for each column in the table; you may optionally specify the column names in any order. If you do not specify column names, list the data values in the order of the columns in the table, separated by commas.

For example, to specify the column names and the values to insert:

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

To specify only the values to insert:

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

Usually, the data values are literals (constants), but you can also use scalar expressions. For example:

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

You can insert multiple rows in a single command. For example:

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

To insert data into a partitioned table, you specify the root partitioned table, the table created with the **CREATE TABLE** command. You also can specify a leaf child table of the partitioned table in an **INSERT** command. An error is returned if the data is not valid for the specified leaf child table. Specifying a child table that is not a leaf child table in the **INSERT** command is not supported.

To insert large amounts of data, use external tables or the **COPY** command. These load mechanisms are more efficient than **INSERT** for inserting large quantities of rows. See **Loading and Unloading Data** for more information about bulk data loading.

The storage model of append-optimized tables is optimized for bulk data loading. Greenplum does not recommend single row **INSERT** statements for append-optimized tables. For append-optimized tables, Greenplum Database supports a maximum of 127 concurrent **INSERT** transactions into a single append-optimized table.
Updating Existing Rows

The UPDATE command updates rows in a table. You can update all rows, a subset of all rows, or individual rows in a table. You can update each column separately without affecting other columns.

To perform an update, you need:

• The name of the table and columns to update
• The new values of the columns
• One or more conditions specifying the row or rows to be updated.

For example, the following command updates all products that have a price of 5 to have a price of 10:

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

Using UPDATE in Greenplum Database has the following restrictions:

• The Greenplum distribution key columns may not be updated.
• If mirrors are enabled, you cannot use STABLE or VOLATILE functions in an UPDATE statement.
• Greenplum Database does not support the RETURNING clause.
• Greenplum Database partitioning columns cannot be updated.

Deleting Rows

The DELETE command deletes rows from a table. Specify a WHERE clause to delete rows that match certain criteria. If you do not specify a WHERE clause, all rows in the table are deleted. The result is a valid, but empty, table. For example, to remove all rows from the products table that have a price of 10:

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

To delete all rows from a table:

```
DELETE FROM products;
```

Using DELETE in Greenplum Database has similar restrictions to using UPDATE:

• If mirrors are enabled, you cannot use STABLE or VOLATILE functions in an UPDATE statement.
• The RETURNING clause is not supported in Greenplum Database.

Truncating a Table

Use the TRUNCATE command to quickly remove all rows in a table. For example:

```
TRUNCATE mytable;
```

This command empties a table of all rows in one operation. Note that TRUNCATE does not scan the table, therefore it does not process inherited child tables or ON DELETE rewrite rules. The command truncates only rows in the named table.

Working With Transactions

Transactions allow you to bundle multiple SQL statements in one all-or-nothing operation.

The following are the Greenplum Database SQL transaction commands:

• BEGIN or START TRANSACTION starts a transaction block.
• END or COMMIT commits the results of a transaction.
• ROLLBACK abandons a transaction without making any changes.
• **SAVEPOINT** marks a place in a transaction and enables partial rollback. You can roll back commands executed after a savepoint while maintaining commands executed before the savepoint.
• **ROLLBACK TO SAVEPOINT** rolls back a transaction to a savepoint.
• **RELEASE SAVEPOINT** destroys a savepoint within a transaction.

### Transaction Isolation Levels

Greenplum Database accepts the standard SQL transaction levels as follows:

• *read uncommitted and read committed* behave like the standard **read committed**
• **repeatable read** is disallowed. If the behavior of **repeatable read** is required, use **serializable**.
• **serializable** behaves in a manner similar to SQL standard **serializable**

The following information describes the behavior of the Greenplum transaction levels:

**read committed/read uncommitted** — Provides fast, simple, partial transaction isolation. With read committed and read uncommitted transaction isolation, **SELECT**, **UPDATE**, and **DELETE** transactions operate on a snapshot of the database taken when the query started.

A **SELECT** query:
• Sees data committed before the query starts.
• Sees updates executed within the transaction.
• Does not see uncommitted data outside the transaction.
• Can possibly see changes that concurrent transactions made if the concurrent transaction is committed after the initial read in its own transaction.

Successive **SELECT** queries in the same transaction can see different data if other concurrent transactions commit changes before the queries start. **UPDATE** and **DELETE** commands find only rows committed before the commands started.

Read committed or read uncommitted transaction isolation allows concurrent transactions to modify or lock a row before **UPDATE** or **DELETE** finds the row. Read committed or read uncommitted transaction isolation may be inadequate for applications that perform complex queries and updates and require a consistent view of the database.

**serializable** — Provides strict transaction isolation in which transactions execute as if they run one after another rather than concurrently. Applications on the serializable level must be designed to retry transactions in case of serialization failures. In Greenplum Database, **SERIALIZABLE** prevents dirty reads, non-repeatable reads, and phantom reads without expensive locking, but there are other interactions that can occur between some **SERIALIZABLE** transactions in Greenplum Database that prevent them from being truly serializable. Transactions that run concurrently should be examined to identify interactions that are not prevented by disallowing concurrent updates of the same data. Problems identified can be prevented by using explicit table locks or by requiring the conflicting transactions to update a dummy row introduced to represent the conflict.

A **SELECT** query:
• Sees a snapshot of the data as of the start of the transaction (not as of the start of the current query within the transaction).
• Sees only data committed before the query starts.
• Sees updates executed within the transaction.
• Does not see uncommitted data outside the transaction.
• Does not see changes that concurrent transactions made.

Successive **SELECT** commands within a single transaction always see the same data.

**UPDATE, DELETE, SELECT FOR UPDATE,** and **SELECT FOR SHARE** commands find only rows committed before the command started. If a concurrent transaction has already updated, deleted, or locked a target row when the row is found, the serializable or repeatable read transaction waits for the concurrent transaction to update the row, delete the row, or roll back.
If the concurrent transaction updates or deletes the row, the serializable or repeatable read transaction rolls back. If the concurrent transaction rolls back, then the serializable or repeatable read transaction updates or deletes the row.

The default transaction isolation level in Greenplum Database is *read committed*. To change the isolation level for a transaction, declare the isolation level when you `BEGIN` the transaction or use the `SET TRANSACTION` command after the transaction starts.

**Vacuuming the Database**

Deleted or updated data rows occupy physical space on disk even though new transactions cannot see them. Periodically running the `VACUUM` command removes these expired rows. For example:

```
VACUUM mytable;
```

The `VACUUM` command collects table-level statistics such as the number of rows and pages. Vacuum all tables after loading data, including append-optimized tables. For information about recommended routine vacuum operations, see *Routine Vacuum and Analyze*.

**Important:** The `VACUUM`, `VACUUM FULL`, and `VACUUM ANALYZE` commands should be used to maintain the data in a Greenplum database especially if updates and deletes are frequently performed on your database data. See the `VACUUM` command in the *Greenplum Database Reference Guide* for information about using the command.

**Configuring the Free Space Map**

Expired rows are held in the *free space map*. The free space map must be sized large enough to hold all expired rows in your database. If not, a regular `VACUUM` command cannot reclaim space occupied by expired rows that overflow the free space map.

`VACUUM FULL` reclaims all expired row space, but it is an expensive operation and can take an unacceptably long time to finish on large, distributed Greenplum Database tables. If the free space map overflows, you can recreate the table with a `CREATE TABLE AS` statement and drop the old table. Using `VACUUM FULL` is discouraged.

Size the free space map with the following server configuration parameters:

- `max_fsm_pages`
- `max_fsm_relations`
Querying Data

This topic provides information about using SQL in Greenplum databases.

You enter SQL statements called queries to view, change, and analyze data in a database using the `psql` interactive SQL client and other client tools.

About Greenplum Query Processing

This topic provides an overview of how Greenplum Database processes queries. Understanding this process can be useful when writing and tuning queries.

Users issue queries to Greenplum Database as they would to any database management system. They connect to the database instance on the Greenplum master host using a client application such as `psql` and submit SQL statements.

Understanding Query Planning and Dispatch

The master receives, parses, and optimizes the query. The resulting query plan is either parallel or targeted. The master dispatches parallel query plans to all segments, as shown in Figure 25: Dispatching the Parallel Query Plan. The master dispatches targeted query plans to a single segment, as shown in Figure 26: Dispatching a Targeted Query Plan. Each segment is responsible for executing local database operations on its own set of data.

Most database operations—such as table scans, joins, aggregations, and sorts—execute across all segments in parallel. Each operation is performed on a segment database independent of the data stored in the other segment databases.

Figure 25: Dispatching the Parallel Query Plan
Certain queries may access only data on a single segment, such as single-row \texttt{INSERT}, \texttt{UPDATE}, \texttt{DELETE}, or \texttt{SELECT} operations or queries that filter on the table distribution key column(s). In queries such as these, the query plan is not dispatched to all segments, but is targeted at the segment that contains the affected or relevant row(s).

![Figure 26: Dispatching a Targeted Query Plan](image)

### Understanding Greenplum Query Plans

A query plan is the set of operations Greenplum Database will perform to produce the answer to a query. Each node or step in the plan represents a database operation such as a table scan, join, aggregation, or sort. Plans are read and executed from bottom to top.

In addition to common database operations such as tables scans, joins, and so on, Greenplum Database has an additional operation type called \textit{motion}. A motion operation involves moving tuples between the segments during query processing. Note that not every query requires a motion. For example, a targeted query plan does not require data to move across the interconnect.

To achieve maximum parallelism during query execution, Greenplum divides the work of the query plan into \textit{slices}. A slice is a portion of the plan that segments can work on independently. A query plan is sliced wherever a \textit{motion} operation occurs in the plan, with one slice on each side of the motion.

For example, consider the following simple query involving a join between two tables:

```sql
SELECT customer, amount
FROM sales JOIN customer USING (cust_id)
WHERE dateCol = '04-30-2016';
```

![Figure 27: Query Slice Plan](image) shows the query plan. Each segment receives a copy of the query plan and works on it in parallel.

The query plan for this example has a \textit{redistribute motion} that moves tuples between the segments to complete the join. The redistribute motion is necessary because the customer table is distributed across the segments by \texttt{cust_id}, but the sales table is distributed across the segments by \texttt{sale_id}. To perform
the join, the sales tuples must be redistributed by cust_id. The plan is sliced on either side of the redistribute motion, creating slice 1 and slice 2.

This query plan has another type of motion operation called a gather motion. A gather motion is when the segments send results back up to the master for presentation to the client. Because a query plan is always sliced wherever a motion occurs, this plan also has an implicit slice at the very top of the plan (slice 3). Not all query plans involve a gather motion. For example, a `CREATE TABLE x AS SELECT...` statement would not have a gather motion because tuples are sent to the newly created table, not to the master.

![Query Slice Plan](image)

**Figure 27: Query Slice Plan**

**Understanding Parallel Query Execution**

Greenplum creates a number of database processes to handle the work of a query. On the master, the query worker process is called the query dispatcher (QD). The QD is responsible for creating and dispatching the query plan. It also accumulates and presents the final results. On the segments, a query worker process is called a query executor (QE). A QE is responsible for completing its portion of work and communicating its intermediate results to the other worker processes.

There is at least one worker process assigned to each slice of the query plan. A worker process works on its assigned portion of the query plan independently. During query execution, each segment will have a number of processes working on the query in parallel.

Related processes that are working on the same slice of the query plan but on different segments are called gangs. As a portion of work is completed, tuples flow up the query plan from one gang of processes
to the next. This inter-process communication between the segments is referred to as the *interconnect* component of Greenplum Database.

*Figure 28: Query Worker Processes* shows the query worker processes on the master and two segment instances for the query plan illustrated in *Figure 27: Query Slice Plan*.

---

**About GPORCA**

In Greenplum Database, the default GPORCA optimizer co-exists with the legacy query optimizer. These sections describe GPORCA functionality and usage:

- Overview of GPORCA
- Enabling and Disabling GPORCA
- Considerations when Using GPORCA
- GPORCA Features and Enhancements
- Changed Behavior with the GPORCA
- GPORCA Limitations
- Determining the Query Optimizer that is Used
- About Uniform Multi-level Partitioned Tables

**Overview of GPORCA**

GPORCA extends the planning and optimization capabilities of the Greenplum Database legacy optimizer. GPORCA is extensible and achieves better optimization in multi-core architecture environments. Greenplum Database uses GPORCA by default to generate an execution plan for a query when possible.

GPORCA also enhances Greenplum Database query performance tuning in the following areas:

- Queries against partitioned tables
- Queries that contain a common table expression (CTE)
- Queries that contain subqueries
In Greenplum Database, GPORCA co-exists with the legacy query optimizer. By default, Greenplum Database uses GPORCA. If GPORCA cannot be used, then the legacy query optimizer is used.

The following figure shows how GPORCA fits into the query planning architecture.

![Query Plan Generation Diagram]

**Note:** All legacy query optimizer (planner) server configuration parameters are ignored by GPORCA. However, if Greenplum Database falls back to the legacy optimizer, the planner server configuration parameters will impact the query plan generation. For a list of legacy query optimizer (planner) server configuration parameters, see *Query Tuning Parameters*.

### Enabling and Disabling GPORCA

By default, Greenplum Database uses GPORCA instead of the legacy query planner. Server configuration parameters enable or disable GPORCA.

- Set the `optimizer_analyze_root_partition` parameter to `on` to enable statistics collection for the root partition of a partitioned table.
- Set the `optimizer` parameter to `on` (the default) or `off` to enable or disable GPORCA. You can set the parameter at these levels:
  - A Greenplum Database system
  - A specific Greenplum database
  - A session or query

**Note:** You can disable the ability to enable or disable GPORCA with the server configuration parameter `optimizer_control`. For information about the server configuration parameters, see the *Greenplum Database Reference Guide*.

**Important:** If you intend to execute queries on partitioned tables with GPORCA enabled, you must collect statistics on the partitioned table root partition with the `ANALYZE ROOTPARTITION` command. The command `ANALYZE ROOTPARTITION` collects statistics on the root partition of a partitioned table based on the data in the table. Statistics are not collected on the leaf partitions, leaf partition data is only sampled. If you specify a list of column names for a partitioned table, the statistics for the columns and the root partition are collected. For information on the `ANALYZE` command, see the *Greenplum Database Reference Guide*.

You can also use the Greenplum Database utility `analyzedb` to update table statistics. The Greenplum Database utility `analyzedb` can update statistics for multiple tables in parallel.
utility can also check table statistics and update statistics only if the statistics are not current or do not exist. For information about the `analyzedb` utility, see the Greenplum Database Utility Guide.

As part of routine database maintenance, Refresh statistics on the root partition when there are significant changes to leaf partition data.

**Setting the optimizer_analyze_root_partition Parameter**

When the configuration parameter `optimizer_analyze_root_partition` is set to `on`, root partition statistics will be collected when `ANALYZE` is run on a partitioned table. Root partition statistics are required by GPORCA.

1. Log into the Greenplum Database master host as `gpadmin`, the Greenplum Database administrator.
2. Set the values of the server configuration parameters. These Greenplum Database `gpconfig` utility commands sets the value of the parameters to `on`:

   ```bash
   $ gpconfig -c optimizer_analyze_root_partition -v on --masteronly
   ```

3. Restart Greenplum Database. This Greenplum Database `gpstop` utility command reloads the `postgresql.conf` files of the master and segments without shutting down Greenplum Database.

   ```bash
   gpstop -u
   ```

**Enabling GPORCA for a System**

Set the server configuration parameter `optimizer` for the Greenplum Database system.

1. Log into the Greenplum Database master host as `gpadmin`, the Greenplum Database administrator.
2. Set the values of the server configuration parameters. These Greenplum Database `gpconfig` utility commands sets the value of the parameters to `on`:

   ```bash
   $ gpconfig -c optimizer -v on --masteronly
   ```

3. Restart Greenplum Database. This Greenplum Database `gpstop` utility command reloads the `postgresql.conf` files of the master and segments without shutting down Greenplum Database.

   ```bash
   gpstop -u
   ```

**Enabling GPORCA for a Database**

Set the server configuration parameter `optimizer` for individual Greenplum databases with the `ALTER DATABASE` command. For example, this command enables GPORCA for the database `test_db`.

```bash
> ALTER DATABASE test_db SET OPTIMIZER = ON ;
```

**Enabling GPORCA for a Session or a Query**

You can use the `SET` command to set `optimizer` server configuration parameter for a session. For example, after you use the `psql` utility to connect to Greenplum Database, this `SET` command enables GPORCA:

```bash
> set optimizer = on ;
```

To set the parameter for a specific query, include the `SET` command prior to running the query.

**Considerations when Using GPORCA**

To execute queries optimally with GPORCA, query criteria to consider.
Ensure the following criteria are met:

- The table does not contain multi-column partition keys.
- The multi-level partitioned table is a uniform multi-level partitioned table. See About Uniform Multi-level Partitioned Tables.
- The server configuration parameter `optimizer_enable_master_only_queries` is set to `on` when running against master only tables such as the system table `pg_attribute`. For information about the parameter, see the Greenplum Database Reference Guide.

  Note: Enabling this parameter decreases performance of short running catalog queries. To avoid this issue, set this parameter only for a session or a query.

- Statistics have been collected on the root partition of a partitioned table.

If the partitioned table contains more than 20,000 partitions, consider a redesign of the table schema.

These server configuration parameters affect GPORCA query processing.

- `optimizer_cte_inlining_bound` controls the amount of inlining performed for common table expression (CTE) queries (queries that contain a `WHERE` clause).
- `optimizer_force_multistage_agg` forces GPORCA to choose a 3 stage aggregate plan for a scalar distinct qualified aggregate.
- `optimizer_force_three_stage_scalar_dqa` forces GPORCA to choose a plan with multistage aggregates when such a plan alternative is generated.
- `optimizer_join_order_threshold` specifies the maximum number of join children for which GPORCA uses the dynamic programming-based join ordering algorithm.
- `optimizer_nestloop_factor` controls nested loop join cost factor to apply to during query optimization.
- `optimizer_parallel_union` controls the amount of parallelization that occurs for queries that contain a `UNION` or `UNION ALL` clause. When the value is `on`, GPORCA can generate a query plan the child operations of a `UNION` or `UNION ALL` operation execute in parallel on segment instances.
- `optimizer_sort_factor` controls the cost factor that GPORCA applies to sorting operations during query optimization. The cost factor can be adjusted for queries when data skew is present.

These server configuration parameters control the display and logging of information.

- `optimizer_print_missing_stats` controls the display of column information about columns with missing statistics for a query (default is `true`)
- `optimizer_print_optimization_stats` controls the logging of GPORCA query optimization metrics for a query (default is `off`)

For information about the parameters, see the Greenplum Database Reference Guide.

GPORCA generates minidumps to describe the optimization context for a given query. The minidump files are used by Pivotal support to analyze Greenplum Database issues. The information in the file is not in a format that can be easily used for debugging or troubleshooting. The minidump file is located under the master data directory and uses the following naming format:

Minidump_date_time.mdp

For information about the minidump file, see the server configuration parameter `optimizer_minidump` in the Greenplum Database Reference Guide.

When the `EXPLAIN ANALYZE` command uses GPORCA, the `EXPLAIN` plan shows only the number of partitions that are being eliminated. The scanned partitions are not shown. To show the name of the scanned partitions in the segment logs set the server configuration parameter `gp_log_dynamic_partition_pruning` to `on`. This example `SET` command enables the parameter.

```
SET gp_log_dynamic_partition_pruning = on;
```
GPORCA Features and Enhancements

GPORCA, the Greenplum next generation query optimizer, includes enhancements for specific types of queries and operations:

- Queries Against Partitioned Tables
- Queries that Contain Subqueries
- Queries that Contain Common Table Expressions
- DML Operation Enhancements with GPORCA

GPORCA also includes these optimization enhancements:

- Improved join ordering
- Join-Aggregate reordering
- Sort order optimization
- Data skew estimates included in query optimization

Queries Against Partitioned Tables

GPORCA includes these enhancements for queries against partitioned tables:

- Partition elimination is improved.
- Uniform multi-level partitioned tables are supported. For information about uniform multi-level partitioned tables, see About Uniform Multi-level Partitioned Tables
- Query plan can contain the Partition selector operator.
- Partitions are not enumerated in EXPLAIN plans.

For queries that involve static partition selection where the partitioning key is compared to a constant, GPORCA lists the number of partitions to be scanned in the EXPLAIN output under the Partition Selector operator. This example Partition Selector operator shows the filter and number of partitions selected:

```
Partition Selector for Part_Table (dynamic scan id: 1)
  Filter: a > 10
  Partitions selected:  1 (out of 3)
```

For queries that involve dynamic partition selection where the partitioning key is compared to a variable, the number of partitions that are scanned will be known only during query execution. The partitions selected are not shown in the EXPLAIN output.

- Plan size is independent of number of partitions.
- Out of memory errors caused by number of partitions are reduced.

This example CREATE TABLE command creates a range partitioned table.

```
CREATE TABLE sales(order_id int, item_id int, amount numeric(15,2),
  date date, yr_qtr int)
  range partitioned by yr_qtr;
```

GPORCA improves on these types of queries against partitioned tables:

- Full table scan. Partitions are not enumerated in plans.

```
SELECT * FROM sales;
```

- Query with a constant filter predicate. Partition elimination is performed.

```
SELECT * FROM sales WHERE yr_qtr = 201501;
```
• Range selection. Partition elimination is performed.

```
SELECT * FROM sales WHERE yr_qtr BETWEEN 201601 AND 201704;
```

• Joins involving partitioned tables. In this example, the partitioned dimension table `date_dim` is joined with fact table `catalog_sales`:

```
SELECT * FROM catalog_sales
WHERE date_id IN (SELECT id FROM date_dim WHERE month=12);
```

### Queries that Contain Subqueries

GPORCA handles subqueries more efficiently. A subquery is a query that is nested inside an outer query block. In the following query, the `SELECT` in the `WHERE` clause is a subquery.

```
SELECT * FROM part
WHERE price > (SELECT avg(price) FROM part);
```

GPORCA also handles queries that contain a correlated subquery (CSQ) more efficiently. A correlated subquery is a subquery that uses values from the outer query. In the following query, the `price` column is used in both the outer query and the subquery.

```
SELECT * FROM part p1
WHERE price > (SELECT avg(price) FROM part p2
WHERE p2.brand = p1.brand);
```

GPORCA generates more efficient plans for the following types of subqueries:

• **CSQ in the `SELECT` list.**

```
SELECT *
(SELECT min(price) FROM part p2 WHERE p1.brand = p2.brand)
AS foo
FROM part p1;
```

• **CSQ in disjunctive (OR) filters.**

```
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)
```

• **Nested CSQ with skip level correlations**

```
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:** Nested CSQ with skip level correlations are not supported by the legacy query optimizer.

• **CSQ with aggregate and inequality.** This example contains a CSQ with an inequality.

```
SELECT * FROM part p1 WHERE p1.p_retailprice =
(SELECT min(p_retailprice) FROM part p2 WHERE p2.p_brand <> p1.p_brand);
```
- CSQ that must return one row.

```sql
SELECT p_partkey,
    (SELECT p_retailprice FROM part p2 WHERE p2.p_brand = p1.p_brand)
FROM part p1;
```

**Queries that Contain Common Table Expressions**

GPORCA handles queries that contain the `WITH` clause. The `WITH` clause, also known as a common table expression (CTE), generates temporary tables that exist only for the query. This example query contains a CTE.

```sql
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;
```

As part of query optimization, GPORCA can push down predicates into a CTE. For example query, GPORCA pushes the equality predicates to the CTE.

```sql
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 can handle these types of CTEs:

- CTE that defines one or multiple tables. In this query, the CTE defines two tables.

```sql
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;
```

- Nested CTEs.

```sql
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;
```

**DML Operation Enhancements with GPORCA**

GPORCA contains enhancements for DML operations such as `INSERT`, `UPDATE`, and `DELETE`.

- A DML node in a query plan is a query plan operator.
  - Can appear anywhere in the plan, as a regular node (top slice only for now)
  - Can have consumers
  - `UPDATE` operations use the query plan operator `Split` and supports these operations:
    - `UPDATE` operations on the table distribution key columns.
• **UPDATE** operations on the table on the partition key column.

This example plan shows the **Split** operator.

```
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)
                    ->  Table Scan on dmltest
```

• New query plan operator **Assert** is used for constraints checking.

This example plan shows the **Assert** operator.

```
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)
                        ->  Table Scan on dmlsource
```

**Changed Behavior with the GPORCA**

There are changes to Greenplum Database behavior with the GPORCA optimizer enabled (the default) as compared to the legacy planner.

• **UPDATE** operations on distribution keys are allowed.
• **UPDATE** operations on partitioned keys are allowed.
• Queries against uniform partitioned tables are supported.
• Queries against partitioned tables that are altered to use an external table as a leaf child partition fall back to the legacy query optimizer.
• Except for **INSERT**, DML operations directly on partition (child table) of a partitioned table are not supported.

For the **INSERT** command, you can specify a leaf child table of the partitioned table to insert data into a partitioned table. An error is returned if the data is not valid for the specified leaf child table. Specifying a child table that is not a leaf child table is not supported.

• The command **CREATE TABLE AS** distributes table data randomly if the **DISTRIBUTED BY** clause is not specified and no primary or unique keys are specified.
• Non-deterministic updates not allowed. The following **UPDATE** command returns an error.

```
update r set b = r.b + 1 from s where r.a in (select a from s);
```

• Statistics are required on the root table of a partitioned table. The **ANALYZE** command generates statistics on both root and individual partition tables (leaf child tables). See the **ROOTPARTITION** clause for **ANALYZE** command.
• Additional Result nodes in the query plan:
  • Query plan **Assert** operator.
  • Query plan **Partition selector** operator.
  • Query plan **Split** operator.
• When running EXPLAIN, the query plan generated by GPORCA is different than the plan generated by the legacy query optimizer.
• Greenplum Database adds the log file message Planner produced plan when GPORCA is enabled and Greenplum Database falls back to the legacy query optimizer to generate the query plan.
• Greenplum Database issues a warning when statistics are missing from one or more table columns. When executing an SQL command with GPORCA, Greenplum Database issues a warning if the command performance could be improved by collecting statistics on a column or set of columns referenced by the command. The warning is issued on the command line and information is added to the Greenplum Database log file. For information about collecting statistics on table columns, see the ANALYZE command in the Greenplum Database Reference Guide.

GPORCA Limitations
There are limitations in Greenplum Database when using the default GPORCA optimizer. GPORCA and the legacy query optimizer currently coexist in Greenplum Database because GPORCA does not support all Greenplum Database features.
This section describes the limitations.
• Unsupported SQL Query Features
• Performance Regressions

Unsupported SQL Query Features
These are unsupported features when GPORCA is enabled (the default):
• Indexed expressions (an index defined as expression based on one or more columns of the table)
• PERCENTILE window function
• External parameters
• These types of partitioned tables:
  • Non-uniform partitioned tables.
  • Partitioned tables that have been altered to use an external table as a leaf child partition.
• SortMergeJoin (SMJ)
• Ordered aggregations
• These analytics extensions:
  • CUBE
  • Multiple grouping sets
• These scalar operators:
  • ROW
  • ROWCOMPARE
  • FIELDSELECT
• Multiple DISTINCT qualified aggregate functions
• Inverse distribution functions

Performance Regressions
The following features are known performance regressions that occur with GPORCA enabled:
• Short running queries - For GPORCA, short running queries might encounter additional overhead due to GPORCA enhancements for determining an optimal query execution plan.
• ANALYZE - For GPORCA, the ANALYZE command generates root partition statistics for partitioned tables. For the legacy optimizer, these statistics are not generated.
• DML operations - For GPORCA, DML enhancements including the support of updates on partition and distribution keys might require additional overhead.
Also, enhanced functionality of the features from previous versions could result in additional time required when GPORCA executes SQL statements with the features.

**Determining the Query Optimizer that is Used**

When GPORCA is enabled (the default), you can determine if Greenplum Database is using GPORCA or is falling back to the legacy query optimizer.

You can examine the `EXPLAIN` query plan for the query determine which query optimizer was used by Greenplum Database to execute the query:

- When GPORCA generates the query plan, the setting `optimizer=on` and GPORCA version are displayed at the end of the query plan. For example.

  | Settings: | optimizer=on |
  | Optimizer status: | PQO version 1.584 |

  When Greenplum Database falls back to the legacy optimizer to generate the plan, the setting `optimizer=on` and `legacy query optimizer` are displayed at the end of the query plan. For example.

  | Settings: | optimizer=on |
  | Optimizer status: | legacy query optimizer |

  When the server configuration parameter `OPTIMIZER` is `off`, these lines are displayed at the end of a query plan.

  | Settings: | optimizer=off |
  | Optimizer status: | legacy query optimizer |

- These plan items appear only in the `EXPLAIN` plan output generated by GPORCA. The items are not supported in a legacy optimizer query plan.

  - Assert operator
  - Sequence operator
  - DynamicIndexScan
  - DynamicTableScan
  - Table Scan

- When a query against a partitioned table is generated by GPORCA, the `EXPLAIN` plan displays only the number of partitions that are being eliminated is listed. The scanned partitions are not shown. The `EXPLAIN` plan generated by the legacy optimizer lists the scanned partitions.

  The log file contains messages that indicate which query optimizer was used. If Greenplum Database falls back to the legacy optimizer, a message with `NOTICE` information is added to the log file that indicates the unsupported feature. Also, the label `Planner produced plan:` appears before the query in the query execution log message when Greenplum Database falls back to the legacy optimizer.

  **Note:** You can configure Greenplum Database to display log messages on the `psql` command line by setting the Greenplum Database server configuration parameter `client_min_messages` to `LOG`. See the Greenplum Database Reference Guide for information about the parameter.

**Examples**

This example shows the differences for a query that is run against partitioned tables when GPORCA is enabled.

This `CREATE TABLE` statement creates a table with single level partitions:

```sql
CREATE TABLE sales (trans_id int, date date, amount decimal(9,2), region text)
```
DISTRIBUTED BY (trans_id)
PARTITION BY RANGE (date)
  (START (date '20160101')
   INCLUSIVE END (date '20170101')
   EXCLUSIVE EVERY (INTERVAL '1 month'),
 DEFAULT PARTITION outlying_dates);

This query against the table is supported by GPORCA and does not generate errors in the log file:

```
select * from sales ;
```

The EXPLAIN plan output lists only the number of selected partitions.

```
->  Partition Selector for sales (dynamic scan id: 1)  (cost=10.00..100.00
    rows=50 width=4)
    Partitions selected:  13 (out of 13)
```

If a query against a partitioned table is not supported by GPORCA, Greenplum Database falls back to the legacy optimizer. The EXPLAIN plan generated by the legacy optimizer lists the selected partitions. This example shows a part of the explain plan that lists some selected partitions.

```
->  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)
    ...
```

This example shows the log output when the Greenplum Database falls back to the legacy query optimizer from GPORCA.

When this query is run, Greenplum Database falls back to the legacy query optimizer.

```
explain select * from pg_class;
```

A message is added to the log file. The message contains this NOTICE information that indicates the reason GPORCA did not execute the query:

```
NOTICE,""Feature not supported by the Pivotal Query Optimizer: Queries on master-only tables"
```

**About Uniform Multi-level Partitioned Tables**

GPORCA supports queries on a multi-level partitioned (MLP) table if the MLP table is a *uniform partitioned table*. A multi-level partitioned table is a partitioned table that was created with the **SUBPARTITION** clause. A uniform partitioned table must meet these requirements.

- The partitioned table structure is uniform. Each partition node at the same level must have the same hierarchical structure.
- The partition key constraints must be consistent and uniform. At each subpartition level, the sets of constraints on the child tables created for each branch must match.

You can display information about partitioned tables in several ways, including displaying information from these sources:

- The **pg_partitions** system view contains information on the structure of a partitioned table.
- The **pg_constraint** system catalog table contains information on table constraints.
- The **psql** meta command `\d+ tablename` displays the table constraints for child leaf tables of a partitioned table.
**Example**

This CREATE TABLE command creates a uniform partitioned table.

```sql
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));
```

These are child tables and the partition hierarchy that are created for the table mlp. This hierarchy consists of one subpartition level that contains two branches.

<table>
<thead>
<tr>
<th>branch</th>
<th>constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>mlp_1_prt_11</td>
<td></td>
</tr>
<tr>
<td>mlp_1_prt_11_2_prt_usa</td>
<td>(region = 'usa')::text</td>
</tr>
<tr>
<td>mlp_1_prt_11_2_prt_europe</td>
<td>(region = 'europe')::text</td>
</tr>
<tr>
<td>mlp_1_prt_11_2_prt_asia</td>
<td>(region = 'asia')::text</td>
</tr>
<tr>
<td>mlp_1_prt_21</td>
<td></td>
</tr>
<tr>
<td>mlp_1_prt_21_2_prt_usa</td>
<td>(region = 'usa')::text</td>
</tr>
<tr>
<td>mlp_1_prt_21_2_prt_europe</td>
<td>(region = 'europe')::text</td>
</tr>
<tr>
<td>mlp_1_prt_21_2_prt_asia</td>
<td>(region = 'asia')::text</td>
</tr>
</tbody>
</table>

The hierarchy of the table is uniform, each partition contains a set of three child tables (subpartitions). The constraints for the region subpartitions are uniform, the set of constraints on the child tables for the branch table mlp_1_prt_11 are the same as the constraints on the child tables for the branch table mlp_1_prt_21.

As a quick check, this query displays the constraints for the partitions.

```sql
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;
```

<table>
<thead>
<tr>
<th>part</th>
<th>consrc</th>
</tr>
</thead>
<tbody>
<tr>
<td>mlp_1_prt_2_2_prt_usa</td>
<td>(region = 'usa')::text</td>
</tr>
<tr>
<td>mlp_1_prt_1_2_prt_usa</td>
<td>(region = 'usa')::text</td>
</tr>
<tr>
<td>mlp_1_prt_2_2_prt_europe</td>
<td>(region = 'europe')::text</td>
</tr>
<tr>
<td>mlp_1_prt_1_2_prt_europe</td>
<td>(region = 'europe')::text</td>
</tr>
<tr>
<td>mlp_1_prt_2_2_prt_usa</td>
<td>(region = 'usa')::text</td>
</tr>
<tr>
<td>mlp_1_prt_1_2_prt_usa</td>
<td>(region = 'usa')::text</td>
</tr>
<tr>
<td>mlp_1_prt_1_2_prt_asia</td>
<td>(year &gt;= 2006) AND (year &lt; 2011)</td>
</tr>
<tr>
<td>mlp_1_prt_1_2_prt_usa</td>
<td>(year &gt;= 2006) AND (year &lt; 2011)</td>
</tr>
</tbody>
</table>

**Note:** You will need modify the query for more complex partitioned tables. For example, the query does not account for table names in different schemas.

The consrc column displays constraints on the subpartitions. The set of region constraints for the subpartitions in mlp_1_prt_1 match the constraints for the subpartitions in mlp_1_prt_2. The constraints for year are inherited from the parent branch tables.
If you add a default partition to the example partitioned table with this command:

```
ALTER TABLE mlp ADD DEFAULT PARTITION def
```

The partitioned table remains a uniform partitioned table. The branch created for default partition contains three child tables and the set of constraints on the child tables match the existing sets of child table constraints.

In the above example, if you drop the subpartition `mlp_1_prt_21_2_prt_asia` and add another subpartition for the region `canada`, the constraints are no longer uniform.

```
ALTER TABLE mlp ALTER PARTITION FOR (RANK(2))
DROP PARTITION asia ;

ALTER TABLE mlp ALTER PARTITION FOR (RANK(2))
ADD PARTITION canada VALUES ('canada');
```

Also, if you add a partition `canada` under `mlp_1_prt_21`, the partitioning hierarchy is not uniform.

However, if you add the subpartition `canada` to both `mlp_1_prt_21` and `mlp_1_prt_11` the of the original partitioned table, it remains a uniform partitioned table.

**Note:** Only the constraints on the sets of partitions at a partition level must be the same. The names of the partitions can be different.

---

**Defining Queries**

Greenplum Database is based on the PostgreSQL implementation of the SQL standard.

This topic describes how to construct SQL queries in Greenplum Database.

- **SQL Lexicon**
- **SQL Value Expressions**

**SQL Lexicon**

SQL is a standard language for accessing databases. The language consists of elements that enable data storage, retrieval, analysis, viewing, manipulation, and so on. You use SQL commands to construct queries and commands that the Greenplum Database engine understands. SQL queries consist of a sequence of commands. Commands consist of a sequence of valid tokens in correct syntax order, terminated by a semicolon (`;`).

For more information about SQL commands, see the *Greenplum Database Reference Guide*.

Greenplum Database uses PostgreSQL’s structure and syntax, with some exceptions. For more information about SQL rules and concepts in PostgreSQL, see “SQL Syntax” in the PostgreSQL documentation.

**SQL Value Expressions**

SQL value expressions consist of one or more values, symbols, operators, SQL functions, and data. The expressions compare data or perform calculations and return a value as the result. Calculations include logical, arithmetic, and set operations.

The following are value expressions:
• An aggregate expression
• An array constructor
• A column reference
• A constant or literal value
• A correlated subquery
• A field selection expression
• A function call
• A new column value in an **INSERT** or **UPDATE**
• An operator invocation column reference
• A positional parameter reference, in the body of a function definition or prepared statement
• A row constructor
• A scalar subquery
• A search condition in a **WHERE** clause
• A target list of a **SELECT** command
• A type cast
• A value expression in parentheses, useful to group sub-expressions and override precedence
• A window expression

SQL constructs such as functions and operators are expressions but do not follow any general syntax rules. For more information about these constructs, see *Using Functions and Operators*.

**Column References**

A column reference has the form:

```
correlation.columnname
```

Here, *correlation* is the name of a table (possibly qualified with a schema name) or an alias for a table defined with a **FROM** clause or one of the keywords **NEW** or **OLD**. **NEW** and **OLD** can appear only in rewrite rules, but you can use other correlation names in any SQL statement. If the column name is unique across all tables in the query, you can omit the "correlation." part of the column reference.

**Positional Parameters**

Positional parameters are arguments to SQL statements or functions that you reference by their positions in a series of arguments. For example, $1 refers to the first argument, $2 to the second argument, and so on. The values of positional parameters are set from arguments external to the SQL statement or supplied when SQL functions are invoked. Some client libraries support specifying data values separately from the SQL command, in which case parameters refer to the out-of-line data values. A parameter reference has the form:

```
$number
```

For example:

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

Here, the $1 references the value of the first function argument whenever the function is invoked.
**Subscripts**

If an expression yields a value of an array type, you can extract a specific element of the array value as follows:

```
expression[subscript]
```

You can extract multiple adjacent elements, called an array slice, as follows (including the brackets):

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

Each subscript is an expression and yields an integer value.

Array expressions usually must be in parentheses, but you can omit the parentheses when the expression to be subscripted is a column reference or positional parameter. You can concatenate multiple subscripts when the original array is multidimensional. For example (including the parentheses):

```
mytable.arraycolumn[4]
```

```
mytable.two_d_column[17][34]
```

```
$1[10:42]
```

```
(arrayfunction(a,b))[42]
```

**Field Selection**

If an expression yields a value of a composite type (row type), you can extract a specific field of the row as follows:

```
expression.fieldname
```

The row expression usually must be in parentheses, but you can omit these parentheses when the expression to be selected from is a table reference or positional parameter. For example:

```
mytable.mycolumn
```

```
$1.somecolumn
```

```
(rowfunction(a,b)).col3
```

A qualified column reference is a special case of field selection syntax.

**Operator Invocations**

Operator invocations have the following possible syntaxes:

```
expression operator expression (binary infix operator)
```

```
operator expression (unary prefix operator)
```

```
expression operator (unary postfix operator)
```
Where *operator* is an operator token, one of the key words **AND**, **OR**, or **NOT**, or qualified operator name in the form:

```
OPERATOR(schema.operatorname)
```

Available operators and whether they are unary or binary depends on the operators that the system or user defines. For more information about built-in operators, see *Built-in Functions and Operators*.

**Function Calls**

The syntax for a function call is the name of a function (possibly qualified with a schema name), followed by its argument list enclosed in parentheses:

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

For example, the following function call computes the square root of 2:

```
sqrt(2)
```

See the *Greenplum Database Reference Guide* for lists of the built-in functions by category. You can add custom functions, too.

**Aggregate Expressions**

An aggregate expression applies an aggregate function across the rows that a query selects. An aggregate function performs a calculation on a set of values and returns a single value, such as the sum or average of the set of values. The syntax of an aggregate expression is one of the following:

- `aggregate_name(expression [, ... ])` — operates across all input rows for which the expected result value is non-null. **ALL** is the default.
- `aggregate_name(ALL expression [, ... ])` — operates identically to the first form because **ALL** is the default.
- `aggregate_name(DISTINCT expression [, ... ])` — operates across all distinct non-null values of input rows.
- `aggregate_name(*)` — operates on all rows with values both null and non-null. Generally, this form is most useful for the `count(*)` aggregate function.

Where *aggregate_name* is a previously defined aggregate (possibly schema-qualified) and *expression* is any value expression that does not contain an aggregate expression.

For example, `count(*)` yields the total number of input rows, `count(f1)` yields the number of input rows in which `f1` is non-null, and `count(distinct f1)` yields the number of distinct non-null values of `f1`.

For predefined aggregate functions, see *Built-in Functions and Operators*. You can also add custom aggregate functions.

Greenplum Database provides the **MEDIAN** aggregate function, which returns the fiftieth percentile of the **PERCENTILE_CONT** result and special aggregate expressions for inverse distribution functions as follows:

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

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

Currently you can use only these two expressions with the keyword **WITHIN GROUP**.

**Limitations of Aggregate Expressions**

The following are current limitations of the aggregate expressions:
Greenplum Database does not support the following keywords: ALL, DISTINCT, FILTER and OVER. See Table 46: Advanced Aggregate Functions for more details.

An aggregate expression can appear only in the result list or HAVING clause of a SELECT command. It is forbidden in other clauses, such as WHERE, because those clauses are logically evaluated before the results of aggregates form. This restriction applies to the query level to which the aggregate belongs.

When an aggregate expression appears in a subquery, the aggregate is normally evaluated over the rows of the subquery. If the aggregate's arguments contain only outer-level variables, the aggregate belongs to the nearest such outer level and evaluates over the rows of that query. The aggregate expression as a whole is then an outer reference for the subquery in which it appears, and the aggregate expression acts as a constant over any one evaluation of that subquery. See Scalar Subqueries and Table 44: Built-in functions and operators.

Greenplum Database does not support DISTINCT with multiple input expressions.

**Window Expressions**

Window expressions allow application developers to more easily compose complex online analytical processing (OLAP) queries using standard SQL commands. For example, with window expressions, users can calculate moving averages or sums over various intervals, reset aggregations and ranks as selected column values change, and express complex ratios in simple terms.

A window expression represents the application of a window function applied to a window frame, which is defined in a special OVER() clause. A window partition is a set of rows that are grouped together to apply a window function. Unlike aggregate functions, which return a result value for each group of rows, window functions return a result value for every row, but that value is calculated with respect to the rows in a particular window partition. If no partition is specified, the window function is computed over the complete intermediate result set.

The syntax of a window expression is:

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

Where `window_function` is one of the functions listed in Table 45: Window functions, `expression` is any value expression that does not contain a window expression, and `window_specification` is:

```
[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 }]
```

and where `window_frame_bound` can be one of:

- UNBOUNDED PRECEDING
- expression PRECEDING
- CURRENT ROW
- expression FOLLOWING
- UNBOUNDED FOLLOWING

A window expression can appear only in the select list of a SELECT command. For example:

```sql
SELECT count(*) OVER(PARTITION BY customer_id), * FROM sales;
```
The OVER clause differentiates window functions from other aggregate or reporting functions. The OVER clause defines the window specification to which the window function is applied. A window specification has the following characteristics:

- The PARTITION BY clause defines the window partitions to which the window function is applied. If omitted, the entire result set is treated as one partition.
- The ORDER BY clause defines the expression(s) for sorting rows within a window partition. The ORDER BY clause of a window specification is separate and distinct from the ORDER BY clause of a regular query expression. The ORDER BY clause is required for the window functions that calculate rankings, as it identifies the measure(s) for the ranking values. For OLAP aggregations, the ORDER BY clause is required to use window frames (the ROWS | RANGE clause).

**Note:** Columns of data types without a coherent ordering, such as time, are not good candidates for use in the ORDER BY clause of a window specification. Time, with or without a specified time zone, lacks a coherent ordering because addition and subtraction do not have the expected effects. For example, the following is not generally true: \( x::time < x::time + '2 hour'::interval \)

- The ROWS/RANGE clause defines a window frame for aggregate (non-ranking) window functions. A window frame defines a set of rows within a window partition. When a window frame is defined, the window function computes on the contents of this moving frame rather than the fixed contents of the entire window partition. Window frames are row-based (ROWS) or value-based (RANGE).

**Type Casts**

A type cast specifies a conversion from one data type to another. Greenplum Database accepts two equivalent syntaxes for type casts:

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

The CAST syntax conforms to SQL; the syntax with :: is historical PostgreSQL usage.

A cast applied to a value expression of a known type is a run-time type conversion. The cast succeeds only if a suitable type conversion function is defined. This differs from the use of casts with constants. A cast applied to a string literal represents the initial assignment of a type to a literal constant value, so it succeeds for any type if the contents of the string literal are acceptable input syntax for the data type.

You can usually omit an explicit type cast if there is no ambiguity about the type a value expression must produce; for example, when it is assigned to a table column, the system automatically applies a type cast. The system applies automatic casting only to casts marked "OK to apply implicitly" in system catalogs. Other casts must be invoked with explicit casting syntax to prevent unexpected conversions from being applied without the user's knowledge.

**Scalar Subqueries**

A scalar subquery is a SELECT query in parentheses that returns exactly one row with one column. Do not use a SELECT query that returns multiple rows or columns as a scalar subquery. The query runs and uses the returned value in the surrounding value expression. A correlated scalar subquery contains references to the outer query block.

**Correlated Subqueries**

A correlated subquery (CSQ) is a SELECT query with a WHERE clause or target list that contains references to the parent outer clause. CSQs efficiently express results in terms of results of another query. Greenplum Database supports correlated subqueries that provide compatibility with many existing applications. A CSQ is a scalar or table subquery, depending on whether it returns one or multiple rows. Greenplum Database does not support correlated subqueries with skip-level correlations.
Correlated Subquery Examples

Example 1 – Scalar correlated subquery

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

Example 2 – Correlated EXISTS subquery

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

Greenplum Database uses one of the following methods to run CSQs:

- Unnest the CSQ into join operations – This method is most efficient, and it is how Greenplum Database runs most CSQs, including queries from the TPC-H benchmark.
- Run the CSQ on every row of the outer query – This method is relatively inefficient, and it is how Greenplum Database runs queries that contain CSQs in the `SELECT` list or are connected by OR conditions.

The following examples illustrate how to rewrite some of these types of queries to improve performance.

Example 3 - CSQ in the Select List

**Original Query**

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

Rewrite this query to perform an inner join with `t1` first and then perform a left join with `t1` again. The rewrite applies for only an equijoin in the correlated condition.

**Rewritten Query**

```sql
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);
```

Example 4 - CSQs connected by OR Clauses

**Original Query**

```sql
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)
```

Rewrite this query to separate it into two parts with a union on the OR conditions.

**Rewritten Query**

```sql
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)

To view the query plan, use `EXPLAIN SELECT` or `EXPLAIN ANALYZE SELECT`. Subplan nodes in the query plan indicate that the query will run on every row of the outer query, and the query is a candidate for rewriting. For more information about these statements, see *Query Profiling*.

### Advanced Table Functions

Greenplum Database supports table functions with `TABLE` value expressions. You can sort input rows for advanced table functions with an `ORDER BY` clause. You can redistribute them with a `SCATTER BY` clause to specify one or more columns or an expression for which rows with the specified characteristics are available to the same process. This usage is similar to using a `DISTRIBUTED BY` clause when creating a table, but the redistribution occurs when the query runs.

**Note:** Based on the distribution of data, Greenplum Database automatically parallelizes table functions with `TABLE` value parameters over the nodes of the cluster.

The following command uses the `TABLE` function with the `SCATTER BY` clause in the `gptext.index()` function to populate the index `mytest.articles` with data from the `messages` table:

```sql
SELECT * FROM gptext.index(TABLE(SELECT * FROM messages SCATTER BY distrib_id), 'mytest.articles');
```

For information about the function `gptext.index()`, see the Pivotal GPText documentation.

### Array Constructors

An array constructor is an expression that builds an array value from values for its member elements. A simple array constructor consists of the key word `ARRAY`, a left square bracket `[`, one or more expressions separated by commas for the array element values, and a right square bracket `]`. For example:

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

The array element type is the common type of its member expressions, determined using the same rules as for `UNION` or `CASE` constructs.

You can build multidimensional array values by nesting array constructors. In the inner constructors, you can omit the keyword `ARRAY`. For example, the following two `SELECT` statements produce the same result:

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

Since multidimensional arrays must be rectangular, inner constructors at the same level must produce sub-arrays of identical dimensions.

Multidimensional array constructor elements are not limited to a sub-`ARRAY` construct; they are anything that produces an array of the proper kind. For example:

```
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
----------------------------------
```
You can construct an array from the results of a subquery. Write the array constructor with the keyword `ARRAY` followed by a subquery in parentheses. For example:

```
SELECT ARRAY(SELECT oid FROM pg_proc WHERE proname LIKE 'bytea%');
```

The subquery must return a single column. The resulting one-dimensional array has an element for each row in the subquery result, with an element type matching that of the subquery's output column. The subscripts of an array value built with `ARRAY` always begin with 1.

**Row Constructors**

A row constructor is an expression that builds a row value (also called a composite value) from values for its member fields. For example,

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

Row constructors have the syntax `rowvalue.*`, which expands to a list of the elements of the row value, as when you use the syntax `.*` at the top level of a `SELECT` list. For example, if table `t` has columns `f1` and `f2`, the following queries are the same:

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

By default, the value created by a `ROW` expression has an anonymous record type. If necessary, it can be cast to a named composite type — either the row type of a table, or a composite type created with `CREATE TYPE AS`. To avoid ambiguity, you can explicitly cast the value if necessary. For example:

```
CREATE TABLE mytable(f1 int, f2 float, f3 text);
CREATE FUNCTION getf1(mytable) RETURNS int AS 'SELECT $1.f1'
LANGUAGE SQL;
```

In the following query, you do not need to cast the value because there is only one `getf1()` function and therefore no ambiguity:

```
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;
```

Now we need a cast to indicate which function to call:

```
SELECT getf1(ROW(1,2.5,'this is a test'))::mytable);
```

```
ERROR:  function getf1(record) is not unique
```

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

```
getf1
-------
1
```

```
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;
```

```
SELECT getf1(ROW(11,'this is a test',2.5) AS myrowtype));
```

```
getf1
-------
1
```
You can use row constructors to build composite values to be stored in a composite-type table column or to be passed to a function that accepts a composite parameter.

**Expression Evaluation Rules**

The order of evaluation of subexpressions is undefined. The inputs of an operator or function are not necessarily evaluated left-to-right or in any other fixed order.

If you can determine the result of an expression by evaluating only some parts of the expression, then other subexpressions might not be evaluated at all. For example, in the following expression:

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

`somefunc()` would probably not be called at all. The same is true in the following expression:

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

This is not the same as the left-to-right evaluation order that Boolean operators enforce in some programming languages.

Do not use functions with side effects as part of complex expressions, especially in `WHERE` and `HAVING` clauses, because those clauses are extensively reprocessed when developing an execution plan. Boolean expressions (`AND`/`OR`/`NOT` combinations) in those clauses can be reorganized in any manner that Boolean algebra laws allow.

Use a `CASE` construct to force evaluation order. The following example is an untrustworthy way to avoid division by zero in a `WHERE` clause:

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

The following example shows a trustworthy evaluation order:

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

This `CASE` construct usage defeats optimization attempts; use it only when necessary.

**WITH Queries (Common Table Expressions)**

The `WITH` clause of the `SELECT` command provides a way to write subqueries for use in a larger `SELECT` query.

The subqueries, which are often referred to as Common Table Expressions or CTEs, can be thought of as defining temporary tables that exist just for the query. One use of this feature is to break down complicated queries into simpler parts. This example query displays per-product sales totals in only the top sales regions:

```sql
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,
```
The query could have been written without the WITH clause, but would have required two levels of nested sub-SELECTs. It is easier to follow with the WITH clause.

The RECURSIVE keyword can be enabled by setting the server configuration parameter gp_recursive_cte_prototype to on.

**Warning:** The RECURSIVE keyword is an experimental feature and is not intended for use in a production environment. Experimental features are subject to change without notice in future releases.

The optional RECURSIVE keyword changes WITH accomplishes things not otherwise possible in standard SQL. Using RECURSIVE, a query in the WITH clause can refer to its own output. This is a simple example that computes the sum the integers from 1 through 100:

```sql
WITH RECURSIVE t(n) AS (  
VALUES (1)  
UNION ALL  
SELECT n+1 FROM t WHERE n < 100  
)  
SELECT sum(n) FROM t;
```

The required form of a recursive WITH clause is a non-recursive term, followed by a UNION (or UNION ALL), and then a recursive term, where only the recursive term can contain a reference to the query output.

```sql
non_recursive_term UNION [ ALL ] recursive_term
```

A recursive WITH query is executed as follows:

1. Evaluate the non-recursive term. For UNION (but not UNION ALL), discard duplicate rows. Include all remaining rows in the result of the recursive query, and also place them in a temporary working table.

2. As long as the working table is not empty, repeat these steps:

   a. Evaluate the recursive term, substituting the current contents of the working table for the recursive self-reference. For UNION (but not UNION ALL), discard duplicate rows and rows that duplicate any previous result row. Include all remaining rows in the result of the recursive query, and also place them in a temporary intermediate table.

   b. Replace the contents of the working table with the contents of the intermediate table, then empty the intermediate table.

   **Note:** Strictly speaking, the process is iteration not recursion, but RECURSIVE is the terminology chosen by the SQL standards committee.

Recursive WITH queries are typically used to deal with hierarchical or tree-structured data. An example is this query to find all the direct and indirect sub-parts of a product, given only a table that shows immediate inclusions:

```sql
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

When working with recursive WITH queries, you must ensure that the recursive part of the query eventually returns no tuples, or else the query loops indefinitely. In the example that computes the sum the integers, the working table contains a single row in each step, and it takes on the values from 1 through 100 in successive steps. In the 100th step, there is no output because of the WHERE clause, and the query terminates.

For some queries, using UNION instead of UNION ALL can ensure that the recursive part of the query eventually returns no tuples by discarding rows that duplicate previous output rows. However, often a cycle does not involve output rows that are complete duplicates: it might sufficient to check just one or a few fields to see if the same point has been reached before. The standard method for handling such situations is to compute an array of the visited values. For example, consider the following query that searches a table graph using a link field:

```
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;
```

This query loops if the link relationships contain cycles. Because the query requires a depth output, changing UNION ALL to UNION does not eliminate the looping. Instead the query needs to recognize whether it has reached the same row again while following a particular path of links. This modified query adds two columns path and cycle to the loop-prone query:

```
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;
```

Aside from detecting cycles, the array value of path is useful in its own right since it represents the path taken to reach any particular row.

In the general case where more than one field needs to be checked to recognize a cycle, an array of rows can be used. For example, if we needed to compare fields f1 and 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: Omit the `ROW()` syntax in the case where only one field needs to be checked to recognize a cycle. This uses a simple array rather than a composite-type array to be used, gaining efficiency.

Tip: The recursive query evaluation algorithm produces its output in breadth-first search order. You can display the results in depth-first search order by making the outer query `ORDER BY` a path column constructed in this way.

A helpful technique for testing a query when you are not certain if it might loop indefinitely is to place a `LIMIT` in the parent query. For example, this query would loop forever without the `LIMIT` clause:

```sql
WITH RECURSIVE t(n) AS (  
    SELECT 1  
    UNION ALL  
    SELECT n+1 FROM t  
)  
SELECT n FROM t LIMIT 100;
```

The technique works because the recursive `WITH` implementation evaluates only as many rows of a `WITH` query as are actually fetched by the parent query. Using this technique in production is not recommended, because other systems might work differently. Also, the technique might not work if the outer query sorts the recursive `WITH` results or join the results to another table.

### Using Functions and Operators

Description of user-defined and built-in functions and operators in Greenplum Database.

- Using Functions in Greenplum Database
- User-Defined Functions
- Built-in Functions and Operators
- Window Functions
- Advanced Aggregate Functions

### Using Functions in Greenplum Database

Table 43: Functions in Greenplum Database

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Greenplum Support</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMMUTABLE</td>
<td>Yes</td>
<td>Relies only on information directly in its argument list. Given the same argument values, always returns the same result.</td>
<td></td>
</tr>
<tr>
<td>STABLE</td>
<td>Yes, in most cases</td>
<td>Within a single table scan, returns the same result for same argument values, but results change across SQL statements.</td>
<td>Results depend on database lookups or parameter values. current_timestamp family of functions is STABLE; values do not change within an execution.</td>
</tr>
<tr>
<td>VOLATILE</td>
<td>Restricted</td>
<td>Function values can change within a single table scan. For example: random(), timestamp().</td>
<td>Any function with side effects is volatile, even if its result is predictable. For example: setval().</td>
</tr>
</tbody>
</table>
Refer to the PostgreSQL Function Volatility Categories documentation for additional information about the Greenplum Database function volatility classifications.

In Greenplum Database, data is divided up across segments — each segment is a distinct PostgreSQL database. To prevent inconsistent or unexpected results, do not execute functions classified as VOLATILE at the segment level if they contain SQL commands or modify the database in any way. For example, functions such as `setval()` are not allowed to execute on distributed data in Greenplum Database because they can cause inconsistent data between segment instances.

To ensure data consistency, you can safely use VOLATILE and STABLE functions in statements that are evaluated on and run from the master. For example, the following statements run on the master (statements without a FROM clause):

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

If a statement has a FROM clause containing a distributed table and the function in the FROM clause returns a set of rows, the statement can run on the segments:

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

Greenplum Database does not support functions that return a table reference (rangeFuncs) or functions that use the refCursor datatype.

**Function Volatility and Plan Caching**

There is relatively little difference between the STABLE and IMMUTABLE function volatility categories for simple interactive queries that are planned and immediately executed. It does not matter much whether a function is executed once during planning or once during query execution startup. But there is a big difference when you save the plan and reuse it later. If you mislabel a function IMMUTABLE, Greenplum Database may prematurely fold it to a constant during planning, possibly reusing a stale value during subsequent execution of the plan. You may run into this hazard when using PREPARED statements, or when using languages such as PL/pgSQL that cache plans.

**User-Defined Functions**

Greenplum Database supports user-defined functions. See Extending SQL in the PostgreSQL documentation for more information.

Use the CREATE FUNCTION statement to register user-defined functions that are used as described in Using Functions in Greenplum Database. By default, user-defined functions are declared as VOLATILE, so if your user-defined function is IMMUTABLE or STABLE, you must specify the correct volatility level when you register your function.

When you create user-defined functions, avoid using fatal errors or destructive calls. Greenplum Database may respond to such errors with a sudden shutdown or restart.

In Greenplum Database, the shared library files for user-created functions must reside in the same library path location on every host in the Greenplum Database array (masters, segments, and mirrors).

You can also create and execute anonymous code blocks that are written in a Greenplum Database procedural language such as PL/pgSQL. The anonymous blocks run as transient anonymous functions. For information about creating and executing anonymous blocks, see the DO command.

**Built-in Functions and Operators**

The following table lists the categories of built-in functions and operators supported by PostgreSQL. All functions and operators are supported in Greenplum Database as in PostgreSQL with the exception of STABLE and VOLATILE functions, which are subject to the restrictions noted in Using Functions in
Greenplum Database. See the Functions and Operators section of the PostgreSQL documentation for more information about these built-in functions and operators.

Greenplum Database includes JSON processing functions that manipulate values the JSON data type. For information about JSON data, see Working with JSON Data.

Table 44: Built-in functions and operators

<table>
<thead>
<tr>
<th>Operator/Function Category</th>
<th>VOLATILE Functions</th>
<th>STABLE Functions</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical Operators</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comparison Operators</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mathematical Functions and Operators</td>
<td>random setseed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String Functions and Operators</td>
<td>All built-in conversion functions</td>
<td>convert pg_client_encoding</td>
<td></td>
</tr>
<tr>
<td>Binary String Functions and Operators</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit String Functions and Operators</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pattern Matching</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Type Formatting Functions</td>
<td>to_char to_timestamp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date/Time Functions and Operators</td>
<td>timeofday</td>
<td>age current_date current_time current_timestamp localtime localtime_timestamp now</td>
<td></td>
</tr>
<tr>
<td>Enum Support Functions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometric Functions and Operators</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network Address Functions and Operators</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence Manipulation Functions</td>
<td>nextval() setval()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conditional Expressions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Array Functions and Operators</td>
<td>All array functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aggregate Functions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operator/Function Category</td>
<td>VOLATILE Functions</td>
<td>STABLE Functions</td>
<td>Restrictions</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------</td>
<td>-----------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Subquery Expressions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Row and Array Comparisons</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set Returning Functions</td>
<td>generate_series</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Information Functions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>All session information functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>All access privilege inquiry functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>All schema visibility inquiry functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>All system catalog information functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>All comment information functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>All transaction ids and snapshots</td>
<td></td>
</tr>
<tr>
<td>System Administration Functions</td>
<td></td>
<td>current_setting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>set_config</td>
<td>All database object size functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pg_cancel_backend</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pg_reload_conf</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pg_rotate_logfile</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pg_start_backup</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pg_stop_backup</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pg_size_pretty</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pg_ls_dir</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pg_read_file</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pg_stat_file</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The function `pg_column_size` displays bytes required to store the value, possibly with TOAST compression.
<table>
<thead>
<tr>
<th>Operator/Function Category</th>
<th>VOLATILE Functions</th>
<th>STABLE Functions</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Functions and function-like expressions</td>
<td>cursor_to_xml(cursor refcursor, count int, nulls boolean, tableforest boolean, targetns text)</td>
<td>database_to_xml(nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cursor_to_xmlschema(cursor refcursor, nulls boolean, tableforest boolean, targetns text)</td>
<td>database_to_xmlschema(nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>database_to_xml_and_xmlschema(nulls boolean, tableforest boolean, targetns text)</td>
<td>database_to_xml_and_xmlschema(nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>query_to_xml(query text, nulls boolean, tableforest boolean, targetns text)</td>
<td>query_to_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>query_to_xml_and_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)</td>
<td>query_to_xml_and_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>schema_to_xml(schema name, nulls boolean, tableforest boolean, targetns text)</td>
<td>schema_to_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>schema_to_xml_and_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)</td>
<td>schema_to_xml_and_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table_to_xml(tbl regclass, nulls boolean, tableforest boolean, targetns text)</td>
<td>table_to_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td>Operator/Function Category</td>
<td>VOLATILE Functions</td>
<td>STABLE Functions</td>
<td>Restrictions</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------</td>
<td>------------------</td>
<td>--------------</td>
</tr>
<tr>
<td></td>
<td>table_to_xml_and_xmlschema(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) xmlparse ( { DOCUMENT</td>
<td>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</td>
<td>no value [, standalone yes</td>
</tr>
</tbody>
</table>

**Window Functions**

The following built-in window functions are Greenplum extensions to the PostgreSQL database. All window functions are *immutable*. For more information about window functions, see *Window Expressions*. 
### Table 45: Window functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Full Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cume_dist()</td>
<td>double precision</td>
<td><code>CUME_DIST() OVER ( [PARTITION BY expr ] ORDER BY expr )</code></td>
<td>Calculates the cumulative distribution of a value in a group of values. Rows with equal values always evaluate to the same cumulative distribution value.</td>
</tr>
<tr>
<td>dense_rank()</td>
<td>bigint</td>
<td><code>DENSE_RANK () OVER ( [PARTITION BY expr ] ORDER BY expr )</code></td>
<td>Computes the rank of a row in an ordered group of rows without skipping rank values. Rows with equal rank values are given the same rank value.</td>
</tr>
<tr>
<td>first_value(expr)</td>
<td>same as input expr</td>
<td>`FIRST_VALUE( expr ) OVER ( [PARTITION BY expr ] ORDER BY expr [ROWS</td>
<td>RANGE frame_expr] )`</td>
</tr>
<tr>
<td>lag(expr [,offset] [,default])</td>
<td>same as input expr type</td>
<td><code>LAG( expr [, offset] [, default] ) OVER ( [PARTITION BY expr ] ORDER BY expr )</code></td>
<td>Provides access to more than one row of the same table without doing a self join. Given a series of rows returned from a query and a position of the cursor, LAG provides access to a row at a given physical offset prior to that position. The default offset is 1. default sets the value that is returned if the offset goes beyond the scope of the window. If default is not specified, the default value is null.</td>
</tr>
<tr>
<td>last_value(expr)</td>
<td>same as input expr</td>
<td>`LAST_VALUE(expr) OVER ( [PARTITION BY expr ] ORDER BY expr [ROWS</td>
<td>RANGE frame_expr] )`</td>
</tr>
<tr>
<td>Function</td>
<td>Return Type</td>
<td>Full Syntax</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>lead(expr [,offset] [,default])</td>
<td>same as input expr type</td>
<td>LEAD(expr [,offset] [,exprdefault]) OVER ( [PARTITION BY expr] ORDER BY expr )</td>
<td>Provides access to more than one row of the same table without doing a self join. Given a series of rows returned from a query and a position of the cursor, lead provides access to a row at a given physical offset after that position. If offset is not specified, the default offset is 1. default sets the value that is returned if the offset goes beyond the scope of the window. If default is not specified, the default value is null.</td>
</tr>
<tr>
<td>ntile(expr)</td>
<td>bigint</td>
<td>NTILE(expr) OVER ( [PARTITION BY expr] ORDER BY expr )</td>
<td>Divides an ordered data set into a number of buckets (as defined by expr) and assigns a bucket number to each row.</td>
</tr>
<tr>
<td>percent_rank()</td>
<td>double precision</td>
<td>PERCENT_RANK () OVER ( [PARTITION BY expr] ORDER BY expr )</td>
<td>Calculates the rank of a hypothetical row (R) minus 1, divided by 1 less than the number of rows being evaluated (within a window partition).</td>
</tr>
<tr>
<td>rank()</td>
<td>bigint</td>
<td>RANK () OVER ( [PARTITION BY expr] ORDER BY expr )</td>
<td>Calculates the rank of a row in an ordered group of values. Rows with equal values for the ranking criteria receive the same rank. The number of tied rows are added to the rank number to calculate the next rank value. Ranks may not be consecutive numbers in this case.</td>
</tr>
<tr>
<td>row_number()</td>
<td>bigint</td>
<td>ROW_NUMBER () OVER ( [PARTITION BY expr] ORDER BY expr )</td>
<td>Assigns a unique number to each row to which it is applied (either each row in a window partition or each row of the query).</td>
</tr>
</tbody>
</table>

**Advanced Aggregate Functions**

The following built-in advanced aggregate functions are Greenplum extensions of the PostgreSQL database. These functions are *immutable*. 
**Note:** The Greenplum MADlib Extension for Analytics provides additional advanced functions to perform statistical analysis and machine learning with Greenplum Database data. See Greenplum MADlib Extension for Analytics in the Greenplum Database Reference Guide.

Table 46: Advanced Aggregate Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Full Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEDIAN (expr)</td>
<td>timestamp, timestamptz, interval, float</td>
<td>MEDIAN (expression)</td>
<td>Can take a two-dimensional array as input. Treats such arrays as matrices.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Example:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SELECT department_id, MEDIAN(salary) FROM employees GROUP BY department_id;</td>
<td></td>
</tr>
<tr>
<td>PERCENTILE_CONT (expr) WITHIN GROUP (ORDER BY expr [DESC/ASC])</td>
<td>timestamp, timestamptz, interval, float</td>
<td>PERCENTILE_CONT(percentage) WITHIN GROUP (ORDER BY expression)</td>
<td>Performs an inverse distribution function that assumes a continuous distribution model. It takes a percentile value and a sort specification and returns the same datatype as the numeric datatype of the argument. This returned value is a computed result after performing linear interpolation. Null are ignored in this calculation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Example:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SELECT department_id, PERCENTILE_CONT (0.5) WITHIN GROUP (ORDER BY salary DESC) &quot;Median_cont&quot;; FROM employees GROUP BY department_id;</td>
<td></td>
</tr>
<tr>
<td>PERCENTILE_DISC (expr) WITHIN GROUP (ORDER BY expr [DESC/ASC])</td>
<td>timestamp, timestamptz, interval, float</td>
<td>PERCENTILE_DISC(percentage) WITHIN GROUP (ORDER BY expression)</td>
<td>Performs an inverse distribution function that assumes a discrete distribution model. It takes a percentile value and a sort specification. This returned value is an element from the set. Null are ignored in this calculation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Example:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SELECT department_id, PERCENTILE_DISC (0.5) WITHIN GROUP (ORDER BY salary DESC) &quot;Median_desc&quot;; FROM employees GROUP BY department_id;</td>
<td></td>
</tr>
</tbody>
</table>
Working with JSON Data

Greenplum Database supports the `json` data type that stores JSON (JavaScript Object Notation) data.

Greenplum Database supports JSON as specified in the RFC 7159 document and enforces data validity according to the JSON rules. There are also JSON-specific functions and operators available for `json` data. See JSON Functions and Operators.

This section contains the following topics:

- About JSON Data
- JSON Input and Output Syntax
- Designing JSON documents
- JSON Functions and Operators

About JSON Data

When Greenplum Database stores data as `json` data type, an exact copy of the input text is stored and the JSON processing functions reparse the data on each execution.

- Semantically-insignificant white space between tokens is retained, as well as the order of keys within JSON objects.
- All key/value pairs are kept even if a JSON object contains duplicate keys. For duplicate keys, JSON processing functions consider the last value as the operative one.

Greenplum Database allows only one character set encoding per database. It is not possible for the `json` type to conform rigidly to the JSON specification unless the database encoding is UTF8. Attempts to include characters that cannot be represented in the database encoding will fail. Characters that can be represented in the database encoding but not in UTF8 are allowed.
The RFC 7159 document permits JSON strings to contain Unicode escape sequences denoted by \uXXXX. For the json type, the Greenplum Database input function allows Unicode escapes regardless of the database encoding and checks Unicode escapes only for syntactic correctness (a \u followed by four hex digits).

**Note:** Many of the JSON processing functions described in JSON Functions and Operators convert Unicode escapes to regular characters. The functions throw an error for characters that cannot be represented in the database encoding. You should avoid mixing Unicode escapes in JSON with a non-UTF8 database encoding, if possible.

**JSON Input and Output Syntax**

The input and output syntax for the json data type is as specified in RFC 7159.

The following are all valid json expressions:

-- Simple scalar/primitive value
-- Primitive values can be numbers, quoted strings, true, false, or null
SELECT '5'::json;

-- Array of zero or more elements (elements need not be of same type)
SELECT '[1, 2, "foo", null]'::json;

-- Object containing pairs of keys and values
-- Note that object keys must always be quoted strings
SELECT '{"bar": "baz", "balance": 7.77, "active": false}'::json;

-- Arrays and objects can be nested arbitrarily
SELECT '{"foo": [true, "bar"], "tags": {"a": 1, "b": null}}'::json;

**Designing JSON documents**

Representing data as JSON can be considerably more flexible than the traditional relational data model, which is compelling in environments where requirements are fluid. It is quite possible for both approaches to co-exist and complement each other within the same application. However, even for applications where maximal flexibility is desired, it is still recommended that JSON documents have a somewhat fixed structure. The structure is typically unenforced (though enforcing some business rules declaratively is possible), but having a predictable structure makes it easier to write queries that usefully summarize a set of "documents" (datums) in a table.

JSON data is subject to the same concurrency-control considerations as any other data type when stored in a table. Although storing large documents is practicable, keep in mind that any update acquires a row-level lock on the whole row. Consider limiting JSON documents to a manageable size in order to decrease lock contention among updating transactions. Ideally, JSON documents should each represent an atomic datum that business rules dictate cannot reasonably be further subdivided into smaller datums that could be modified independently.

**JSON Functions and Operators**

Built-in functions and operators that create and manipulate JSON data.

- **JSON Operators**
- **JSON Creation Functions**
- **JSON Processing Functions**

  **Note:** For json values, all key/value pairs are kept even if a JSON object contains duplicate keys. For duplicate keys, JSON processing functions consider the last value as the operative one.

**JSON Operators**

This table describes the operators that are available for use with the json data type.
Table 47: json Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Right Operand Type</th>
<th>Description</th>
<th>Example</th>
<th>Example Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-&gt; int</td>
<td>Get JSON array element (indexed from zero).</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
'[{"a":"foo"},
{"b":"bar"},
{"c":"baz"}]:json- >2 |
| -> text  | Get JSON object field by key. | 
'{"a":
{"b":"foo"}}':json- >'a' |
| ->> int  | Get JSON array element as text. | 
'[{1,2,3}]:json->>2 |
| ->> text | Get JSON object field as text. | 
'{"a":1,"b":2}':json->>'b' |
| #> text[] | Get JSON object at specified path. | 
'{"a": {"b":{"c": "foo"}}}:json#>"a,b"' |
| #>> text[] | Get JSON object at specified path as text. | 
'{"a":[1,2,3],"b": [4,5,6}]:json#>>'[a,2]' |

JSON Creation Functions

This table describes the functions that create json values.

Table 48: JSON Creation Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
<th>Example Result</th>
</tr>
</thead>
</table>
| array_to_json(anyarray [, pretty_bool]) | Returns the array as a JSON array. A Greenplum Database multidimensional array becomes a JSON array of arrays. Line feeds are added between dimension 1 elements if pretty_bool is true. | array_to_json('{{1,5},
{99,100}}':int[]) | [[1,5],[99,100]] |
| row_to_json(record [, pretty_bool]) | Returns the row as a JSON object. Line feeds are added between level 1 elements if pretty_bool is true. | row_to_json(row(1,'foo')) | {"f1":1,"f2":"foo"} |

JSON Processing Functions

This table describes the functions that process json values.
<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Description</th>
<th>Example</th>
<th>Example Result</th>
</tr>
</thead>
</table>
| `json_each(json)`                | setof key text, value json | Expands the outermost JSON object into a set of key/value pairs.            | `select * from json_each('"a":"foo", "b":"bar"')`                    | `key | value
     -----+-------
     a   | "foo”
     b   | "bar"` |
| `json_each_text(json)`           | setof key text, value text | Expands the outermost JSON object into a set of key/value pairs. The returned values are of type text. | `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[])` | json | Returns the JSON value specified to by path_elems. Equivalent to `#>` operator. | `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[])` | text | Returns the JSON value specified to by path_elems as text. Equivalent to `#>>` operator. | `json_extract_path_text('"f2":{"f3":1},"f4":{"f5":99,"f6":"foo"}",'f4','f6')` | `foo` |
| `json_object_keys(json)`         | setof text          | Returns set of keys in the outermost JSON object.                           | `json_object_keys('"f1":"abc","f2":{"a":1,"b":2})'`                | `json_object_keys
     ------------------
     f1
     f2` |
| `json_populate_record(base anyelement, from_json json)` | anyelement         | Expands the object in from_json to a row whose columns match the record type defined by base. See Note. | `select * from json_populate_record(null::myrowtype, 'a":1,"b":2')` | `a | b
     -----
     1 | 2` |
| `json_populate_recordset(base anyelement, from_json json)` | setof anyelement    | Expands the outermost array of objects in from_json to a set of rows whose columns match the record type defined by base. See Note. | `select * from json_populate_recordset(null::myrowtype, 'a":1,"b":2), ("a":3,"b":4)]'` | `a | b
     -----
     1 | 2
     3 | 4` |
<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Description</th>
<th>Example</th>
<th>Example Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>json_array_elements(json)</code></td>
<td>setof json</td>
<td>Expands a JSON array to a set of</td>
<td><code>select * from</code></td>
<td>value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JSON values.</td>
<td><code>json_array_elements('[1,true,[2,false]]')</code></td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[2,false]</td>
</tr>
</tbody>
</table>

**Note:** Many of these functions and operators convert Unicode escapes in JSON strings to regular characters. The functions throw an error for characters that cannot be represented in the database encoding.

For `json_populate_record` and `json_populate_recordset`, type coercion from JSON is best effort and might not result in desired values for some types. JSON keys are matched to identical column names in the target row type. JSON fields that do not appear in the target row type are omitted from the output, and target columns that do not match any JSON field return `NULL`.

---

**Working with XML Data**

Greenplum Database supports the `xml` data type that stores XML data.

The `xml` data type checks the input values for well-formedness, providing an advantage over simply storing XML data in a text field. Additionally, support functions allow you to perform type-safe operations on this data; refer to [XML Function Reference](#), below.

The `xml` type can store well-formed “documents”, as defined by the XML standard, as well as “content” fragments, which are defined by the production `XMLDecl? content` in the XML standard. Roughly, this means that content fragments can have more than one top-level element or character node. The expression `xmlvalue IS DOCUMENT` can be used to evaluate whether a particular `xml` value is a full document or only a content fragment.

This section contains the following topics:
- Creating XML Values
- Encoding Handling
- Accessing XML Values
- Processing XML
- Mapping Tables to XML
- Using XML Functions and Expressions

**Creating XML Values**

To produce a value of type `xml` from character data, use the function `xmlparse`:

```
xmlparse ( { DOCUMENT | CONTENT } value)
```

For example:

```
XMLPARSE (DOCUMENT '<?xml version="1.0"?><book><title>Manual</title><chapter>...</chapter></book>')
XMLPARSE (CONTENT 'abc<foo>bar</foo><bar>foo</bar>')
```

The above method converts character strings into XML values according to the SQL standard, but you can also use Greenplum Database syntax like the following:

```
xml '<foo>bar</foo>'
'<foo>bar</foo>':'::xml
```
The `xml` type does not validate input values against a document type declaration (DTD), even when the input value specifies a DTD. There is also currently no built-in support for validating against other XML schema languages such as XML schema.

The inverse operation, producing a character string value from `xml`, uses the function `xmlserialize`:

```sql
xmlserialize ( { DOCUMENT | CONTENT } value AS type )
```

does not validate input values against a document type declaration (DTD), even when the input value specifies a DTD. There is also currently no built-in support for validating against other XML schema languages such as XML schema.

The inverse operation, producing a character string value from `xml`, uses the function `xmlserialize`:

```sql
xmlserialize ( { DOCUMENT | CONTENT } value AS type )
```

type can be character, character varying, or text (or an alias for one of those). Again, according to the SQL standard, this is the only way to convert between type `xml` and character types, but Greenplum Database also allows you to simply cast the value.

When a character string value is cast to or from type `xml` without going through `XMLPARSE` or `XMLSERIALIZE`, respectively, the choice of `DOCUMENT` versus `CONTENT` is determined by the `XML OPTION` session configuration parameter, which can be set using the standard command:

```sql
SET XML OPTION { DOCUMENT | CONTENT };
```
or simply like Greenplum Database:

```sql
SET XML OPTION TO { DOCUMENT | CONTENT };
```

The default is `CONTENT`, so all forms of XML data are allowed.

Note:

With the default XML option setting, you cannot directly cast character strings to type `xml` if they contain a document type declaration, because the definition of XML content fragment does not accept them. If you need to do that, either use `XMLPARSE` or change the XML option.

Encoding Handling

Be careful when dealing with multiple character encodings on the client, server, and in the XML data passed through them. When using the text mode to pass queries to the server and query results to the client (which is the normal mode), Greenplum Database converts all character data passed between the client and the server, and vice versa, to the character encoding of the respective endpoint; see `Character Set Support`.

Ordinarily, this means that encoding declarations contained in XML data can become invalid, as the character data is converted to other encodings while travelling between client and server, because the embedded encoding declaration is not changed. To cope with this behavior, encoding declarations contained in character strings presented for input to the `xml` type are ignored, and content is assumed to be in the current server encoding. Consequently, for correct processing, character strings of XML data must be sent from the client in the current client encoding. It is the responsibility of the client to either convert documents to the current client encoding before sending them to the server, or to adjust the client encoding appropriately. On output, values of type `xml` will not have an encoding declaration, and clients should assume all data is in the current client encoding.

When using binary mode to pass query parameters to the server and query results back to the client, no character set conversion is performed, so the situation is different. In this case, an encoding declaration in the XML data will be observed, and if it is absent, the data will be assumed to be in UTF-8 (as required by the XML standard; note that Greenplum Database does not support UTF-16). On output, data will have an encoding declaration specifying the client encoding, unless the client encoding is UTF-8, in which case it will be omitted.

Note:

Processing XML data with Greenplum Database will be less error-prone and more efficient if the XML data encoding, client encoding, and server encoding are the same. Because XML data is internally processed in UTF-8, computations will be most efficient if the server encoding is also UTF-8.
Accessing XML Values

The xml data type is unusual in that it does not provide any comparison operators. This is because there is no well-defined and universally useful comparison algorithm for XML data. One consequence of this is that you cannot retrieve rows by comparing an xml column against a search value. XML values should therefore typically be accompanied by a separate key field such as an ID. An alternative solution for comparing XML values is to convert them to character strings first, but note that character string comparison has little to do with a useful XML comparison method.

Because there are no comparison operators for the xml data type, it is not possible to create an index directly on a column of this type. If speedy searches in XML data are desired, possible workarounds include casting the expression to a character string type and indexing that, or indexing an XPath expression. Of course, the actual query would have to be adjusted to search by the indexed expression.

Processing XML

To process values of data type xml, Greenplum Database offers the functions xpath and xpath_exists, which evaluate XPath 1.0 expressions.

```
xpath(xpath, xml [, nsarray])
```

The function xpath evaluates the XPath expression xpath (a text value) against the XML value xml. It returns an array of XML values corresponding to the node set produced by the XPath expression.

The second argument must be a well formed XML document. In particular, it must have a single root node element.

The optional third argument of the function is an array of namespace mappings. This array should be a two-dimensional text array with the length of the second axis being equal to 2 (i.e., it should be an array of arrays, each of which consists of exactly 2 elements). The first element of each array entry is the namespace name (alias), the second the namespace URI. It is not required that aliases provided in this array be the same as those being used in the XML document itself (in other words, both in the XML document and in the xpath function context, aliases are local).

Example:

```
```

```
xpath
--------
{test}
(1 row)
```

To deal with default (anonymous) namespaces, do something like this:

```
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])
```
The function `xpath_exists` is a specialized form of the `xpath` function. Instead of returning the individual XML values that satisfy the XPath, this function returns a Boolean indicating whether the query was satisfied or not. This function is equivalent to the standard `XMLExists` predicate, except that it also offers support for a namespace mapping argument.

Example:

```sql
```

```
xpath_exists
--------------
t
(1 row)
```

## Mapping Tables to XML

The following functions map the contents of relational tables to XML values. They can be thought of as XML export functionality:

- `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)`

The return type of each function is `xml`.

`table_to_xml` maps the content of the named table, passed as parameter `tbl`. The `regclass` type accepts strings identifying tables using the usual notation, including optional schema qualifications and double quotes. `query_to_xml` executes the query whose text is passed as parameter `query` and maps the result set. `cursor_to_xml` fetches the indicated number of rows from the cursor specified by the parameter `cursor`. This variant is recommended if large tables have to be mapped, because the result value is built up in memory by each function.

If `tableforest` is false, then the resulting XML document looks like this:

```xml
<tablename>
  <row>
    <columnname1>data</columnname1>
    <columnname2>data</columnname2>
  </row>
  ...
</tablename>
```

If `tableforest` is true, the result is an XML content fragment that looks like this:

```xml
<tablename>
  <columnname1>data</columnname1>
  <columnname2>data</columnname2>
</tablename>

<tablename>
  ...
</tablename>
```
If no table name is available, that is, when mapping a query or a cursor, the string `table` is used in the first format, `row` in the second format.

The choice between these formats is up to the user. The first format is a proper XML document, which will be important in many applications. The second format tends to be more useful in the `cursor_to_xml` function if the result values are to be later reassembled into one document. The functions for producing XML content discussed above, in particular `xmlelement`, can be used to alter the results as desired.

The data values are mapped in the same way as described for the function `xmlelement`, above.

The parameter `nulls` determines whether null values should be included in the output. If true, null values in columns are represented as:

```xml
<columnname xsi:nil="true"/>
```

where `xsi` is the XML namespace prefix for XML schema Instance. An appropriate namespace declaration will be added to the result value. If false, columns containing null values are simply omitted from the output.

The parameter `targetns` specifies the desired XML namespace of the result. If no particular namespace is wanted, an empty string should be passed.

The following functions return XML schema documents describing the mappings performed by the corresponding functions above:

```sql
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)
```

It is essential that the same parameters are passed in order to obtain matching XML data mappings and XML schema documents.

The following functions produce XML data mappings and the corresponding XML schema in one document (or forest), linked together. They can be useful where self-contained and self-describing results are desired:

```sql
table_to_xml_and_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text)
query_to_xml_and_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)
```

In addition, the following functions are available to produce analogous mappings of entire schemas or the entire current database:

```sql
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)
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)
```
Note that these potentially produce large amounts of data, which needs to be built up in memory. When requesting content mappings of large schemas or databases, consider mapping the tables separately instead, possibly even through a cursor.

The result of a schema content mapping looks like this:

```xml
<schemaname>
  table1-mapping
  table2-mapping
  ...
</schemaname>
```

where the format of a table mapping depends on the `tableforest` parameter, as explained above.

The result of a database content mapping looks like this:

```xml
<dbname>
  <schemaname>
    ...
  </schemaname>
  ...
  <schemaname>
    ...
  </schemaname>
  ...
</dbname>
```

where the schema mapping is as above.

The example below demonstrates using the output produced by these functions. The example shows an XSLT stylesheet that converts the output of `table_to_xml_and_xmlschema` to an HTML document containing a tabular rendition of the table data. In a similar manner, the results from these functions can be converted into other XML-based formats.

```xml
<?xml version="1.0"?
  <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=$tabletypename]/xsd:sequence/xsd:element[@name='row']/@type"/>
    <xsl:variable name="rowtypename" select="$schema/xsd:complexType[@name=$tabletypename]/xsd:sequence/xsd:element[@name='row']/@type"/>
    <html>
```
XML Function Reference

The functions described in this section operate on values of type xml. The section *XML Predicates* also contains information about the xml functions and function-like expressions.

**Function:**

xmlcomment

**Synopsis:**

xmlcomment(text)

The function `xmlcomment` creates an XML value containing an XML comment with the specified text as content. The text cannot contain "--" or end with a "-" so that the resulting construct is a valid XML comment. If the argument is null, the result is null.

**Example:**

```
SELECT xmlcomment('hello');
```

```
xmlcomment
------------
<!--hello-->
```

**Function:**

xmlconcat

**Synopsis:**

xmlconcat(xml[, …])

The function `xmlconcat` concatenates a list of individual XML values to create a single value containing an XML content fragment. Null values are omitted; the result is only null if there are no nonnull arguments.
Example:

```sql
SELECT xmlconcat('<abc/>', '<bar>foo</bar>');
```

```
xmllconcat
----------------------
<abc/><bar>foo</bar>
```

XML declarations, if present, are combined as follows:

- If all argument values have the same XML version declaration, that version is used in the result, else no version is used.
- If all argument values have the standalone declaration value "yes", then that value is used in the result.
- If all argument values have a standalone declaration value and at least one is "no", then that is used in the result. Otherwise, the result will have no standalone declaration.
- If the result is determined to require a standalone declaration but no version declaration, a version declaration with version 1.0 will be used because XML requires an XML declaration to contain a version declaration.

Encoding declarations are ignored and removed in all cases.

Example:

```sql
SELECT xmlconcat('<?xml version="1.1"?><foo/>', '<?xml version="1.1" standalone="no"?><bar/>');
```

```
xmllconcat
-----------------------------------
<?xml version="1.1"?><foo/><?xml version="1.1" standalone="no"?><bar/>
```

Function:

`xmlelement`

Synopsis:

```sql
xmlelement(name name [, xmlattributes(value [AS attname] [, ... ])) [, content, ...])
```

The `xmlelement` expression produces an XML element with the given name, attributes, and content.

Examples:

```sql
SELECT xmlelement(name foo);
```

```
xmlelement
------------
<foo/>
```

```sql
SELECT xmlelement(name foo, xmlattributes('xyz' as bar));
```

```
xmlelement
------------------
<foo bar="xyz"/>
```

```sql
SELECT xmlelement(name foo, xmlattributes(current_date as bar), 'cont', 'ent');
```

```
xmlelement
-------------------------------------
<foo bar="2017-01-26">content</foo>
```
Element and attribute names that are not valid XML names are escaped by replacing the offending characters by the sequence _H_H_H_H_, where HHHH is the character's Unicode codepoint in hexadecimal notation. For example:

```
SELECT xmlelement(name "foo$bar", xmlattributes('xyz' as "a&b"));
```

```
xmlelement
----------------------------------
<foo_x0024_bar a_x0026_b="xyz"/>
```

An explicit attribute name need not be specified if the attribute value is a column reference, in which case the column's name will be used as the attribute name by default. In other cases, the attribute must be given an explicit name. So this example is valid:

```
CREATE TABLE test (a xml, b xml);
SELECT xmlelement(name test, xmlattributes(a, b)) FROM test;
```

But these are not:

```
SELECT xmlelement(name test, xmlattributes('constant'), a, b) FROM test;
SELECT xmlelement(name test, xmlattributes(func(a, b))) FROM test;
```

Element content, if specified, will be formatted according to its data type. If the content is itself of type xml, complex XML documents can be constructed. For example:

```
SELECT xmlelement(name foo, xmlattributes('xyz' as bar),
    xmlelement(name abc),
    xmlcomment('test'),
    xmlelement(name xyz));
```

```
xmlelement
----------------------------------------------
<foo bar="xyz"><abc/><!--test--><xyz/></foo>
```

Content of other types will be formatted into valid XML character data. This means in particular that the characters <, >, and & will be converted to entities. Binary data (data type bytea) will be represented in base64 or hex encoding, depending on the setting of the configuration parameter xmlbinary. The particular behavior for individual data types is expected to evolve in order to align the SQL and Greenplum Database data types with the XML schema specification, at which point a more precise description will appear.

**Function:**
xmlforest

**Synopsis:**

```
xmlforest(content [AS name] [, ...])
```

The xmlforest expression produces an XML forest (sequence) of elements using the given names and content.

**Examples:**

```
SELECT xmlforest('abc' AS foo, 123 AS bar);
```

```
xmforest
-------------------------------
<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>

As seen in the second example, the element name can be omitted if the content value is a column reference, in which case the column name is used by default. Otherwise, a name must be specified.

Element names that are not valid XML names are escaped as shown for xml_element above. Similarly, content data is escaped to make valid XML content, unless it is already of type xml.

Note that XML forests are not valid XML documents if they consist of more than one element, so it might be useful to wrap xmlforest expressions in xml_element.

Function:
xmlpi

Synopsis:
xmlpi(name target [, content])

The xmlpi expression creates an XML processing instruction. The content, if present, must not contain the character sequence `?>`.

Example:

SELECT xmlpi(name php, 'echo "hello world";');

xmlpi

-----------------------------------------------------------
<?php echo "hello world";?>

Function:
xmlroot

Synopsis:
xmlroot(xml, version text | no value [, standalone yes|no|no value])

The xmlroot expression alters the properties of the root node of an XML value. If a version is specified, it replaces the value in the root node's version declaration; if a standalone setting is specified, it replaces the value in the root node's standalone declaration.

Example:

SELECT xmlroot(xmlparse(document '<?xml version="1.1"?><content>abc</content>'),
             version '1.0', standalone yes);

xmlroot

----------------------------------------
<?xml version="1.0" standalone="yes"?><content>abc</content>

Function:
xmlagg

xmlagg (xml)
The function `xmlagg` is, unlike the other functions described here, an aggregate function. It concatenates the input values to the aggregate function call, much like `xmlconcat` does, except that concatenation occurs across rows rather than across expressions in a single row. See *Using Functions and Operators* for additional information about aggregate functions.

Example:

```
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/>
```

To determine the order of the concatenation, an `ORDER BY` clause may be added to the aggregate call. For example:

```
SELECT xmlagg(x ORDER BY y DESC) FROM test;
          xmlagg
----------------------
<bar/><foo>abc</foo>
```

The following non-standard approach used to be recommended in previous versions, and may still be useful in specific cases:

```
SELECT xmlagg(x) FROM (SELECT * FROM test ORDER BY y DESC) AS tab;
          xmlagg
----------------------
<bar/><foo>abc</foo>
```

**XML Predicates**

The expressions described in this section check properties of `xml` values.

**Expression:**

**IS DOCUMENT**

**Synopsis:**

```
xml IS DOCUMENT
```

The expression `IS DOCUMENT` returns true if the argument XML value is a proper XML document, false if it is not (that is, it is a content fragment), or null if the argument is null.

**Expression:**

**XMLEXISTS**

**Synopsis:**

```
XMLEXISTS(text PASSING [BY REF] xml [BY REF])
```

The function `xmlexists` returns true if the XPath expression in the first argument returns any nodes, and false otherwise. (If either argument is null, the result is null.)

Example:

```
SELECT xmlexists('//town[text()] = ''Toronto''') PASSING BY REF 'towns'><town>Toronto</town><town>Ottawa</town></towns>');
```
xmlexists
----------
t
(1 row)

The **BY REF** clauses have no effect in Greenplum Database, but are allowed for SQL conformance and compatibility with other implementations. Per SQL standard, the first **BY REF** is required, the second is optional. Also note that the SQL standard specifies the xmlexists construct to take an XQuery expression as first argument, but Greenplum Database currently only supports XPath, which is a subset of XQuery.

**Expression:**

xml_is_well_formed

**Synopsis:**

```sql
xml_is_well_formed(text)
xml_is_well_formed_document(text)
xml_is_well_formed_content(text)
```

These functions check whether a text string is well-formed XML, returning a Boolean result. `xml_is_well_formed_document` checks for a well-formed document, while `xml_is_well_formed_content` checks for well-formed content. `xml_is_well_formed` does the former if the `xmloption` configuration parameter is set to `DOCUMENT`, or the latter if it is set to `CONTENT`. This means that `xml_is_well_formed` is useful for seeing whether a simple cast to type `xml` will succeed, whereas the other two functions are useful for seeing whether the corresponding variants of `XMLPARSE` will succeed.

**Examples:**

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>');</n
  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>');</n
  xml_is_well_formed_document
     --------------------------
     f
     (1 row)
The last example shows that the checks include whether namespaces are correctly matched.

**Query Performance**

Greenplum Database dynamically eliminates irrelevant partitions in a table and optimally allocates memory for different operators in a query. These enhancements scan less data for a query, accelerate query processing, and support more concurrency.

- **Dynamic Partition Elimination**
  
  In Greenplum Database, values available only when a query runs are used to dynamically prune partitions, which improves query processing speed. Enable or disable dynamic partition elimination by setting the server configuration parameter `gp_dynamic_partition_pruning` to ON or OFF; it is ON by default.

- **Memory Optimizations**

  Greenplum Database allocates memory optimally for different operators in a query and frees and re-allocates memory during the stages of processing a query.

  *Note:* Greenplum Database uses GPORCA, the Greenplum next generation query optimizer, by default. GPORCA extends the planning and optimization capabilities of the Greenplum Database legacy optimizer. For information about the features and limitations of GPORCA, see Overview of GPORCA.

**Managing Spill Files Generated by Queries**

Greenplum Database creates spill files, also known as workfiles, on disk if it does not have sufficient memory to execute an SQL query in memory. The default value of 100,000 spill files is sufficient for the majority of queries. However, if a query creates more than the specified number of spill files, Greenplum Database returns this error:

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

Reasons that cause a large number of spill files to be generated include:

- Data skew is present in the queried data.
- The amount memory allocated for the query is too low.

You might be able to run the query successfully by changing the query, changing the data distribution, or changing the system memory configuration. You can use the `gp_workfile_*` views to see spill file usage information. You can control the maximum amount of memory that can used by a query with the Greenplum Database server configuration parameters `max_statement_mem`, `statement_mem`, or through resource queues.

*Monitoring a Greenplum System* contains the following information:

- Information about skew and how to check for data skew
- Information about using the `gp_workfile_*` views

For information about server configuration parameters, see the Greenplum Database Reference Guide. For information about resource queues, see Using Resource Queues.

If you have determined that the query must create more spill files than allowed by the value of server configuration parameter `gp_workfile_limit_files_per_query`, you can increase the value of the parameter.

**Query Profiling**

Examine the query plans of poorly performing queries to identify possible performance tuning opportunities.
Greenplum Database devises a query plan for each query. Choosing the right query plan to match the query and data structure is necessary for good performance. A query plan defines how Greenplum Database will run the query in the parallel execution environment.

The query optimizer uses data statistics maintained by the database to choose a query plan with the lowest possible cost. Cost is measured in disk I/O, shown as units of disk page fetches. The goal is to minimize the total execution cost for the plan.

View the plan for a given query with the EXPLAIN command. EXPLAIN shows the query optimizer's estimated cost for the query plan. For example:

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

EXPLAIN ANALYZE runs the statement in addition to displaying its plan. This is useful for determining how close the optimizer's estimates are to reality. For example:

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

**Note:** In Greenplum Database, the default GPORCA optimizer co-exists with the legacy query optimizer. The EXPLAIN output generated by GPORCA is different than the output generated by the legacy query optimizer.

By default, Greenplum Database uses GPORCA to generate an execution plan for a query when possible.

When the EXPLAIN ANALYZE command uses GPORCA, the EXPLAIN plan shows only the number of partitions that are being eliminated. The scanned partitions are not shown. To show name of the scanned partitions in the segment logs set the server configuration parameter `gp_log_dynamic_partition_pruning` to on. This example SET command enables the parameter.

```sql
SET gp_log_dynamic_partition_pruning = on;
```

For information about GPORCA, see *Querying Data*.

**Reading EXPLAIN Output**

A query plan is a tree of nodes. Each node in the plan represents a single operation, such as a table scan, join, aggregation, or sort.

Read plans from the bottom to the top: each node feeds rows into the node directly above it. The bottom nodes of a plan are usually table scan operations: sequential, index, or bitmap index scans. If the query requires joins, aggregations, sorts, or other operations on the rows, there are additional nodes above the scan nodes to perform these operations. The topmost plan nodes are usually Greenplum Database motion nodes: redistribute, explicit redistribute, broadcast, or gather motions. These operations move rows between segment instances during query processing.

The output of EXPLAIN has one line for each node in the plan tree and shows the basic node type and the following execution cost estimates for that plan node:

- **cost** —Measured in units of disk page fetches. 1.0 equals one sequential disk page read. The first estimate is the start-up cost of getting the first row and the second is the total cost of getting all rows. The total cost assumes all rows will be retrieved, which is not always true; for example, if the query uses LIMIT, not all rows are retrieved.

- **rows** —The total number of rows output by this plan node. This number is usually less than the number of rows processed or scanned by the plan node, reflecting the estimated selectivity of any `WHERE` clause conditions. Ideally, the estimate for the topmost node approximates the number of rows that the query actually returns, updates, or deletes.

- **width** —The total bytes of all the rows that this plan node outputs.
Note the following:

- The cost of a node includes the cost of its child nodes. The topmost plan node has the estimated total execution cost for the plan. This is the number the optimizer intends to minimize.
- The cost reflects only the aspects of plan execution that the query optimizer takes into consideration. For example, the cost does not reflect time spent transmitting result rows to the client.

**EXPLAIN Example**

The following example describes how to read an **EXPLAIN** query plan for a query:

```
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
```

Read the plan from the bottom to the top. To start, the query optimizer sequentially scans the `names` table. Notice the `WHERE` clause is applied as a `filter` condition. This means the scan operation checks the condition for each row it scans and outputs only the rows that satisfy the condition.

The results of the scan operation are passed to a `gather motion` operation. In Greenplum Database, a gather motion is when segments send rows to the master. In this example, we have two segment instances that send to one master instance. This operation is working on `slice1` of the parallel query execution plan. A query plan is divided into `slices` so the segments can work on portions of the query plan in parallel.

The estimated startup cost for this plan is 00.00 (no cost) and a total cost of 20.88 disk page fetches. The optimizer estimates this query will return one row.

**Reading EXPLAIN ANALYZE Output**

**EXPLAIN ANALYZE** plans and runs the statement. The **EXPLAIN ANALYZE** plan shows the actual execution cost along with the optimizer’s estimates. This allows you to see if the optimizer’s estimates are close to reality. **EXPLAIN ANALYZE** also shows the following:

- The total runtime (in milliseconds) in which the query executed.
- The memory used by each slice of the query plan, as well as the memory reserved for the whole query statement.
- The number of `workers` (segments) involved in a plan node operation. Only segments that return rows are counted.
- The maximum number of rows returned by the segment that produced the most rows for the operation. If multiple segments produce an equal number of rows, **EXPLAIN ANALYZE** shows the segment with the longest `<time> to end`.
- The segment id of the segment that produced the most rows for an operation.
- For relevant operations, the amount of memory (`work_mem`) used by the operation. If the `work_mem` was insufficient to perform the operation in memory, the plan shows the amount of data spilled to disk for the lowest-performing segment. For example:

  Work_mem used: 64K bytes avg, 64K bytes max (seg0).
  Work_mem wanted: 90K bytes avg, 90K byes max (seg0) to lessen
  workfile I/O affecting 2 workers.

- The time (in milliseconds) in which the segment that produced the most rows retrieved the first row, and the time taken for that segment to retrieve all rows. The result may omit `<time> to first row` if it is the same as the `<time> to end`.
**EXPLAIN ANALYZE Examples**

This example describes how to read an EXPLAIN ANALYZE query plan using the same query. The bold parts of the plan show actual timing and rows returned for each plan node, as well as memory and time statistics for the whole query.

```
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
```

Read the plan from the bottom to the top. The total elapsed time to run this query was 22.548 milliseconds.

The sequential scan operation had only one segment (seg0) that returned rows, and it returned just 1 row. It took 0.255 milliseconds to find the first row and 0.486 to scan all rows. This result is close to the optimizer's estimate: the query optimizer estimated it would return one row for this query. The gather motion (segments sending data to the master) received 1 row. The total elapsed time for this operation was 0.537 milliseconds.

**Determining the Query Optimizer**

You can view EXPLAIN output to determine if GPORCA is enabled for the query plan and whether GPORCA or the legacy query optimizer generated the explain plan. The information appears at the end of the EXPLAIN output. The Settings line displays the setting of the server configuration parameter OPTIMIZER. The Optimizer status line displays whether GPORCA or the legacy query optimizer generated the explain plan.

For these two example query plans, GPORCA is enabled, the server configuration parameter OPTIMIZER is on. For the first plan, GPORCA generated the EXPLAIN plan. For the second plan, Greenplum Database fell back to the legacy query optimizer to generate the 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)
      -> Table Scan on part (cost=0.00..97.69 rows=100040
Settings: optimizer=on
Optimizer status: PGO version 1.584
(5 rows)
```

```
explain select count(*) from part;
```
For this query, the server configuration parameter `OPTIMIZER` is off.

```sql
explain select count(*) from part;
```

Examining Query Plans to Solve Problems

If a query performs poorly, examine its query plan and ask the following questions:

- **Do operations in the plan take an exceptionally long time?** Look for an operation consumes the majority of query processing time. For example, if an index scan takes longer than expected, the index could be out-of-date and need to be reindexed. Or, adjust `enable_<operator>` parameters to see if you can force the legacy query optimizer (planner) to choose a different plan by disabling a particular query plan operator for that query.

- **Are the optimizer’s estimates close to reality?** Run `EXPLAIN ANALYZE` and see if the number of rows the optimizer estimates is close to the number of rows the query operation actually returns. If there is a large discrepancy, collect more statistics on the relevant columns.

  See the *Greenplum Database Reference Guide* for more information on the `EXPLAIN` and `ANALYZE` commands.

- **Are selective predicates applied early in the plan?** Apply the most selective filters early in the plan so fewer rows move up the plan tree. If the query plan does not correctly estimate query predicate selectivity, collect more statistics on the relevant columns. See the `ANALYZE` command in the *Greenplum Database Reference Guide* for more information collecting statistics. You can also try reordering the `WHERE` clause of your SQL statement.

- **Does the optimizer choose the best join order?** When you have a query that joins multiple tables, make sure that the optimizer chooses the most selective join order. Joins that eliminate the largest number of rows should be done earlier in the plan so fewer rows move up the plan tree.

  If the plan is not choosing the optimal join order, set `joinCollapseLimit=1` and use explicit `JOIN` syntax in your SQL statement to force the legacy query optimizer (planner) to the specified join order. You can also collect more statistics on the relevant join columns.

  See the `ANALYZE` command in the *Greenplum Database Reference Guide* for more information collecting statistics.
• **Does the optimizer selectively scan partitioned tables?** If you use table partitioning, is the optimizer selectively scanning only the child tables required to satisfy the query predicates? Scans of the parent tables should return 0 rows since the parent tables do not contain any data. See [Verifying Your Partition Strategy](#) for an example of a query plan that shows a selective partition scan.

• **Does the optimizer choose hash aggregate and hash join operations where applicable?** Hash operations are typically much faster than other types of joins or aggregations. Row comparison and sorting is done in memory rather than reading/writing from disk. To enable the query optimizer to choose hash operations, there must be sufficient memory available to hold the estimated number of rows. Try increasing work memory to improve performance for a query. If possible, run an `EXPLAIN ANALYZE` for the query to show which plan operations spilled to disk, how much work memory they used, and how much memory was required to avoid spilling to disk. For example:

   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.

   The "bytes wanted" message from `EXPLAIN ANALYZE` is based on the amount of data written to work files and is not exact. The minimum `work_mem` needed can differ from the suggested value.
Working with External Data

External tables provide access to data stored in data sources outside of Greenplum Database as if the data were stored in regular database tables. Data can be read from or written to external tables.

An external table is a Greenplum database table backed with data that resides outside of the database. An external table is either readable or writable. It can be used like a regular database table in SQL commands such as `SELECT` and `INSERT` and joined with other tables. External tables are most often used to load and unload database data.

Web-based external tables provide access to data served by an HTTP server or an operating system process. See Creating and Using External Web Tables for more about web-based tables.

Defining External Tables

External tables enable accessing external data as if it were a regular database table. They are often used to move data into and out of a Greenplum database.

To create an external table definition, you specify the format of your input files and the location of your external data sources. For information input file formats, see Formatting Data Files.

Use one of the following protocols to access external table data sources. You cannot mix protocols in `CREATE EXTERNAL TABLE` statements:

- `file://` accesses external data files on segment host that the Greenplum Database superuser (gpadmin) can access. See `file:// Protocol`.
- `gpfdist://` points to a directory on the file host and serves external data files to all Greenplum Database segments in parallel. See `gpfdist:// Protocol`.
- `gpfdists://` is the secure version of `gpfdist`. See `gpfdists:// Protocol`.
  - The files can be stored on an Amazon EMR instance HDFS. See Using Amazon EMR with Greenplum Database installed on AWS.
- `s3://` accesses files in an Amazon S3 bucket. See `s3:// Protocol`.

External tables access external files from within the database as if they are regular database tables. External tables defined with the `gpfdist/gpfdists, gphdfs, and s3` protocols utilize Greenplum parallelism by using the resources of all Greenplum Database segments to load or unload data. The `gphdfs` protocol leverages the parallel architecture of the Hadoop Distributed File System to access files on that system. The `s3` protocol utilizes the Amazon Web Services (AWS) capabilities.

You can query external table data directly and in parallel using SQL commands such as `SELECT, JOIN, or SORT EXTERNAL TABLE DATA`, and you can create views for external tables.

The steps for using external tables are:

1. Define the external table.
   - To use the `s3` protocol, you must also configure Greenplum Database and enable the protocol. See `s3:// Protocol`.
2. Do one of the following:
   - Start the Greenplum Database file server(s) when using the `gpfdist` or `gpfdists` protocols.
   - Verify that you have already set up the required one-time configuration for the `gphdfs` protocol.
   - Verify the Greenplum Database configuration for the `s3` protocol.
3. Place the data files in the correct locations.
4. Query the external table with SQL commands.
Greenplum Database provides readable and writable external tables:

- Readable external tables for data loading. Readable external tables support basic extraction, transformation, and loading (ETL) tasks common in data warehousing. Greenplum Database segment instances read external table data in parallel to optimize large load operations. You cannot modify readable external tables.

- Writable external tables for data unloading. Writable external tables support:
  - Selecting data from database tables to insert into the writable external table.
  - Sending data to an application as a stream of data. For example, unload data from Greenplum Database and send it to an application that connects to another database or ETL tool to load the data elsewhere.
  - Receiving output from Greenplum parallel MapReduce calculations.

Writable external tables allow only `INSERT` operations.

External tables can be file-based or web-based. External tables using the `file://` protocol are read-only tables.

- Regular (file-based) external tables access static flat files. Regular external tables are rescannable: the data is static while the query runs.
- Web (web-based) external tables access dynamic data sources, either on a web server with the `http://` protocol or by executing OS commands or scripts. External web tables are not rescannable: the data can change while the query runs.

Dump and restore operate only on external and external web table definitions, not on the data sources.

**file:// Protocol**

The `file://` protocol is used in a URI that specifies the location of an operating system file.

The URI includes the host name, port, and path to the file. Each file must reside on a segment host in a location accessible by the Greenplum superuser (`gpadmin`). The host name used in the URI must match a segment host name registered in the `gp_segment_configuration` system catalog table.

The `LOCATION` clause can have multiple URIs, as shown in this example:

```sql
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);
```

The number of URIs you specify in the `LOCATION` clause is the number of segment instances that will work in parallel to access the external table. For each URI, Greenplum assigns a primary segment on the specified host to the file. For maximum parallelism when loading data, divide the data into as many equally sized files as you have primary segments. This ensures that all segments participate in the load. The number of external files per segment host cannot exceed the number of primary segment instances on that host. For example, if your array has four primary segment instances per segment host, you can place four external files on each segment host. Tables based on the `file://` protocol can only be readable tables.

The system view `pg_max_external_files` shows how many external table files are permitted per external table. This view lists the available file slots per segment host when using the `file://` protocol. The view is only applicable for the `file://` protocol. For example:

```sql
SELECT * FROM pg_max_external_files;
```
**gpfdist:// Protocol**

The gpfdist:// protocol is used in a URI to reference a running gpfdist instance.

The gpfdist utility serves external data files from a directory on a file host to all Greenplum Database segments in parallel.

gpfdist is located in the $GPHOME/bin directory on your Greenplum Database master host and on each segment host.

Run gpfdist on the host where the external data files reside. gpfdist uncompresses gzip (.gz) and bzip2 (.bz2) files automatically. You can use the wildcard character (*) or other C-style pattern matching to denote multiple files to read. The files specified are assumed to be relative to the directory that you specified when you started the gpfdist instance.

All primary segments access the external file(s) in parallel, subject to the number of segments set in the gp_external_max_segments server configuration parameter. Use multiple gpfdist data sources in a CREATE EXTERNAL TABLE statement to scale the external table's scan performance. For more information about configuring gpfdist, see *Using the Greenplum Parallel File Server (gpfdist)*.

See the gpfdist reference documentation for more information about using gpfdist with external tables.

---

**gpfdists:// Protocol**

The gpfdists:// protocol is a secure version of the gpfdist:// protocol.

To use it, you run the gpfdist utility with the --ssl option. When specified in a URI, the gpfdists:// protocol enables encrypted communication and secure identification of the file server and the Greenplum Database to protect against attacks such as eavesdropping and man-in-the-middle attacks.

gpfdists implements SSL security in a client/server scheme with the following attributes and limitations:

- Client certificates are required.
- Multilingual certificates are not supported.
- A Certificate Revocation List (CRL) is not supported.
- The TLSv1 protocol is used with the TLS_RSA_WITH_AES_128_CBC_SHA encryption algorithm.
- SSL parameters cannot be changed.
- SSL renegotiation is supported.
- The SSL ignore host mismatch parameter is set to false.
- Private keys containing a passphrase are not supported for the gpfdist file server (server.key) and for the Greenplum Database (client.key).
- Issuing certificates that are appropriate for the operating system in use is the user's responsibility. Generally, converting certificates as shown in [https://www.sslshopper.com/ssl-converter.html](https://www.sslshopper.com/ssl-converter.html) is supported.

**Note:** A server started with the gpfdist --ssl option can only communicate with the gpfdists protocol. A server that was started with gpfdist without the --ssl option can only communicate with the gpfdist protocol.

- The client certificate file, client.crt
- The client private key file, client.key

Use one of the following methods to invoke the gpfdists protocol.

- Run gpfdist with the --ssl option and then use the gpfdists protocol in the LOCATION clause of a CREATE EXTERNAL TABLE statement.
- Use a gpload YAML control file with the SSL option set to true. Running gpload starts the gpfdist server with the --ssl option, then uses the gpfdists protocol.

Using gpfdists requires that the following client certificates reside in the $PGDATA/gpfdists directory on each segment.
- The client certificate file, `client.crt`
- The client private key file, `client.key`
- The trusted certificate authorities, `root.crt`

For an example of loading data into an external table security, see Example 3—Multiple gpfdists instances.

**gphdfs:// Protocol**

The `gphdfs://` protocol specifies an external file path on a Hadoop Distributed File System (HDFS).

The protocol allows specifying external files in Hadoop clusters configured with or without Hadoop HA (high availability) and in MapR clusters. File names may contain wildcard characters and the files can be in CSV, TEXT, or custom formats.

When Greenplum links with HDFS files, all the data is read in parallel from the HDFS data nodes into the Greenplum segments for rapid processing. Greenplum determines the connections between the segments and nodes.

Each Greenplum segment reads one set of Hadoop data blocks. For writing, each Greenplum segment writes only the data it contains. The following figure illustrates an external table located on a HDFS file system.

![Figure 29: External Table Located on a Hadoop Distributed File System](image)

The FORMAT clause describes the format of the external table files. Valid file formats are similar to the formatting options available with the PostgreSQL COPY command and user-defined formats for the gphdfs protocol. If the data in the file does not use the default column delimiter, escape character, null string and so on, you must specify the additional formatting options so that Greenplum Database reads the data in the external file correctly. The gphdfs protocol requires a one-time setup. See One-time HDFS Protocol Installation.
**pxf:// Protocol**

You can use the PXF Extension Framework `pxf://` protocol to access data on external HDFS and Hive systems.

The PXF Extension Framework `pxf` protocol is packaged as a Greenplum Database extension. The `pxf` protocol supports reading HDFS file and Hive table data. The protocol does not yet support writing to HDFS or Hive data stores.

When you use the `pxf` protocol to query HDFS and Hive systems, you specify the HDFS file or Hive table you want to access. PXF requests the data from HDFS and delivers the relevant portions in parallel to each Greenplum Database segment instance serving the query.

You must explicitly initialize and start the PXF Extension Framework before you can use the `pxf` protocol to read external data. You must also grant permissions to the `pxf` protocol and enable PXF in each database in which you want to create external tables to access external data.

For detailed information about configuring and using the PXF Extension Framework and the `pxf` protocol, refer to *Accessing HDFS and Hive Data with PXF*.

**s3:// Protocol**

The `s3` protocol is used in a URL that specifies the location of an Amazon S3 bucket and a prefix to use for reading or writing files in the bucket.

Amazon Simple Storage Service (Amazon S3) provides secure, durable, highly-scalable object storage. For information about Amazon S3, see *Amazon S3*.

You can define read-only external tables that use existing data files in the S3 bucket for table data, or writable external tables that store the data from INSERT operations to files in the S3 bucket. Greenplum Database uses the S3 URL and prefix specified in the protocol URL either to select one or more files for a read-only table, or to define the location and filename format to use when uploading S3 files for INSERT operations to writable tables.

The `s3` protocol also supports Dell EMC Elastic Cloud Storage (ECS), an Amazon S3 compatible service.

This topic contains the sections:

- Configuring and Using S3 External Tables
- About the S3 Protocol URL
- About S3 Data Files
- `s3` Protocol AWS Server-Side Encryption Support
- `s3` Protocol Proxy Support
- About the `s3` Protocol config Parameter
- `s3` Protocol Configuration File
- `s3` Protocol Limitations
- Using the `gpcheckcloud` Utility

### Configuring and Using S3 External Tables

Follow these basic steps to configure the S3 protocol and use S3 external tables, using the available links for more information. See also *s3 Protocol Limitations* to better understand the capabilities and limitations of S3 external tables:

1. Configure each database to support the `s3` protocol:
   a. In each database that will access an S3 bucket with the `s3` protocol, create the read and write functions for the `s3` protocol library:

   ```sql
   CREATE OR REPLACE FUNCTION write_to_s3() RETURNS integer AS
   ```
CREATE OR REPLACE FUNCTION read_from_s3() RETURNS integer AS '$libdir/gps3ext.so', 's3_export' LANGUAGE C STABLE;

CREATE OR REPLACE FUNCTION write_to_s3() RETURNS void AS '$libdir/gps3ext.so', 's3_import' LANGUAGE C STABLE;

b. In each database that will access an S3 bucket, declare the s3 protocol and specify the read and write functions you created in the previous step:

CREATE PROTOCOL s3 (writefunc = write_to_s3, readfunc = read_from_s3);

**Note:** The protocol name s3 must be the same as the protocol of the URL specified for the external table you create to access an S3 resource.

The corresponding function is called by every Greenplum Database segment instance. All segment hosts must have access to the S3 bucket.

2. On each Greenplum Database segment, create and install the s3 protocol configuration file:

   a. Create a template s3 protocol configuration file using the gpcheckcloud utility:

   ```bash
   gpcheckcloud -t > ./mytest_s3.config
   ```

   b. Edit the template file to specify the accessid and secret required to connect to the S3 location. See s3 Protocol Configuration File for information about other s3 protocol configuration parameters.

   c. Copy the file to the same location and filename for all Greenplum Database segments on all hosts. The default file location is `gpseg_data_dir/gpseg_prefixN/s3/s3.conf`. `gpseg_data_dir` is the path to the Greenplum Database segment data directory, `gpseg_prefix` is the segment prefix, and `N` is the segment ID. The segment data directory, prefix, and ID are set when you initialize a Greenplum Database system.

   If you copy the file to a different location or filename, then you must specify the location with the config parameter in the s3 protocol URL. See About the s3 Protocol config Parameter.

   d. Use the gpcheckcloud utility to validate connectivity to the S3 bucket:

   ```bash
   gpcheckcloud -c "s3://<s3-endpoint>/<s3-bucket> config=./mytest_s3.config"
   ```

   Specify the correct path to the configuration file for your system, as well as the S3 endpoint name and bucket that you want to check. gpcheckcloud attempts to connect to the S3 endpoint and lists any files in the S3 bucket, if available. A successful connection ends with the message:

   ```
   Your configuration works well.
   ```

   You can optionally use gpcheckcloud to validate uploading to and downloading from the S3 bucket, as described in Using the gpcheckcloud Utility.

3. After completing the previous steps to create and configure the s3 protocol, you can specify an s3 protocol URL in the CREATE EXTERNAL TABLE command to define S3 external tables. For read-only S3 tables, the URL defines the location and prefix used to select existing data files that comprise the S3 table. For example:

```
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';
```
For writable S3 tables, the protocol URL defines the S3 location in which Greenplum database stores data files for the table, as well as a prefix to use when creating files for table INSERT operations. For example:

```
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';
```

See About the S3 Protocol URL for more information.

**About the S3 Protocol URL**

For the s3 protocol, you specify a location for files and an optional configuration file location in the LOCATION clause of the CREATE EXTERNAL TABLE command. This is the syntax:

```
's3://S3_endpoint[:port]/bucket_name/[S3_prefix] [region=S3_region]
[config=config_file_location]'
```

The s3 protocol requires that you specify the S3 endpoint and S3 bucket name. Each Greenplum Database segment instance must have access to the S3 location. The optional S3_prefix value is used to select files for read-only S3 tables, or as a filename prefix to use when uploading files for S3 writable tables.

**Note:** The Greenplum Database s3 protocol URL must include the S3 endpoint hostname.

To specify an ECS endpoint (an Amazon S3 compatible service) in the LOCATION clause, you must set the s3 configuration file parameter version to 2. The version parameter controls whether the region parameter is used in the LOCATION clause. You can also specify an Amazon S3 location when the version parameter is 2. For information about version parameter, see s3 Protocol Configuration File.

**Note:** Although the S3_prefix is an optional part of the syntax, you should always include an S3 prefix for both writable and read-only S3 tables to separate datasets as part of the CREATE EXTERNAL TABLE syntax.

For writable S3 tables, the s3 protocol URL specifies the endpoint and bucket name where Greenplum Database uploads data files for the table. The S3 bucket permissions must be Upload/Delete for the S3 user ID that uploads the files. The S3 file prefix is used for each new file uploaded to the S3 location as a result of inserting data to the table. See About S3 Data Files.

For read-only S3 tables, the S3 file prefix is optional. If you specify an S3_prefix, then the s3 protocol selects all files that start with the specified prefix as data files for the external table. The s3 protocol does not use the slash character (/) as a delimiter, so a slash character following a prefix is treated as part of the prefix itself.

For example, consider the following 5 files that each have the S3_endpoint named s3-us-west-2.amazonaws.com and the 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/abcdfff
```

- If the S3 URL is provided as `s3://s3-us-west-2.amazonaws.com/test1/abc`, then the abc prefix selects all 5 files.
• If the S3 URL is provided as `s3://s3-us-west-2.amazonaws.com/test1/abc/`, then the `abc/` prefix selects the files `s3://s3-us-west-2.amazonaws.com/test1/abc/` and `s3://s3-us-west-2.amazonaws.com/test1/abc/xx`.

• If the S3 URL is provided as `s3://s3-us-west-2.amazonaws.com/test1/abcd`, then the `abcd` prefix selects the files `s3://s3-us-west-2.amazonaws.com/test1/abcdef` and `s3://s3-us-west-2.amazonaws.com/test1/abcdefff`.

Wildcard characters are not supported in an S3 prefix; however, the S3 prefix functions as if a wildcard character immediately followed the prefix itself.

All of the files selected by the S3 URL (`S3_endpoint/bucket_name/S3_prefix`) are used as the source for the external table, so they must have the same format. Each file must also contain complete data rows. A data row cannot be split between files. The S3 file permissions must be Open/Download and View for the S3 user ID that is accessing the files.

For information about the Amazon S3 endpoints see [http://docs.aws.amazon.com/general/latest/gr/rande.html#s3_region](http://docs.aws.amazon.com/general/latest/gr/rande.html#s3_region). For information about S3 buckets and folders, see the Amazon S3 documentation [https://aws.amazon.com/documentation/s3/](https://aws.amazon.com/documentation/s3/). For information about the S3 file prefix, see the Amazon S3 documentation [Listing Keys Hierarchically Using a Prefix and Delimiter](https://aws.amazon.com/documentation/s3/).

The `config` parameter specifies the location of the required `s3` protocol configuration file that contains AWS connection credentials and communication parameters. See About the `s3` Protocol config Parameter.

### About S3 Data Files

For each `INSERT` operation to a writable S3 table, each Greenplum Database segment uploads a single file to the configured S3 bucket using the filename format `<prefix><segment_id><random><extension> [.gz]` where:

- `<prefix>` is the prefix specified in the S3 URL.
- `<segment_id>` is the Greenplum Database segment ID.
- `<random>` is a random number that is used to ensure that the filename is unique.
- `<extension>` describes the file type (.txt or .csv, depending on the value you provide in the `FORMAT` clause of `CREATE WRITABLE EXTERNAL TABLE`). Files created by the `gpcheckcloud` utility always uses the extension `.data`.
- `.gz` is appended to the filename if compression is enabled for S3 writable tables (the default).

For writable S3 tables, you can configure the buffer size and the number of threads that segments use for uploading files. See [s3 Protocol Configuration File](https://aws.amazon.com/documentation/s3/).

For read-only S3 tables, all of the files specified by the S3 file location (`S3_endpoint/bucket_name/S3_prefix`) are used as the source for the external table and must have the same format. Each file must also contain complete data rows. If the files contain an optional header row, the column names in the header row cannot contain a newline character (`\n`) or a carriage return (`\r`). Also, the column delimiter cannot be a newline character (`\n`) or a carriage return character (`\r`).

The `s3` protocol recognizes the gzip format and uncompress the files. Only the gzip compression format is supported.

The S3 file permissions must be Open/Download and View for the S3 user ID that is accessing the files. Writable S3 tables require the S3 user ID to have Upload/Delete permissions.

For read-only S3 tables, each segment can download one file at a time from S3 location using several threads. To take advantage of the parallel processing performed by the Greenplum Database segments, the files in the S3 location should be similar in size and the number of files should allow for multiple segments to download the data from the S3 location. For example, if the Greenplum Database system consists of 16 segments and there was sufficient network bandwidth, creating 16 files in the S3 location allows each segment to download a file from the S3 location. In contrast, if the location contained only 1 or 2 files, only 1 or 2 segments download data.
**s3 Protocol AWS Server-Side Encryption Support**

Greenplum Database supports server-side encryption using Amazon S3-managed keys (SSE-S3) for AWS S3 files you access with readable and writable external tables created using the s3 protocol. SSE-S3 encrypts your object data as it writes to disk, and transparently decrypts the data for you when you access it.

**Note:** The s3 protocol supports SSE-S3 only for Amazon Web Services S3 files. SS3-SE is not supported when accessing files in S3 compatible services.

Your S3 accessid and secret permissions govern your access to all S3 bucket objects, whether the data is encrypted or not. However, you must configure your client to use S3-managed keys for accessing encrypted data.

Refer to Protecting Data Using Server-Side Encryption in the AWS documentation for additional information about AWS Server-Side Encryption.

**Configuring S3 Server-Side Encryption**

s3 protocol server-side encryption is disabled by default. To take advantage of server-side encryption on AWS S3 objects you write using the Greenplum Database s3 protocol, you must set the server_side_encryption configuration parameter in your s3 configuration file to the value sse-s3:

```
server_side_encryption = sse-s3
```

When the configuration file you provide to a CREATE WRITABLE EXTERNAL TABLE call using the s3 protocol includes the server_side_encryption = sse-s3 setting, Greenplum Database applies encryption headers for you on all INSERT operations on that external table. S3 then encrypts on write the object(s) identified by the URI you provided in the LOCATION clause.

S3 transparently decrypts data during read operations of encrypted files accessed via readable external tables you create using the s3 protocol. No additional configuration is required.

For further encryption configuration granularity, you may consider creating Amazon Web Services S3 Bucket Policy(s), identifying the objects you want to encrypt and the write actions on those objects as described in the Protecting Data Using Server-Side Encryption with Amazon S3-Managed Encryption Keys (SSE-S3) AWS documentation.

**s3 Protocol Proxy Support**

You can specify a URL that is the proxy that S3 uses to connect to a data source. S3 supports these protocols: HTTP, HTTPS, and SOCKS (4, 4a, 5, 5h). You can specify a proxy with the s3 protocol configuration parameter proxy or an environment variable. If the configuration parameter is set, the environment variables are ignored.

To specify proxy with an environment variable, you set the environment variable based on the protocol: http_proxy, https_proxy, or socks_proxy. You can specify a different URL for each protocol by setting the appropriate environment variable. S3 supports these environment variables.

- all_proxy specifies the proxy URL that is used if an environment variable for a specific protocol is not set.
- no_proxy specifies a comma-separated list of hosts names that do not use the proxy specified by an environment variable.

The environment variables must be set must and must be accessible to Greenplum Database on all Greenplum Database hosts.

For information about the configuration parameter proxy, see s3 Protocol Configuration File.
About the s3 Protocol config Parameter

The optional config parameter specifies the location of the required s3 protocol configuration file. The file contains Amazon Web Services (AWS) connection credentials and communication parameters. For information about the file, see s3 Protocol Configuration File.

The configuration file is required on all Greenplum Database segment hosts. This is default location is a location in the data directory of each Greenplum Database segment instance.

```
gpseg_data_dir/gpseg_prefixN/s3/s3.conf
```

The gpseg_data_dir is the path to the Greenplum Database segment data directory, the gpseg_prefix is the segment prefix, and N is the segment ID. The segment data directory, prefix, and ID are set when you initialize a Greenplum Database system.

If you have multiple segment instances on segment hosts, you can simplify the configuration by creating a single location on each segment host. Then you specify the absolute path to the location with the config parameter in the s3 protocol LOCATION clause. This example specifies a location in the gpadmin home directory.

```
LOCATION ('s3://s3-us-west-2.amazonaws.com/test/my_data config=/home/gpadmin/s3.conf')
```

All segment instances on the hosts use the file /home/gpadmin/s3.conf.

s3 Protocol Configuration File

When using the s3 protocol, an s3 protocol configuration file is required on all Greenplum Database segments. The default location is:

```
gpseg_data_dir/gpseg-prefixN/s3/s3.conf
```

The gpseg_data_dir is the path to the Greenplum Database segment data directory, the gpseg-prefix is the segment prefix, and N is the segment ID. The segment data directory, prefix, and ID are set when you initialize a Greenplum Database system.

If you have multiple segment instances on segment hosts, you can simplify the configuration by creating a single location on each segment host. Then you can specify the absolute path to the location with the config parameter in the s3 protocol LOCATION clause. However, note that both read-only and writable S3 external tables use the same parameter values for their connections. If you want to configure protocol parameters differently for read-only and writable S3 tables, then you must use two different s3 protocol configuration files and specify the correct file in the CREATE EXTERNAL TABLE statement when you create each table.

This example specifies a single file location in the s3 directory of the gpadmin home directory:

```
config=/home/gpadmin/s3/s3.conf
```

All segment instances on the hosts use the file /home/gpadmin/s3/s3.conf.

The s3 protocol configuration file is a text file that consists of a [default] section and parameters This is an example configuration file:

```
[default]
secret = "secret"
accessid = "user access id"
threadnum = 3
chunksize = 67108864
```
You can use the Greenplum Database gpcheckcloud utility to test the S3 configuration file. See Using the gpcheckcloud Utility.

**s3 Configuration File Parameters**

**accessid**
Required. AWS S3 ID to access the S3 bucket.

**secret**
Required. AWS S3 passcode for the S3 ID to access the S3 bucket.

**autocompress**
For writable S3 external tables, this parameter specifies whether to compress files (using gzip) before uploading to S3. Files are compressed by default if you do not specify this parameter.

**chunkszie**
The buffer size that each segment thread uses for reading from or writing to the S3 server. The default is 64 MB. The minimum is 8MB and the maximum is 128MB.

When inserting data to a writable S3 table, each Greenplum Database segment writes the data into its buffer (using multiple threads up to the threadnum value) until it is full, after which it writes the buffer to a file in the S3 bucket. This process is then repeated as necessary on each segment until the insert operation completes.

Because Amazon S3 allows a maximum of 10,000 parts for multipart uploads, the minimum chunksize value of 8MB supports a maximum insert size of 80GB per Greenplum database segment. The maximum chunksize value of 128MB supports a maximum insert size 1.28TB per segment. For writable S3 tables, you must ensure that the chunksize setting can support the anticipated table size of your table. See Multipart Upload Overview in the S3 documentation for more information about uploads to S3.

**encryption**
Use connections that are secured with Secure Sockets Layer (SSL). Default value is true. The values true, t, on, yes, and y (case insensitive) are treated as true. Any other value is treated as false.

If the port is not specified in the URL in the LOCATION clause of the CREATE EXTERNAL TABLE command, the configuration file encryption parameter affects the port used by the s3 protocol (port 80 for HTTP or port 443 for HTTPS). If the port is specified, that port is used regardless of the encryption setting.

**low_speed_limit**
The upload/download speed lower limit, in bytes per second. The default speed is 10240 (10K). If the upload or download speed is slower than the limit for longer than the time specified by low_speed_time, then the connection is aborted and retried. After 3 retries, the s3 protocol returns an error. A value of 0 specifies no lower limit.

**low_speed_time**
When the connection speed is less than low_speed_limit, this parameter specified the amount of time, in seconds, to wait before aborting an upload to or a download from the S3 bucket. The default is 60 seconds. A value of 0 specifies no time limit.

**proxy**
Specify a URL that is the proxy that S3 uses to connect to a data source. S3 supports these protocols: HTTP, HTTPS, and SOCKS (4, 4a, 5, 5h). This is the format for the parameter.

```
proxy = protocol://[user:password@]proxyhost[:port]
```
If this parameter is not set or is an empty string (proxy = ""), S3 uses the proxy specified by the environment variable http_proxy, https_proxy, or socks_proxy (and the environment variables all_proxy and no_proxy). The environment variable that S3 uses depends on the protocol. For information about the environment variables, see s3 Protocol Proxy Support in the Greenplum Database Administrator Guide.

There can be at most one proxy parameter in the configuration file. The URL specified by the parameter is the proxy for all supported protocols.

**server_side_encryption**

The S3 server-side encryption method that has been configured for the bucket. Greenplum Database supports only server-side encryption with Amazon S3-managed keys, identified by the configuration parameter value sse-s3. Server-side encryption is disabled (none) by default.

**threadnum**

The maximum number of concurrent threads a segment can create when uploading data to or downloading data from the S3 bucket. The default is 4. The minimum is 1 and the maximum is 8.

**verfycert**

Controls how the s3 protocol handles authentication when establishing encrypted communication between a client and an S3 data source over HTTPS. The value is either true or false. The default value is true.

- verfycert=false - Ignores authentication errors and allows encrypted communication over HTTPS.
- verfycert=true - Requires valid authentication (a proper certificate) for encrypted communication over HTTPS.

Setting the value to false can be useful in testing and development environments to allow communication without changing certificates.

**Warning:** Setting the value to false exposes a security risk by ignoring invalid credentials when establishing communication between a client and a S3 data store.

**version**

Specifies the version of the information specified in the LOCATION clause of the CREATE EXTERNAL TABLE command. The value is either 1 or 2. The default value is 1.

If the value is 1, the LOCATION clause supports an Amazon S3 URL, and does not contain the region parameter. If the value is 2, the LOCATION clause supports S3 compatible services and must include the region parameter. The region parameter specifies the S3 data source region. For this S3 URL s3://s3-us-west-2.amazonaws.com/s3test.example.com/dataset1/normal/, the AWS S3 region is us-west-2.

If version is 1 or is not specified, this is an example of the LOCATION clause of the CREATE EXTERNAL TABLE command that specifies an Amazon S3 endpoint.

LOCATION ('s3://s3-us-west-2.amazonaws.com/s3test.example.com/dataset1/normal/ config=/home/gpadmin/aws_s3/s3.conf')

If version is 2, this is an example LOCATION clause with the region parameter for an AWS S3 compatible service.

LOCATION ('s3://test.company.com/s3test.company/test1/normal/ region=local-test config=/home/gpadmin/aws_s3/s3.conf')
If version is 2, the LOCATION clause can also specify an Amazon S3 endpoint. This example specifies an Amazon S3 endpoint that uses the region parameter.

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:** Greenplum Database can require up to threadnum * chunksize memory on each segment host when uploading or downloading S3 files. Consider this s3 protocol memory requirement when you configure overall Greenplum Database memory, and increase the value of gp_vmem_protect_limit as necessary.

### s3 Protocol Limitations

These are s3 protocol limitations:

- Only the S3 path-style URL is supported.

  s3://S3_endpoint/bucketname/[S3_prefix]

- Only the S3 endpoint is supported. The protocol does not support virtual hosting of S3 buckets (binding a domain name to an S3 bucket).

- AWS signature version 4 signing process is supported. For information about the S3 endpoints supported by each signing process, see [http://docs.aws.amazon.com/general/latest/gr/rande.html#s3_region](http://docs.aws.amazon.com/general/latest/gr/rande.html#s3_region).

- Only a single URL and optional configuration file is supported in the LOCATION clause of the CREATE EXTERNAL TABLE command.

- If the NEWLINE parameter is not specified in the CREATE EXTERNAL TABLE command, the newline character must be identical in all data files for specific prefix. If the newline character is different in some data files with the same prefix, read operations on the files might fail.

- For writable S3 external tables, only the INSERT operation is supported. UPDATE, DELETE, and TRUNCATE operations are not supported.

- Because Amazon S3 allows a maximum of 10,000 parts for multipart uploads, the maximum chunksize value of 128MB supports a maximum insert size of 1.28TB per Greenplum database segment for writable s3 tables. You must ensure that the chunksize setting can support the anticipated table size of your table. See [Multipart Upload Overview](http://docs.aws.amazon.com/s3-dev-guide/multipart-upload.html) in the S3 documentation for more information about uploads to S3.

- To take advantage of the parallel processing performed by the Greenplum Database segment instances, the files in the S3 location for read-only S3 tables should be similar in size and the number of files should allow for multiple segments to download the data from the S3 location. For example, if the Greenplum Database system consists of 16 segments and there was sufficient network bandwidth, creating 16 files in the S3 location allows each segment to download a file from the S3 location. In contrast, if the location contained only 1 or 2 files, only 1 or 2 segments download data.

### Using the gpcheckcloud Utility

The Greenplum Database utility gpcheckcloud helps users create an s3 protocol configuration file and test a configuration file. You can specify options to test the ability to access an S3 bucket with a configuration file, and optionally upload data to or download data from files in the bucket.

If you run the utility without any options, it sends a template configuration file to STDOUT. You can capture the output and create an s3 configuration file to connect to Amazon S3.

The utility is installed in the Greenplum Database $GPHOME/bin directory.
Syntax

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

Options

-\texttt{c}

Connect to the specified S3 location with the configuration specified in the s3 protocol URL and return information about the files in the S3 location.

If the connection fails, the utility displays information about failures such as invalid credentials, prefix, or server address (DNS error), or server not available.

-\texttt{d}

Download data from the specified S3 location with the configuration specified in the s3 protocol URL and send the output to STDOUT.

If files are gzip compressed, the uncompressed data is sent to STDOUT.

-\texttt{u}

Upload a file to the S3 bucket specified in the s3 protocol URL using the specified configuration file if available. Use this option to test compression and chunksize and autocompress settings for your configuration.

-\texttt{t}

Sends a template configuration file to STDOUT. You can capture the output and create an s3 configuration file to connect to Amazon S3.

-\texttt{h}

Display gpcheckcloud help.

Examples

This example runs the utility without options to create a template s3 configuration file mytest_s3.config in the current directory.

\begin{verbatim}
gpcheckcloud -t > ./mytest_s3.config
\end{verbatim}

This example attempts to upload a local file, test-data.csv to an S3 bucket location using the s3 configuration file s3.mytestconf:

\begin{verbatim}
gpcheckcloud -u ./test-data.csv "s3://s3-us-west-2.amazonaws.com/test1/abc config=s3.mytestconf"
\end{verbatim}

A successful upload results in one or more files placed in the S3 bucket using the filename format abc<segment_id><random>.data[.gz]. See About S3 Data Files.

This example attempts to connect to an S3 bucket location with the s3 configuration file s3.mytestconf.

\begin{verbatim}
gpcheckcloud -c "s3://s3-us-west-2.amazonaws.com/test1/abc config=s3.mytestconf"
\end{verbatim}
Download all files from the S3 bucket location and send the output to STDOUT.

```bash
gpcheckcloud -d "s3://s3-us-west-2.amazonaws.com/test1/abc
config=s3.mytestconf"
```

## Using a Custom Protocol

A custom protocol allows you to connect Greenplum Database to a data source that cannot be accessed with the `file://`, `gpfdist://`, or `gphdfs://` protocols.

Creating a custom protocol requires that you implement a set of C functions with specified interfaces, declare the functions in Greenplum Database, and then use the `CREATE TRUSTED PROTOCOL` command to enable the protocol in the database.

See [Example Custom Data Access Protocol](#) for an example.

## Handling Errors in External Table Data

By default, if external table data contains an error, the command fails and no data loads into the target database table.

Define the external table with single row error handling to enable loading correctly formatted rows and to isolate data errors in external table data. See [Handling Load Errors](#).

The `gpfdist` file server uses the `HTTP` protocol. External table queries that use `LIMIT` end the connection after retrieving the rows, causing an HTTP socket error. If you use `LIMIT` in queries of external tables that use the `gpfdist://` or `http://` protocols, ignore these errors – data is returned to the database as expected.

## Creating and Using External Web Tables

External web tables allow Greenplum Database to treat dynamic data sources like regular database tables. Because web table data can change as a query runs, the data is not rescannable.

`CREATE EXTERNAL WEB TABLE` creates a web table definition. You can define command-based or URL-based external web tables. The definition forms are distinct: you cannot mix command-based and URL-based definitions.

### Command-based External Web Tables

The output of a shell command or script defines command-based web table data. Specify the command in the `EXECUTE` clause of `CREATE EXTERNAL WEB TABLE`. The data is current as of the time the command runs. The `EXECUTE` clause runs the shell command or script on the specified master, and/or segment host or hosts. The command or script must reside on the hosts corresponding to the host(s) defined in the `EXECUTE` clause.

By default, the command is run on segment hosts when active segments have output rows to process. For example, if each segment host runs four primary segment instances that have output rows to process, the command runs four times per segment host. You can optionally limit the number of segment instances that execute the web table command. All segments included in the web table definition in the `ON` clause run the command in parallel.

The command that you specify in the external table definition executes from the database and cannot access environment variables from `.bashrc` or `.profile`. Set environment variables in the `EXECUTE` clause. For example:

```bash
#$ CREATE EXTERNAL WEB TABLE output (output text)
EXECUTE 'PATH=/home/gpadmin/programs; export PATH; myprogram.sh'
FORMAT 'TEXT';
```
Scripts must be executable by the `gpadmin` user and reside in the same location on the master or segment hosts.

The following command defines a web table that runs a script. The script runs on each segment host where a segment has output rows to process.

```sql
=# CREATE EXTERNAL WEB TABLE log_output
   (linenum int, message text)
  EXECUTE '/var/load_scripts/get_log_data.sh' ON HOST
  FORMAT 'TEXT' (DELIMITER '|');
```

### URL-based External Web Tables

A URL-based web table accesses data from a web server using the HTTP protocol. Web table data is dynamic; the data is not rescannable.

Specify the `LOCATION` of files on a web server using `http://`. The web data file(s) must reside on a web server that Greenplum segment hosts can access. The number of URLs specified corresponds to the number of segment instances that work in parallel to access the web table. For example, if you specify two external files to a Greenplum Database system with eight primary segments, two of the eight segments access the web table in parallel at query runtime.

The following sample command defines a web table that gets data from several URLs.

```sql
=# 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 );
```

### 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**

Creates a readable external table, `ext_expenses`, using the `gpfdist` protocol. The files are formatted with a pipe `(|)` as the column delimiter.

```sql
=# CREATE EXTERNAL TABLE ext_expenses (name text, date date, amount float4, category text, description text)
  LOCATION ('gpfdist://etlhost-1:8081/*')
  FORMAT 'TEXT' (DELIMITER '|');
```
Example 2—Multiple gpfdist instances

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.

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

Example 3—Multiple gpfdists instances

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:// Protocol`.

1. Run `gpfdist` with the `--ssl` option.
2. Run the following command.

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

Example 4—Single gpfdist instance with error logging

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 `Viewing Bad Rows in the Error Log` 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.

```sql
=## 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:

```sql
=## 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;
```
**Example 5—TEXT Format on a Hadoop Distributed File Server**

Creates a readable external table, `ext_expenses`, using the `gphdfs` protocol. The column delimiter is a pipe (`|`).

```sql
=# CREATE EXTERNAL TABLE ext_expenses ( name text, date date, amount float4, category text, desc1 text ) LOCATION ('gphdfs://hdfshost-1:8081/data/filename.txt') FORMAT 'TEXT' (DELIMITER '|');
```

gphdfs requires only one data path.

For examples of reading and writing custom formatted data on a Hadoop Distributed File System, see Reading and Writing Custom-Formatted HDFS Data.

**Example 6—Multiple files in CSV format with header rows**

Creates a readable external table, `ext_expenses`, using the `file` protocol. The files are CSV format and have a header row.

```sql
=# 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);
```

**Example 7—Readable External Web Table with Script**

Creates a readable external web table that executes a script once per segment host:

```sql
=# CREATE EXTERNAL WEB TABLE log_output (linenum int, message text) EXECUTE '/var/load_scripts/get_log_data.sh' ON HOST FORMAT 'TEXT' (DELIMITER '|');
```

**Example 8—Writable External Table with gpfdist**

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.

```sql
=# CREATE WRITABLE EXTERNAL TABLE sales_out (LIKE sales) LOCATION ('gpfdist://etl1:8081/sales.out') FORMAT 'TEXT' ( DELIMITER '|' NULL ' ') DISTRIBUTED BY (txn_id);
```

**Example 9—Writable External Web Table with Script**

Creates a writable external web table, `campaign_out`, that pipes output data received by the segments to an executable script, `to_adreport_etl.sh`:

```sql
=# CREATE WRITABLE EXTERNAL WEB TABLE campaign_out (LIKE campaign) EXECUTE '/var/unload_scripts/to_adreport_etl.sh' FORMAT 'TEXT' (DELIMITER '|');
```
**Example 10—Readable and 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 *Transforming XML Data*.

**Accessing HDFS and Hive Data with PXF**

Data managed by your organization may already reside in external sources. The Greenplum Database PXF Extension Framework (PXF) provides access to this external data via built-in connectors that map an external data source to a Greenplum Database table definition.

PXF is installed with HDFS and Hive connectors. These connectors enable you to read external HDFS file system and Hive table data stored in text, Avro, RCFile, Parquet, SequenceFile, and ORC formats.

The PXF Extension Framework includes a protocol C library and a Java service. After you configure and initialize PXF, you start a single PXF JVM process on each Greenplum Database segment host. This long-running process concurrently serves multiple query requests.

For detailed information about the architecture of and using the PXF Extension Framework, refer to the *Using PXF with External Data* documentation.

**Accessing HDFS Data with gphdfs**

Greenplum Database leverages the parallel architecture of a Hadoop Distributed File System to read and write data files efficiently using the `gphdfs` protocol.

There are three steps to using the `gphdfs` protocol with HDFS:

- One-time `gphdfs` Protocol Installation
- Grant Privileges for the `gphdfs` Protocol
- Specify `gphdfs` Protocol in an External Table Definition

For information about using Greenplum Database external tables with Amazon EMR when Greenplum Database is installed on Amazon Web Services (AWS), also see *Using Amazon EMR with Greenplum Database installed on AWS*.

**One-time gphdfs Protocol Installation**

Install and configure Hadoop for use with `gphdfs` as follows:

1. Install Java 1.7 or later on all Greenplum Database hosts: master, segment, and standby master.
2. Install a compatible Hadoop distribution on all hosts. The distribution must be the same on all hosts. For Hadoop installation information, see the Hadoop distribution documentation.
   
   See the *Greenplum Database Release Notes* for information about compatible Hadoop distributions.

3. After installation, ensure that the Greenplum system user (`gpadmin`) has read and execute access to the Hadoop libraries or to the Greenplum MR client.

4. Set the following environment variables on all segments:
   - `JAVA_HOME` – the Java home directory
   - `HADOOP_HOME` – the Hadoop home directory

   For example, add lines such as the following to the `gpadmin` user `.bashrc` profile.

   ```bash
   export JAVA_HOME=/usr/java/default
   export HADOOP_HOME=/usr/lib/gphd
   ```

   The variables must be set in the `~gpadmin/.bashrc` or the `~gpadmin/.bash_profile` file so that the `gpadmin` user shell environment can locate the Java home and Hadoop home.
5. Set the following Greenplum Database server configuration parameters and restart Greenplum Database.

Table 50: Server Configuration Parameters for Hadoop Targets

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp_hadoop_target_version</td>
<td>The Hadoop target. Choose one of the following. \n  cdh5 \n  cdh4.1 \n  hdp2 \n  gpmr-1.2 \n  hadoop2</td>
<td>gphd-1.1</td>
<td>master session reload</td>
</tr>
<tr>
<td>gp_hadoop_home</td>
<td>With Pivotal HD, this parameter specifies the installation directory for Hadoop. For example, the default installation directory is /usr/lib/gphd. \n  When using Greenplum HD 1.2 or earlier, specify the same value as the HADOOP_HOME environment variable.</td>
<td>NULL</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

For example, the following commands use the Greenplum Database utilities `gpconfig` and `gpstop` to set the server configuration parameters and restart Greenplum Database:

```
gpconfig -c gp_hadoop_target_version -v 'hdb2'
gpstop -u
```

For information about the Greenplum Database utilities `gpconfig` and `gpstop`, see the Greenplum Database Utility Guide.

6. If needed, ensure that the `CLASSPATH` environment variable generated by the `$GPHOME/lib/hadoop/hadoop_env.sh` file on every Greenplum Database host contains the path to JAR files that contain Java classes that are required for `gphdfs`.

For example, if `gphdfs` returns a class not found exception, ensure the JAR file containing the class is on every Greenplum Database host and update the `$GPHOME/lib/hadoop/hadoop_env.sh` file so that the `CLASSPATH` environment variable created by file contains the JAR file.

### About gphdfs JVM Memory

When Greenplum Database accesses external table data from an HDFS location with `gphdfs` protocol, each Greenplum Database segment on a host system starts a JVM for use by the protocol. The default JVM heapsize is 1GB and should be enough for most workloads.

If the `gphdfs` JVM runs out of memory, the issue might be related to the density of tuples inside the Hadoop HDFS block assigned to the `gphdfs` segment worker. A higher density of tuples per block requires more `gphdfs` memory. HDFS block size is usually 128MB, 256MB, or 512MB depending on the Hadoop cluster configuration.
You can increase the JVM heapsize by changing the `GP_JAVA_OPT` variable in the file `$GPHOME/lib/hadoop/hadoop_env.sh`. In this example line, the option `-Xmx1000m` specifies that the JVM consumes 1GB of virtual memory.

```bash
export GP_JAVA_OPT='-Xmx1000m -XX:+DisplayVMOutputToStderr'
```

The `$GPHOME/lib/hadoop/hadoop_env.sh` must be updated for every segment instance in the Greenplum Database system.

**Important:** Before increasing the gphdfs JVM memory, ensure that you have sufficient memory on the host. For example, 8 primary segments consume 8GB of virtual memory for the gphdfs JVM when using default. Increasing the Java `-Xmx` value to 2GB results in 16GB allocated in that environment of 8 segments per host.

### Grant Privileges for the gphdfs Protocol

To enable privileges required to create external tables that access files on HDFS using gphdfs:

1. Grant the following privileges on gphdfs to the owner of the external table.
   - Grant `SELECT` privileges to enable creating readable external tables on HDFS.
   - Grant `INSERT` privileges to enable creating writable external tables on HDFS.

   Use the `GRANT` command to grant read privileges (`SELECT`) and, if needed, write privileges (`INSERT`) on HDFS to the Greenplum system user (gpadmin).

   ```sql
   GRANT INSERT ON PROTOCOL gphdfs TO gpadmin;
   ```

2. Greenplum Database uses OS credentials to connect to HDFS. Grant read privileges and, if needed, write privileges to HDFS to the Greenplum administrative user (gpadmin OS user).

### Specify gphdfs Protocol in an External Table Definition

The `gphdfs` LOCATION clause of the `CREATE EXTERNAL TABLE` command for HDFS files differs slightly for Hadoop HA (High Availability) clusters, Hadoop clusters without HA, and MapR clusters.

In a Hadoop HA cluster, the LOCATION clause references the logical nameservices id (the `dfs.nameservices` property in the `hdfs-site.xml` configuration file). The `hdfs-site.xml` file with the nameservices configuration must be installed on the Greenplum master and on each segment host.

For example, if `dfs.nameservices` is set to `mycluster` the LOCATION clause takes this format:

```sql
LOCATION ('gphdfs://mycluster/path/filename.txt')
```

A cluster without HA specifies the hostname and port of the name node in the LOCATION clause:

```sql
LOCATION ('gphdfs://hdfs_host[:port]/path/filename.txt')
```

If you are using MapR clusters, you specify a specific cluster and the file:

- To specify the default cluster, the first entry in the MapR configuration file `/opt/mapr/conf/mapr-clusters.conf`, specify the location of your table with this syntax:

  ```sql
  LOCATION ('gphdfs:///file_path')
  ```

  The `file_path` is the path to the file.

- To specify another MapR cluster listed in the configuration file, specify the file with this syntax:

  ```sql
  LOCATION ('gphdfs:///mapr/cluster_name/file_path')
  ```
The *cluster_name* is the name of the cluster specified in the configuration file and *file_path* is the path to the file.

For information about MapR clusters, see the MapR documentation.

Restrictions for HDFS files are as follows.

- You can specify one path for a readable external table with `gphdfs`. Wildcard characters are allowed. If you specify a directory, the default is all files in the directory.

You can specify only a directory for writable external tables.

- The URI of the `LOCATION` clause cannot contain any of these four characters: `,", '<', '>'. The `CREATE EXTERNAL TABLE` returns an error if the URI contains any of the characters.

- Format restrictions are as follows.
  - Only the `gphdfs_import` formatter is allowed for readable external tables with a custom format.
  - Only the `gphdfs_export` formatter is allowed for writable external tables with a custom format.
  - You can set compression only for writable external tables. Compression settings are automatic for readable external tables.

### Setting Compression Options for Hadoop Writable External Tables

Compression options for Hadoop Writable External Tables use the form of a URI query and begin with a question mark. Specify multiple compression options with an ampersand (`&`).

Table 51: Compression Options

<table>
<thead>
<tr>
<th>Compression Option</th>
<th>Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>compress</code></td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td><code>compression_type</code></td>
<td>BLOCK or RECORD</td>
<td>RECORD</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>For AVRO format, <code>compression_type</code> must be block if <code>compress</code> is true.</em></td>
</tr>
<tr>
<td><code>codec</code></td>
<td>Codec class name</td>
<td>GzipCodec for text format and DefaultCodec for <code>gphdfs_export</code> format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>For AVRO format, the value is either deflate (the default) or snappy</em></td>
</tr>
<tr>
<td><code>codec_level</code> (for AVRO format and deflate codec only)</td>
<td>integer between 1 and 9</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>The level controls the trade-off between speed and compression. Valid values are 1 to 9, where 1 is the fastest and 9 is the most compressed.</em></td>
</tr>
</tbody>
</table>

Place compression options in the query portion of the URI.

### Support for Avro Files

You can use the Greenplum Database `gphdfs` protocol to access Avro files on a Hadoop file system (HDFS).

### About the Avro File Format

An Avro file stores both the data definition (schema) and the data together in one file making it easy for programs to dynamically understand the information stored in an Avro file. The Avro schema is in JSON format, the data is in a binary format making it compact and efficient.
The following example Avro schema defines an Avro record with 3 fields:

- name
- favorite_number
- favorite_color

```json
{"namespace": "example.avro", "type": "record", "name": "User", "fields": [
  {"name": "name", "type": "string"},
  {"name": "favorite_number", "type": ["int", "null"]},
  {"name": "favorite_color", "type": ["string", "null"]}
]}
```

These are two rows of data based on the schema:

```json
{ "name" : "miguno" , "favorite_number" : 6 , "favorite_color" : "red" } 
{ "name" : "BlizzardCS" , "favorite_number" : 21 , "favorite_color" : "green" }
```

For information about the Avro file format, see [http://avro.apache.org/docs/1.7.7/](http://avro.apache.org/docs/1.7.7/)

### Required Avro Jar Files

Support for the Avro file format requires these jar files:

- avro-1.7.7.jar
- avro-tools-1.7.7.jar
- avro-mapred-1.7.5-hadoop2.jar (available with Apache Pig)

**Note:** Hadoop 2 distributions include the Avro jar file $HADOOP_HOME/share/hadoop/common/lib/avro-1.7.4.jar. To avoid conflicts, you can rename the file to another file such as avro-1.7.4.jar.bak.

For the Cloudera 5.4.x Hadoop distribution, only the jar file avro-mapred-1.7.5-hadoop2.jar needs to be downloaded and installed. The distribution contains the other required jar files. The other files are included in the classpath used by the gphdfs protocol.

For information about downloading the Avro jar files, see [https://avro.apache.org/releases.html](https://avro.apache.org/releases.html).

On all the Greenplum Database hosts, ensure that the jar files are installed and are on the classpath used by the gphdfs protocol. The classpath is specified by the shell script $GPHOME/lib/hadoop/hadoop_env.sh.

As an example, if the directory $HADOOP_HOME/share/hadoop/common/lib does not exist, create it on all Greenplum Database hosts as the gpadmin user. Then, add the add the jar files to the directory on all hosts.

The hadoop_env.sh script file adds the jar files to `classpath` for the gphdfs protocol. This fragment in the script file adds the jar files to the classpath.

```bash
if [ -d "${HADOOP_HOME}/share/hadoop/common/lib" ]; then
  for f in ${HADOOP_HOME}/share/hadoop/common/lib/*.jar; do
    CLASSPATH=${CLASSPATH}:${f};
  done
```

### Avro File Format Support

The Greenplum Database gphdfs protocol supports the Avro file type as an external table:

- Avro file format - GPDB certified with Avro version 1.7.7
• Reading and writing Avro files
• Support for overriding the Avro schema when reading an Avro file
• Compressing Avro files during writing
• Automatic Avro schema generation when writing an Avro file

Greenplum Database returns an error if the Avro file contains unsupported features or if the specified schema does not match the data.

**Reading from and Writing to Avro Files**

To read from or write to an Avro file, you create an external table and specify the location of the Avro file in the `LOCATION` clause and 'AVRO' in the `FORMAT` clause. For example, this is the syntax for a readable external table.

```sql
CREATE EXTERNAL TABLE tablename (column_spec) LOCATION ('gphdfs://location') FORMAT 'AVRO'
```

The `location` can be an individual Avro file or a directory containing a set of Avro files. If the location specifies multiple files (a directory name or a file name containing wildcard characters), Greenplum Database uses the schema in the first file of the directory as the schema of the whole directory. For the file name you can specify the wildcard character * to match any number of characters.

You can add parameters after the file specified in the `location`. You add parameters with the http query string syntax that starts with `?` and `&` between field and value pairs.

For readable external tables, the only valid parameter is `schema`. The gphdfs uses this schema instead of the Avro file schema when reading Avro files. See [Avro Schema Overrides for Readable External Tables](#).

For writable external tables, you can specify `schema`, `namespace`, and parameters for compression.

**Table 52: Avro External Table Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Readable/Writable</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td><code>URL_to_schema_file</code></td>
<td>Read and Write</td>
<td>None.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For a readable external table</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• The specified schema overrides the schema in the Avro file. See <a href="#">Avro Schema Overrides</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• If not specified, Greenplum Database uses the Avro file schema.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For a writable external table</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Uses the specified schema when creating the Avro file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• If not specified, Greenplum Database creates a schema according to the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>external table definition.</td>
</tr>
<tr>
<td>namespace</td>
<td><code>avro_namespace</code></td>
<td>Write only</td>
<td><code>public.avro</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If specified, a valid Avro <code>namespace</code>.</td>
</tr>
<tr>
<td>compress</td>
<td><code>true</code> or <code>false</code></td>
<td>Write only</td>
<td><code>false</code>.</td>
</tr>
<tr>
<td>compression_type</td>
<td><code>block</code></td>
<td>Write only</td>
<td>Optional.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For <code>avro</code> format, <code>compression_type</code> must be <code>block</code> if <code>compress</code> is <code>true</code>.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
<td>Readable/Writable</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------</td>
<td>-------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>codec</td>
<td>deflate or snappy</td>
<td>Write only</td>
<td>deflate</td>
</tr>
<tr>
<td>codec_level</td>
<td>integer between 1 and 9</td>
<td>Write only</td>
<td>6</td>
</tr>
</tbody>
</table>

The level controls the trade-off between speed and compression. Valid values are 1 to 9, where 1 is the fastest and 9 is the most compressed.

This set of parameters specify **snappy** compression:

```
'compress=true&codec=snappy'
```

These two sets of parameters specify **deflate** compression and are equivalent:

```
'compress=true&codec=deflate&codec_level=1'
'compress=true&codec_level=1'
```

### Data Conversion When Reading Avro Files

When you create a readable external table to Avro file data, Greenplum Database converts Avro data types to Greenplum Database data types.

**Note:** When reading an Avro, Greenplum Database converts the Avro field data at the top level of the Avro schema to a Greenplum Database table column. This is how the gphdfs protocol converts the Avro data types.

- An Avro primitive data type, Greenplum Database converts the data to a Greenplum Database type.
- An Avro complex data type that is not map or record, Greenplum Database converts the data to a Greenplum Database type.
- An Avro record that is a sub-record (nested within the top level Avro schema record), Greenplum Database converts the data XML.

This table lists the Avro primitive data types and the Greenplum Database type it is converted to.

#### Table 53: Avro Primitive Data Type Support for Readable External Tables

<table>
<thead>
<tr>
<th>Avro Data Type</th>
<th>Greenplum Database Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>Supported only in an Avro union data type. See <a href="#">Data Conversion when Writing Avro Files</a>.</td>
</tr>
<tr>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>int</td>
<td>int or smallint</td>
</tr>
<tr>
<td>long</td>
<td>bigint</td>
</tr>
<tr>
<td>float</td>
<td>real</td>
</tr>
<tr>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>bytes</td>
<td>bytea</td>
</tr>
<tr>
<td>string</td>
<td>text</td>
</tr>
</tbody>
</table>
**Note:** When reading the Avro `int` data type as Greenplum Database `smallint` data type, you must ensure that the Avro `int` values do not exceed the Greenplum Database maximum `smallint` value. If the Avro value is too large, the Greenplum Database value will be incorrect.

The `gphdfs` protocol converts performs this conversion for `smallint`:  

```java
(short)IntValue;
```

This table lists the Avro complex data types and the and the Greenplum Database type it is converted to.

**Table 54: Avro Complex Data Type Support for Readable External Tables**

<table>
<thead>
<tr>
<th>Avro Data Type</th>
<th>Greenplum Database Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>enum</code></td>
<td><code>int</code></td>
</tr>
<tr>
<td></td>
<td>The integer represents the zero-based position of the symbol in the schema.</td>
</tr>
<tr>
<td><code>array</code></td>
<td><code>array</code></td>
</tr>
<tr>
<td></td>
<td>The Greenplum Database array dimensions match the Avro array dimensions. The element type is converted from the Avro data type to the Greenplum Database data type.</td>
</tr>
<tr>
<td><code>maps</code></td>
<td>Not supported</td>
</tr>
<tr>
<td><code>union</code></td>
<td>The first non-null data type.</td>
</tr>
<tr>
<td><code>fixed</code></td>
<td><code>bytea</code></td>
</tr>
<tr>
<td><code>record</code></td>
<td>XML data</td>
</tr>
</tbody>
</table>

**Example Avro Schema**

This is an example Avro schema. When reading the data from the Avro file the `gphdfs` protocol performs these conversions:

- `name` and `color` data are converted to Greenplum Database `string`.
- `age` data is converted to Greenplum Database `int`.
- `clist` records are converted to XML.

```json
{"namespace": "example.avro",
 "type": "record",
 "name": "User",
 "fields": [
 {"name": "name", "type": "string"},
 {"name": "number", "type": ["int", "null"]},
 {"name": "color", "type": ["string", "null"]},
 {"name": "clist",
  "type": {"type":"record",
             "name":"clistRecord",
             "fields":[
              {"name": "class", "type": ["string", "null"]},
              {"name": "score", "type": ["double", "null"]},
              {"name": "grade",
               "type": {"type":"record",
                          "name":"inner2",
                          "fields":[
                           {"name": "a", "type": ["double", "null"]},
                           {"name": "b", "type": ["string", "null"]}
                          ]
               }
              ]
             }]
  }
},
```
This XML is an example of how the `gpfist` protocol converts Avro data from the `clist` field to XML data based on the previous schema. For records nested in the Avro top-level record, `gpfist` protocol converts the Avro element name to the XML element name and the name of the record is an attribute of the XML element. For example, the name of the top most element `clist` and the `type` attribute is the name of the Avro record element `clistRecord`.

```
<clist type="clistRecord">
  <class type="string">math</class>
  <score type="double">99.5</score>
  <grade type="inner2">
    <a type="double">88.8</a>
    <b type="string">subb0</b>
  </grade>
  <grade2 type="inner">
    <a type="double">77.7</a>
    <b type="string">subb20</b>
    <c type="inner3">
      <c1 type="string">subc</c1>
      <c2 type="int">0</c2>
    </c>
  </grade2>
</clist>
```

**Avro Schema Overrides for Readable External Tables**

When you specify schema for a readable external table that specifies an Avro file as a source, Greenplum Database uses the schema when reading data from the Avro file. The specified schema overrides the Avro file schema.

You can specify a file that contains an Avro schema as part of the location parameter `CREATE EXTERNAL TABLE` command, to override the Avro file schema. If a set of Avro files contain different, related schemas, you can specify an Avro schema to retrieve the data common to all the files.

Greenplum Database extracts the data from the Avro files based on the field name. If an Avro file contains a field with same name, Greenplum Database reads the data, otherwise a `NULL` is returned.

For example, if a set of Avro files contain one of the two different schemas. This is the original schema.

```
{
  "name": "grade2",
  "type": {
    "type": "record",
    "name": "inner",
    "fields": [
      {"name": "a", "type": ["double", "null"]},
      {"name": "b", "type": ["string", "null"]},
      {"name": "c", "type": {
        "type": "record",
        "name": "inner3",
        "fields": [
          {"name": "c1", "type": ["string", "null"]},
          {"name": "c2", "type": ["int", "null"]}
        ]
      }
    ]
  }
}
```
This updated schema contains a comment field.

```
{  
  "type":"record",  
  "name":"tav2",  
  "namespace":"public.avro",  
  "doc":null,  
  "fields":[
    {
      "name":"id","type":null,"int","doc":null},
    {
      "name":"name","type":null,"string","doc":null},
    {
      "name":"birth","type":null,"string","doc":null},
    {
      "name":"age","type":null,"long","doc":null},
    {
      "name":"comment","type":null,"string","doc":null}
  ]
}
```

You can specify an file containing this Avro schema in a CREATE EXTERNAL TABLE command, to read the id, name, birth, and comment fields from the Avro files.

```
{  
  "type":"record",  
  "name":"tav2",  
  "namespace":"public.avro",  
  "doc":null,  
  "fields":[
    {
      "name":"id","type":null,"int","doc":null},
    {
      "name":"name","type":null,"string","doc":null},
    {
      "name":"birth","type":null,"string","doc":null},
    {
      "name":"age","type":null,"long","doc":null},
    {
      "name":"comment","type":null,"string","doc":null}
  ]
}
```

In this example command, the customer data is in the Avro files tmp/cust*.avro. Each file uses one of the schemas listed previously. The file avro/cust.avsc is a text file that contains the Avro schema used to override the schemas in the customer files.

```
CREATE WRITABLE EXTERNAL TABLE cust_avro(id int, name text, birth date)
LOCATION ('gphdfs://my_hdfs:8020/tmp/cust*.avro
?schema=hdfs://my_hdfs:8020/avro/cust.avsc')
FORMAT 'avro';
```

When reading the Avro data, if Greenplum Database reads a file that does not contain a comment field, a NULL is returned for the comment data.

**Data Conversion when Writing Avro Files**

When you create a writable external table to write data to an Avro file, each table row is an Avro record and each table column is an Avro field. When writing an Avro file, the default compression algorithm is deflate.
For a writable external table, if the `schema` option is not specified, Greenplum Database creates an Avro schema for the Avro file based on the Greenplum Database external table definition. The name of the table column is the Avro field name. The data type is a union data type. See the following table:

### Table 55: Avro Data Types Generated by Greenplum Database

<table>
<thead>
<tr>
<th>Greenplum Database Data Type</th>
<th>Avro Union Data Type Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>[&quot;boolean&quot;, &quot;null&quot;]</td>
</tr>
<tr>
<td>int</td>
<td>[&quot;int&quot;, &quot;null&quot;]</td>
</tr>
<tr>
<td>bigint</td>
<td>[&quot;long&quot;, &quot;null&quot;]</td>
</tr>
<tr>
<td>smallint</td>
<td>[&quot;int&quot;, &quot;null&quot;]</td>
</tr>
<tr>
<td>real</td>
<td>[&quot;float&quot;, &quot;null&quot;]</td>
</tr>
<tr>
<td>double</td>
<td>[&quot;double&quot;, &quot;null&quot;]</td>
</tr>
<tr>
<td>bytea</td>
<td>[&quot;bytes&quot;, &quot;null&quot;]</td>
</tr>
<tr>
<td>text</td>
<td>[&quot;string&quot;, &quot;null&quot;]</td>
</tr>
<tr>
<td>array</td>
<td>[&quot;string&quot;, &quot;null&quot;]</td>
</tr>
</tbody>
</table>

The Greenplum Database array is converted to an Avro array with same dimensions and same element type as the Greenplum Database array.

| other data types            | ["string", "null"]             |

Data are formatted strings. The `gphdfs` protocol casts the data to Greenplum Database text and writes the text to the Avro file as an Avro string. For example, date and time data are formatted as date and time strings and converted to Avro string type.

You can specify a schema with the `schema` option. When you specify a schema, the file can be on the segment hosts or a file on the HDFS that is accessible to Greenplum Database. For a local file, the file must exist in all segment hosts in the same location. For a file on the HDFS, the file must exist in the same cluster as the data file.

This example `schema` option specifies a schema on an HDFS.

```
'schema=hdfs://mytest:8000/avro/array_simple.avsc'
```

This example `schema` option specifies a schema on the host file system.

```
'schema=file:///mydata/avro_schema/array_simple.avsc'
```

**gphdfs Limitations for Avro Files**

For a Greenplum Database writable external table definition, columns cannot specify the `NOT NULL` clause.

Greenplum Database supports only a single top-level schema in Avro files or specified with the `schema` parameter in the `CREATE EXTERNAL TABLE` command. An error is returned if Greenplum Database detects multiple top-level schemas.

Greenplum Database does not support the Avro `map` data type and returns an error when encountered.

When Greenplum Database reads an array from an Avro file, the array is converted to the literal text value. For example, the array `[1,3]` is converted to `'\{1,3\}'`. 
User defined types (UDT), including array UDT, are supported. For a writable external table, the type is converted to string.

**Examples**

Simple `CREATE EXTERNAL TABLE` command that reads data from the two Avro fields `id` and `ba`.

```sql
CREATE EXTERNAL TABLE avro1 (id int, ba bytea[]) LOCATION ('gphdfs://my_hdfs:8020/avro/singleAvro/array2.avro') FORMAT 'avro';
```

`CREATE WRITABLE EXTERNAL TABLE` command specifies the Avro schema that is the `gphdfs` protocol uses to create the Avro file.

```sql
CREATE WRITABLE EXTERNAL TABLE at1w(id int, names text[], nums int[]) LOCATION ('gphdfs://my_hdfs:8020/tmp/at1?schema=hdfs://my_hdfs:8020/avro/array_simple.avsc') FORMAT 'avro';
```

`CREATE WRITABLE EXTERNAL TABLE` command that writes to an Avro file and specifies a namespace for the Avro schema.

```sql
CREATE WRITABLE EXTERNAL TABLE atudt1 (id int, info myt, birth date, salary numeric) LOCATION ('gphdfs://my_hdfs:8020/tmp/emp01.avro?namespace=public.example.avro') FORMAT 'avro';
```

**Support for Parquet Files**

You can use the Greenplum Database `gphdfs` protocol to access Parquet files on a Hadoop file system (HDFS).

**About the Parquet File Format**

The Parquet file format is designed to take advantage of compressed, efficient columnar data representation available to projects in the Hadoop ecosystem. Parquet supports complex nested data structures and uses Dremel record shredding and assembly algorithms. Parquet supports very efficient compression and encoding schemes. Parquet allows compression schemes to be specified on a per-column level, and supports adding more encodings as they are invented and implemented.


**Required Parquet Jar Files**

Support for the Parquet file format requires these jar files:

- parquet-hadoop-1.7.0.jar
- parquet-common-1.7.0.jar
- parquet-encoding-1.7.0.jar
- parquet-column-1.7.0.jar
- parquet-generator-1.7.0.jar
- parquet-format-2.3.0-incubating.jar
**Note:** The Cloudera 5.4.x Hadoop distribution includes some Parquet jar files. However, the Java class names in the jar files are `parquet.xxx`. The `gphdfs` protocol uses the Java class names `org.apache.parquet.xxx`. The jar files with the class name `org.apache.parquet` can be downloaded and installed on the Greenplum Database hosts.

The `gphdfs` protocol also supports using `parquet-hadoop-bundle-1.7.0.jar` that contains the classes required to use Parquet within a Hadoop environment. These versions of `parquet-hadoop-bundle` are not supported:

- Version 1.6 and earlier. The versions do not use the Java class names `org.apache.parquet`
- Version 1.8 and later. The versions contain the class `VersionParser` that is not supported by `gphdfs`.

For information about downloading the Parquet jar files, see [http://parquet.apache.org/downloads/](http://parquet.apache.org/downloads/)

On all the Greenplum Database hosts, ensure that the jar files are installed and are on the classpath used by the `gphdfs` protocol. The `classpath` is specified by the shell script `$GPHOME/lib/hadoop/hadoop_env.sh`. As a Hadoop 2 example, you can install the jar files in `$HADOOP_HOME/share/hadoop/common/lib`. The `hadoop_env.sh` script file adds the jar files to the `classpath`.

As an example, if the directory `$HADOOP_HOME/share/hadoop/common/lib` does not exist, create it on all Greenplum Database hosts as the `gpadmin` user. Then, add the add the jar files to the directory on all hosts.

The `hadoop_env.sh` script file adds the jar files to `classpath` for the `gphdfs` protocol. This fragment in the script file adds the jar files to the `classpath`.

```bash
if [ -d "${HADOOP_HOME}/share/hadoop/common/lib" ]; then
  for f in ${HADOOP_HOME}/share/hadoop/common/lib/*.jar; do
    CLASSPATH=${CLASSPATH}:${f};
  done

Parquet File Format Support

The Greenplum Database `gphdfs` protocol supports the Parquet file format version 1 or 2. Parquet takes advantage of compressed, columnar data representation on HDFS. In a Parquet file, the metadata (Parquet schema definition) contains data structure information is written after the data to allow for single pass writing.

This is an example of the Parquet schema definition format:

```sql
message test {
  repeated byte_array binary_field;
  required int32 int32_field;
  optional int64 int64_field;
  required boolean boolen_field;
  required fixed_len_byte_array(3) flba_field;
  required byte_array someDay (utf8);
}
```

The definition for last field `someDay` specifies the `binary` data type with the `utf8` annotation. The data type and annotation defines the data as a UTF-8 encoded character string.

**Reading from and Writing to Parquet Files**

To read from or write to a Parquet file, you create an external table and specify the location of the parquet file in the `LOCATION` clause and `PARQUET` in the `FORMAT` clause. For example, this is the syntax for a readable external table.

```sql
CREATE EXTERNAL TABLE tablename (column_spec) LOCATION
  ( 'gphdfs://location') FORMAT 'PARQUET'
```
The *location* can be an Parquet file or a directory containing a set of Parquet files. For the file name you can specify the wildcard character * to match any number of characters. If the location specifies multiple files when reading Parquet files, Greenplum Database uses the schema in the first file that is read as the schema for the other files.

**Reading a Parquet File**

The following table lists how Greenplum database converts the Parquet data type if the Parquet schema definition does not contain an annotation.

**Table 56: Data Type Conversion when Reading a Parquet File**

<table>
<thead>
<tr>
<th>Parquet Data Type</th>
<th>Greenplum Database Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>int32</td>
<td>int or smallint</td>
</tr>
<tr>
<td>int64</td>
<td>long</td>
</tr>
<tr>
<td>int96</td>
<td>byte</td>
</tr>
<tr>
<td>float</td>
<td>real</td>
</tr>
<tr>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>fixed_lenth_byte_array</td>
<td>byte</td>
</tr>
<tr>
<td>byte_array</td>
<td>byte</td>
</tr>
</tbody>
</table>

**Note:** When reading the Parquet int data type as Greenplum Database smallint data type, you must ensure that the Parquet int values do not exceed the Greenplum Database maximum smallint value. If the value is too large, the Greenplum Database value will be incorrect.

The *gphdfs* protocol considers Parquet schema annotations for these cases. Otherwise, data conversion is based on the parquet schema primitive type:

**Table 57: Data Type (with Annotation) Conversion when Reading Parquet File**

<table>
<thead>
<tr>
<th>Parquet Schema Data Type and Annotation</th>
<th>Greenplum Database Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary with json or utf8 annotation</td>
<td>text</td>
</tr>
<tr>
<td>binary and the Greenplum Database column data type is text</td>
<td>text</td>
</tr>
<tr>
<td>int32 with int_16 annotation</td>
<td>smallint</td>
</tr>
<tr>
<td>int32, int64, fixed_lenth_byte_array, or binary with decimal annotation</td>
<td>decimal</td>
</tr>
<tr>
<td>repeated</td>
<td>array column - The data type is converted according to Table 56: Data Type Conversion when Reading a Parquet File</td>
</tr>
<tr>
<td>optional, required</td>
<td>Data type is converted according to Table 56: Data Type Conversion when Reading a Parquet File</td>
</tr>
</tbody>
</table>

**Note:** See *Limitations and Notes* and the Parquet documentation when specifying decimal, date, interval, or time* annotations.
The `gphdfs` protocol converts the field data to text if the Parquet field type is binary without any annotation, and the data type is defined as text for the corresponding Greenplum Database external table column.

When reading Parquet type group, the `gphdfs` protocol converts the group data into an XML document. This schema contains a required group with the name `inner`.

```java
message test {
  required byte_array binary_field;
  required int64 int64_field;
  required group inner {
    int32 age;
    required boolean test;
    required byte_array name (UTF8);
  }
};
```

This how a single row of the group data would be converted to XML.

```xml
<inner type="group">
  <age type="int">50</age>
  <test type="boolean">true</test>
  <name type="string">fred</name>
</inner>
```

This example schema contains a repeated group with the name `inner`.

```java
message test {
  required byte_array binary_field;
  required int64 int64_field;
  repeated group inner {
    int32 age;
    required boolean test;
    required byte_array name (UTF8);
  }
};
```

For a repeated group, the Parquet file can contain multiple sets of the group data in a single row. For the example schema, the data for the `inner` group is converted into XML data.

This is sample output if the data in the Parquet file contained two sets of data for the `inner` group.

```xml
<inner type="repeated">
  <inner type="group">
    <age type="int">50</age>
    <test type="boolean">true</test>
    <name type="string">fred</name>
  </inner>
  <inner>
    <age type="int">23</age>
    <test type="boolean">false</test>
    <name type="string">sam</name>
  </inner>
</inner>
```

**Reading a Hive Generated Parquet File**

The Apache Hive data warehouse software can manage and query large datasets that reside in distributed storage. Apache Hive 0.13.0 and later can store data in Parquet format files. For information about Parquet used by Apache Hive, see [https://cwiki.apache.org/confluence/display/Hive/Parquet](https://cwiki.apache.org/confluence/display/Hive/Parquet).
For Hive 1.1 data stored in Parquet files, this table lists how Greenplum database converts the data. The conversion is based on the Parquet schema that is generated by Hive. For information about the Parquet schema generated by Hive, see Notes on the Hive Generated Parquet Schema.

**Table 58: Data Type Conversion when Reading a Hive Generated Parquet File**

<table>
<thead>
<tr>
<th>Hive Data Type</th>
<th>Greenplum Database Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>tinyint</td>
<td>int</td>
</tr>
<tr>
<td>smallint</td>
<td>int</td>
</tr>
<tr>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td>bigint</td>
<td>bigint</td>
</tr>
<tr>
<td>decimal</td>
<td>numeric</td>
</tr>
<tr>
<td>float</td>
<td>real</td>
</tr>
<tr>
<td>double</td>
<td>float</td>
</tr>
<tr>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>string</td>
<td>text</td>
</tr>
<tr>
<td>char</td>
<td>text or char</td>
</tr>
<tr>
<td>varchar</td>
<td>text or varchar</td>
</tr>
<tr>
<td>timestamp</td>
<td>bytea</td>
</tr>
<tr>
<td>binary</td>
<td>bytea</td>
</tr>
<tr>
<td>array</td>
<td>xml</td>
</tr>
<tr>
<td>map</td>
<td>xml</td>
</tr>
<tr>
<td>struct</td>
<td>xml</td>
</tr>
</tbody>
</table>

**Notes on the Hive Generated Parquet Schema**

- When writing data to Parquet files, Hive treats all integer data types `tinyint, smallint, int` as `int32`. When you create an external table in Greenplum Database for a Hive generated Parquet file, specify the column data type as `int`. For example, this Hive `CREATE TABLE` command stores data in Parquet files.

```sql
CREATE TABLE hpSimple(c1 tinyint, c2 smallint, c3 int, c4 bigint, c5 float, c6 double, c7 boolean, c8 string) STORED AS PARQUET;
```

This is the Hive generated Parquet schema for the `hpSimple` table data.

```java
message hive_schema {
  optional int32 c1;
  optional int32 c2;
  optional int32 c3;
  optional int64 c4;
  optional float c5;
  optional double c6;
  optional boolean c7;
  optional binary c8 (UTF8);
}
```
The `gphdfs` protocol converts the Parquet integer data types to the Greenplum Database data type `int`.

- For the Hive `char` data type, the Greenplum Database column data types can be either `text` or `char`. For the Hive `varchar` data type, the Greenplum Database column data type can be either `text` or `varchar`.

- Based on the Hive generated Parquet schema, some Hive data is converted to Greenplum Database XML data. For example, Hive array column data that is stored in a Parquet file is converted to XML data. As an example, this is the Hive generated Parquet schema for a Hive column `col1` of data type `array[int]`.

```java
optional group col1 (LIST) {
  repeated group bag {
    optional int32 array_element;
  }
}
```

The `gphdfs` protocol converts the Parquet group data to the Greenplum Database data type `XML`.

- For the Hive `timestamp` data type, the Hive generated Parquet schema for the data type specifies that the data is stored as data type `int96`. The `gphdfs` protocol converts the `int96` data type to the Greenplum Database `bytea` data type.

### Writing a Parquet File

For writable external tables, you can add parameters after the file specified in the `location`. You add parameters with the http query string syntax that starts with `?` and `&` between field and value pairs.

<table>
<thead>
<tr>
<th>Option</th>
<th>Values</th>
<th>Readable/Writable</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td><code>URL_to_schema</code></td>
<td>Write only</td>
<td>None. If not specified, the <code>gphdfs</code> protocol creates a schema according to the external table definition.</td>
</tr>
<tr>
<td>pagesize</td>
<td>&gt; 1024 Bytes</td>
<td>Write only</td>
<td>1 MB</td>
</tr>
<tr>
<td>rowgroupsize</td>
<td>&gt; 1024 Bytes</td>
<td>Write only</td>
<td>8 MB</td>
</tr>
<tr>
<td>version</td>
<td>v1, v2</td>
<td>Write only</td>
<td>v1</td>
</tr>
<tr>
<td>codec</td>
<td>UNCOMPRESSED, GZIP, LZO, snappy</td>
<td>Write only</td>
<td>UNCOMPRESSED</td>
</tr>
<tr>
<td>dictionaryenable†</td>
<td>true, false</td>
<td>Write only</td>
<td>false</td>
</tr>
<tr>
<td>dictionarypagesize†</td>
<td>&gt; 1024 Bytes</td>
<td>Write only</td>
<td>512 KB</td>
</tr>
</tbody>
</table>

**Note:**

1. Creates an internal dictionary. Enabling a dictionary can improve Parquet file compression if text columns contain similar or duplicate data.

When writing a Parquet file, the `gphdfs` protocol can generate a Parquet schema based on the table definition.

- The table name is used as the Parquet `message` name.
- The column name is used as the Parquet `field` name.
When creating the Parquet schema from a Greenplum Database table definition, the schema is generated based on the column data type.

**Table 60: Schema Data Type Conversion when Writing a Parquet File**

<table>
<thead>
<tr>
<th>Greenplum Database Data Type</th>
<th>Parquet Schema Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>optional boolean</td>
</tr>
<tr>
<td>smallint</td>
<td>optional int32 with annotation int_16</td>
</tr>
<tr>
<td>int</td>
<td>optional int32</td>
</tr>
<tr>
<td>bigint</td>
<td>optional int64</td>
</tr>
<tr>
<td>real</td>
<td>optional float</td>
</tr>
<tr>
<td>double</td>
<td>optional double</td>
</tr>
<tr>
<td>numeric or decimal</td>
<td>binary with annotation decimal</td>
</tr>
<tr>
<td>bytea</td>
<td>optional binary</td>
</tr>
<tr>
<td>array column</td>
<td>repeated field - The data type is the same data type as the Greenplum Database the array. For example, array[int] is converted to repeated int</td>
</tr>
<tr>
<td>Others</td>
<td>binary with annotation utf8</td>
</tr>
</tbody>
</table>

**Note:** To support Null data, gphdfs protocol specifies the Parquet optional schema annotation when creating a Parquet schema.

A simple example of a Greenplum Database table definition and the Parquet schema generated by the gphdfs protocol.

An example external table definition for a Parquet file.

```sql
CREATE WRITABLE EXTERNAL TABLE films (  
    code char(5),  
    title varchar(40),  
    id integer,  
    date_prod date,  
    subtitle boolean  
) LOCATION ( 'gphdfs://my-films') FORMAT 'PARQUET' ;
```

This is the Parquet schema for the Parquet file my-films generated by the gphdfs protocol.

```python
message films {  
    optional byte_array code;  
    optional byte_array title (utf8);  
    optional int32 id;  
    optional binary date_prod (utf8);  
    optional boolean subtitle;  
}
```

**Limitations and Notes**

- For writable external tables, column definitions in Greenplum Database external table cannot specify NOT NULL to support automatically generating a Parquet schema. When the gphdfs protocol automatically generates a Parquet schema, the gphdfs protocol specifies the field attribute optional to support null in the Parquet schema. Repeated fields can be null in Parquet.
• The gphdfs protocol supports Parquet nested group structures only for readable external files. The nested structures are converted to an XML document.

• Greenplum Database does not have an unsigned int data type. Greenplum Database converts the Parquet unsigned int data type to the next largest Greenplum Database int type. For example, Parquet uint_8 is converted to Greenplum Database int (32 bit).

• Greenplum Database supports any UDT data type or UDT array data type. Greenplum Database attempts to convert the UDT to a sting. If the UDT cannot be converted to a sting, Greenplum Database returns an error.

• The definition of the Interval data type in Parquet is significantly different than the Interval definition in Greenplum Database and cannot be converted. The Parquet Interval data is formatted as bytea.

• The Date data type in Parquet is starts from 1970.1.1, while Date in Greenplum Database starts from 4173 BC. Greenplum Database cannot convert date data types because largest values are different. A similar situation occurs between Timestamp_millis in Parquet and Timestamp in Greenplum Database.

HDFS Readable and Writable External Table Examples

The following code defines a readable external table for an HDFS file named filename.txt on port 8081.

```sql
=# CREATE EXTERNAL TABLE ext_expenses (  
  name text,  
  date date,  
  amount float4,  
  category text,  
  desc1 text )  
LOCATION ('gphdfs://hdfshost-1:8081/data/filename.txt')  
FORMAT 'TEXT' (DELIMITER ',');
```

**Note:** Omit the port number when using the gpmr-1.0-gnet-1.0.0.1 connector.

The following code defines a set of readable external tables that have a custom format located in the same HDFS directory on port 8081.

```sql
=# CREATE EXTERNAL TABLE ext_expenses  
LOCATION ('gphdfs://hdfshost-1:8081/data/custdat*.dat')  
FORMAT 'custom' (formatter='gphdfs_import');
```

**Note:** Omit the port number when using the gpmr-1.0-gnet-1.0.0.1 connector.

The following code defines an HDFS directory for a writable external table on port 8081 with all compression options specified.

```sql
=# CREATE WRITABLE EXTERNAL TABLE ext_expenses  
LOCATION ('gphdfs://hdfshost-1:8081/data/?compress=true&compression_type=RECORD  
&codec=org.apache.hadoop.io.compress.DefaultCodec')  
FORMAT 'custom' (formatter='gphdfs_export');
```

**Note:** Omit the port number when using the gpmr-1.0-gnet-1.0.0.1 connector.

Because the previous code uses the default compression options for compression_type and codec, the following command is equivalent.

```sql
=# CREATE WRITABLE EXTERNAL TABLE ext_expenses  
LOCATION ('gphdfs://hdfshost-1:8081/data?compress=true')  
FORMAT 'custom' (formatter='gphdfs_export');
```
**Note:** Omit the port number when using the `gpmr-1.0-gnet-1.0.0.1` connector.

## Reading and Writing Custom-Formatted HDFS Data

Use MapReduce and the `CREATE EXTERNAL TABLE` command to read and write data with custom formats on HDFS.

To read custom-formatted data:

1. Author and run a MapReduce job that creates a copy of the data in a format accessible to Greenplum Database.
2. Use `CREATE EXTERNAL TABLE` to read the data into Greenplum Database.

See [Example 1 - Read Custom-Formatted Data from HDFS](#).

To write custom-formatted data:

1. Write the data.
2. Author and run a MapReduce program to convert the data to the custom format and place it on the Hadoop Distributed File System.

See [Example 2 - Write Custom-Formatted Data from Greenplum Database to HDFS](#).

MapReduce code is written in Java. Greenplum provides Java APIs for use in the MapReduce code. The Javadoc is available in the `$GPHOME/docs` directory. To view the Javadoc, expand the file `gnet-1.1-javadoc.tar` and open `index.html`. The Javadoc documents the following packages:

```java
com.emc.greenplum.gpdb.hadoop.io
com.emc.greenplum.gpdb.hadoop.mapred
com.emc.greenplum.gpdb.hadoop.mapreduce.lib.input
com.emc.greenplum.gpdb.hadoop.mapreduce.lib.output
```

The HDFS cross-connect packages contain the Java library, which contains the packages `GPDBWritable`, `GPDBInputFormat`, and `GPDBOutputFormat`. The Java packages are available in `$GPHOME/lib/hadoop`. Compile and run the MapReduce job with the cross-connect package. For example, compile and run the MapReduce job with `gphd-1.0-gnet-1.0.0.1.jar` if you use the Greenplum HD 1.0 distribution of Hadoop.

To make the Java library available to all Hadoop users, the Hadoop cluster administrator should place the corresponding `gphdfs` connector jar in the `$HADOOP_HOME/lib` directory and restart the job tracker. If this is not done, a Hadoop user can still use the `gphdfs` connector jar; but with the `distributed cache` technique.

### Example 1 - Read Custom-Formatted Data from HDFS

The sample code makes the following assumptions.

- The data is contained in HDFS directory `/demo/data/temp` and the name node is running on port 8081.
- This code writes the data in Greenplum Database format to `/demo/data/MRTest1` on HDFS.
- The data contains the following columns, in order.
  1. A long integer
  2. A Boolean
  3. A text string

#### Sample MapReduce Code

```java
import com.emc.greenplum.gpdb.hadoop.io.GPDBWritable;
import com.emc.greenplum.gpdb.hadoop.mapreduce.lib.input.GPDBInputFormat;
import com.emc.greenplum.gpdb.hadoop.mapreduce.lib.output.GPDBOutputFormat;
import java.io.*;
```
import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.output.*;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.util.*;

public class demoMR {

    /*
    * Helper routine to create our generic record. This section shows the
    * format of the data. Modify as necessary.
    */
    public static GPDBWritable generateGenericRecord() throws
        IOException {
        int[] colType = new int[3);
        colType[0] = GPDBWritable.BIGINT;
        colType[1] = GPDBWritable.BOOLEAN;
        colType[2] = GPDBWritable.VARCHAR;

        /*
        * This section passes the values of the data. Modify as necessary.
        */
        GPDBWritable gw = new GPDBWritable(colType);
        gw.setLong (0, (long)12345);
        gw.setBoolean(1, true);
        gw.setString (2, "abcdef");
        return gw;
    }

    /*
    * DEMO Map/Reduce class test1
    * -- Regardless of the input, this section dumps the generic record
    * into GPDBFormat/
    */
    public static class Map_test1
        extends Mapper<LongWritable, Text, LongWritable, GPDBWritable> {
        private LongWritable word = new LongWritable(1);

        public void map(LongWritable key, Text value, Context context) throws
            IOException {
            try {
                GPDBWritable gw = generateGenericRecord();
                context.write(word, gw);
            } catch (Exception e) {
                throw new IOException (e.getMessage());
            }
        }
    }

    Configuration conf = new Configuration(true);
    Job job = new Job(conf, "test1");
    job.setJarByClass(demoMR.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputKeyClass (LongWritable.class);
    job.setOutputValueClass (GPDBWritable.class);
    job.setOutputFormatClass(GPDBOutputFormat.class);
    job.setMapperClass(Map_test1.class);
    FileInputFormat.setInputPaths (job, new Path("/demo/data/tmp"));
    GPDBOutputFormat.setOutputPath(job, new Path("/demo/data/MRTest1"));
Run CREATE EXTERNAL TABLE

The Hadoop location corresponds to the output path in the MapReduce job.

```sql
=# CREATE EXTERNAL TABLE demodata
    LOCATION ('gphdfs://hdfshost-1:8081/demo/data/MRTest1')
    FORMAT 'custom' (formatter='gphdfs_import');
```

Note:
Omit the port number when using the gpmr-1.0-gnet-1.0.0.1 connector.

Example 2 - Write Custom-Formatted Data from Greenplum Database to HDFS

The sample code makes the following assumptions.

- The data in Greenplum Database format is located on the Hadoop Distributed File System on `/demo/data/writeFromGPDB_42` on port 8081.
- This code writes the data to `/demo/data/MRTest2` on port 8081.

1. Run a SQL command to create the writable table.

```sql
=# CREATE WRITABLE EXTERNAL TABLE demodata
    LOCATION ('gphdfs://hdfshost-1:8081/demo/data/MRTest2')
    FORMAT 'custom' (formatter='gphdfs_export');
```

2. Author and run code for a MapReduce job. Use the same import statements shown in Example 1 - Read Custom-Formatted Data from HDFS.

   Note: Omit the port number when using the gpmr-1.0-gnet-1.0.0.1 connector.

Sample MapReduce Code

```java
import java.io.IOException;
import java.util.Map;
import java.util.Context;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.io.ByteArrayWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.GPDBInputFormat;
import org.apache.hadoop.mapreduce.lib.output.GPDBOutputFormat;

public class MapReduceTool {

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration(true);
        Job job = new Job(conf, "test2");
        job.setJarByClass(MapReduceTool.class);
        job.setInputFormatClass(GPDBInputFormat.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(NullWritable.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.setMapperClass(Map_test2.class);
        job.setInputPaths(new Path("/demo/data/writeFromGPDB_42"));
        job.setOutputPath(new Path("/demo/data/MRTest2"));
    }

    public static class Map_test2 extends Mapper<LongWritable, ByteArrayWritable, Text, NullWritable> {
        public void map(LongWritable key, ByteArrayWritable value, Context context)
        throws IOException, InterruptedException {
            context.write(new Text(value.toString().getBytes()), NullWritable.get());
        }
    }

    public static class Reduce_test2 extends Reducer<Text, NullWritable, Text, NullWritable> {
        public void reduce(Text key, Iterable<NullWritable> values, Context context)
        throws IOException, InterruptedException {
            String result = "Result: " + key.toString();
            context.write(new Text(result), NullWritable.get());
        }
    }
}
```
Using Amazon EMR with Greenplum Database installed on AWS

Amazon Elastic MapReduce (EMR) is a managed cluster platform that can run big data frameworks, such as Apache Hadoop and Apache Spark, on Amazon Web Services (AWS) to process and analyze data. For a Greenplum Database system that is installed on Amazon Web Services (AWS), you can define Greenplum Database external tables that use the gphdfs protocol to access files on an Amazon EMR instance HDFS.

In addition to the steps described in One-time gphdfs Protocol Installation, you must also ensure Greenplum Database can access the EMR instance. If your Greenplum Database system is running on an Amazon Elastic Compute Cloud (EC2) instance, you configure the Greenplum Database system and the EMR security group.

For information about Amazon EMR, see https://aws.amazon.com/emr/. For information about Amazon EC2, see https://aws.amazon.com/ec2/

Configuring Greenplum Database and Amazon EMR

These steps describe how to set up Greenplum Database system and an Amazon EMR instance to support Greenplum Database external tables:

1. Ensure that the appropriate Java (including JDK) and Hadoop environments are correctly installed on all Greenplum Database segment hosts.
   
   For example, Amazon EMR Release 4.0.0 includes Apache Hadoop 2.6.0. This Amazon page describes Amazon EMR Release 4.0.0.
   
   For information about Hadoop versions used by EMR and Greenplum Database, see Table 61: EMR Hadoop Configuration Information.

2. Ensure the environment variables and Greenplum Database server configuration parameters are set:
   
   - System environment variables:
     
     - HADOOP_HOME
     - JAVA_HOME
   
   - Greenplum Database server configuration parameters:
     
     - gp_hadoop_target_version
     - gp_hadoop_home

3. Configure communication between Greenplum Database and the EMR instance Hadoop master.
   
   For example, open port 8020 in the AWS security group.

4. Configure for communication between Greenplum Database and EMR instance Hadoop data nodes.
   
   Open a TCP/IP port for so that Greenplum Database segments hosts can communicate with EMR instance Hadoop data nodes.
   
   For example, open port 50010 in the AWS security manager.

This table lists EMR and Hadoop version information that can be used to configure Greenplum Database.

Table 61: EMR Hadoop Configuration Information

<table>
<thead>
<tr>
<th>EMR Version</th>
<th>EMR Apache Hadoop Version</th>
<th>EMR Hadoop Master Port</th>
<th>gp_hadoop_target_version</th>
<th>Hadoop Version on Greenplum Database Segment Hosts</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0</td>
<td>2.6</td>
<td>8020</td>
<td>hadoop2</td>
<td>Apache Hadoop 2.x</td>
</tr>
</tbody>
</table>
Using the Greenplum Parallel File Server (gpfdist)

The gpfdist protocol provides the best performance and is the easiest to set up. gpfdist ensures optimum use of all segments in your Greenplum Database system for external table reads.

This topic describes the setup and management tasks for using gpfdist with external tables.

- About gpfdist Setup and Performance
- Controlling Segment Parallelism
- Installing gpfdist
- Starting and Stopping gpfdist
- Troubleshooting gpfdist

About gpfdist Setup and Performance

Consider the following scenarios for optimizing your ETL network performance.

- Allow network traffic to use all ETL host Network Interface Cards (NICs) simultaneously. Run one instance of gpfdist on the ETL host, then declare the host name of each NIC in the LOCATION clause of your external table definition (see Examples for Creating External Tables).

<table>
<thead>
<tr>
<th>EMR Version</th>
<th>EMR Apache Hadoop Version</th>
<th>EMR Hadoop Master Port</th>
<th>gp_hadoop_target_version</th>
<th>Hadoop Version on Greenplum Database Segment Hosts</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.9</td>
<td>2.4</td>
<td>9000</td>
<td>hadoop2</td>
<td>Apache Hadoop 2.x</td>
</tr>
<tr>
<td>3.8</td>
<td>2.4</td>
<td>9000</td>
<td>hadoop2</td>
<td>Apache Hadoop 2.x</td>
</tr>
<tr>
<td>3.3</td>
<td>2.4</td>
<td>9000</td>
<td>hadoop2</td>
<td>Apache Hadoop 2.x</td>
</tr>
</tbody>
</table>

Figure 30: External Table Using Single gpfdist Instance with Multiple NICs
• Divide external table data equally among multiple `gpfdist` instances on the ETL host. For example, on an ETL system with two NICs, run two `gpfdist` instances (one on each NIC) to optimize data load performance and divide the external table data files evenly between the two `gpfdists`.

Figure 31: External Tables Using Multiple `gpfdist` Instances with Multiple NICs

**Note:** Use pipes (|) to separate formatted text when you submit files to `gpfdist`. Greenplum Database encloses comma-separated text strings in single or double quotes. `gpfdist` has to remove the quotes to parse the strings. Using pipes to separate formatted text avoids the extra step and improves performance.

**Controlling Segment Parallelism**

The `gp_external_max_segs` server configuration parameter controls the number of segment instances that can access a single `gpfdist` instance simultaneously. 64 is the default. You can set the number of segments such that some segments process external data files and some perform other database processing. Set this parameter in the `postgresql.conf` file of your master instance.

**Installing `gpfdist`**

`gpfdist` is installed in `$GPHOME/bin` of your Greenplum Database master host installation. Run `gpfdist` on a machine other than the Greenplum Database master or standby master, such as on a machine devoted to ETL processing. Running `gpfdist` on the master or standby master can have a performance impact on query execution. To install `gpfdist` on your ETL server, get it from the Greenplum Load Tools package and follow its installation instructions.

**Starting and Stopping `gpfdist`**

You can start `gpfdist` in your current directory location or in any directory that you specify. The default port is 8080.
From your current directory, type:

```bash
gpfdist &
```

From a different directory, specify the directory from which to serve files, and optionally, the HTTP port to run on.

To start `gpfdist` in the background and log output messages and errors to a log file:

```bash
$ gpfdist -d /var/load_files -p 8081 -l /home/gpadmin/log &
```

For multiple `gpfdist` instances on the same ETL host (see Figure 30: External Table Using Single `gpfdist` Instance with Multiple NICs), use a different base directory and port for each instance. For example:

```bash
$ gpfdist -d /var/load_files1 -p 8081 -l /home/gpadmin/log1 &
$ gpfdist -d /var/load_files2 -p 8082 -l /home/gpadmin/log2 &
```

To stop `gpfdist` when it is running in the background:

First find its process id:

```bash
$ ps -ef | grep gpfdist
```

Then kill the process, for example (where 3456 is the process ID in this example):

```bash
$ kill 3456
```

**Troubleshooting gpfdist**

The segments access `gpfdist` at runtime. Ensure that the Greenplum segment hosts have network access to `gpfdist`. `gpfdist` is a web server: test connectivity by running the following command from each host in the Greenplum array (segments and master):

```bash
$ wget http://gpfdist_hostname:port/filename
```

The `CREATE EXTERNAL TABLE` definition must have the correct host name, port, and file names for `gpfdist`. Specify file names and paths relative to the directory from which `gpfdist` serves files (the directory path specified when `gpfdist` started). See Examples for Creating External Tables.

If you start `gpfdist` on your system and IPv6 networking is disabled, `gpfdist` displays this warning message when testing for an IPv6 port.

```
[WRN gpfdist.c:2050] Creating the socket failed
```

If the corresponding IPv4 port is available, `gpfdist` uses that port and the warning for IPv6 port can be ignored. To see information about the ports that `gpfdist` tests, use the `-V` option.

For information about IPv6 and IPv4 networking, see your operating system documentation.

When reading or writing data with the `gpfdist` or `gfdists` protocol, the `gpfdist` utility rejects HTTP requests that do not include `X-GP-PROTO` in the request header. If `X-GP-PROTO` is not detected in the header request, `gpfist` returns a 400 error in the status line of the HTTP response header: 400 invalid request (no gp-proto).

Greenplum Database includes `X-GP-PROTO` in the HTTP request header to indicate that the request is from Greenplum Database.
Loading and Unloading Data

The topics in this section describe methods for loading and writing data into and out of a Greenplum Database, and how to format data files.

Greenplum Database supports high-performance parallel data loading and unloading, and for smaller amounts of data, single file, non-parallel data import and export.

Greenplum Database can read from and write to several types of external data sources, including text files, Hadoop file systems, and web servers.

- The COPY SQL command transfers data between an external text file on the master host and a Greenplum Database table.
- External tables allow you to query data outside of the database directly and in parallel using SQL commands such as SELECT, JOIN, or SORT EXTERNAL TABLE DATA, and you can create views for external tables. External tables are often used to load external data into a regular database table using a command such as CREATE TABLE table AS SELECT * FROM ext_table.
- External web tables provide access to dynamic data. They can be backed with data from URLs accessed using the HTTP protocol or by the output of an OS script running on one or more segments.
- The gpfdist utility is the Greenplum Database parallel file distribution program. It is an HTTP server that is used with external tables to allow Greenplum Database segments to load external data in parallel, from multiple file systems. You can run multiple instances of gpfdist on different hosts and network interfaces and access them in parallel.
- The gpreload utility automates the steps of a load task using a YAML-formatted control file.
- The GemFire-Greenplum Connector allows the transfer of data between a Pivotal GemFire region and a Greenplum Database table. Pivotal Gemfire is an in-memory data management system that provides reliable asynchronous event notifications and guaranteed message delivery. For information about using GemFire-Greenplum Connector, see http://ggc.docs.pivotal.io/. For information about Pivotal Gemfire, see http://gemfire.docs.pivotal.io/.
- The Greenplum-Spark Connector provides high speed, parallel data transfer between Pivotal Greenplum Database and Apache Spark. For information about using the Greenplum-Spark Connector, refer to the documentation at https://greenplum-spark.docs.pivotal.io/.

The method you choose to load data depends on the characteristics of the source data—its location, size, format, and any transformations required.

In the simplest case, the COPY SQL command loads data into a table from a text file that is accessible to the Greenplum Database master instance. This requires no setup and provides good performance for smaller amounts of data. With the COPY command, the data copied into or out of the database passes between a single file on the master host and the database. This limits the total size of the dataset to the capacity of the file system where the external file resides and limits the data transfer to a single file write stream.

More efficient data loading options for large datasets take advantage of the Greenplum Database MPP architecture, using the Greenplum Database segments to load data in parallel. These methods allow data to load simultaneously from multiple file systems, through multiple NICs, on multiple hosts, achieving very high data transfer rates. External tables allow you to access external files from within the database as if they are regular database tables. When used with gpfdist, the Greenplum Database parallel file distribution program, external tables provide full parallelism by using the resources of all Greenplum Database segments to load or unload data.

Greenplum Database leverages the parallel architecture of the Hadoop Distributed File System to access files on that system.
**Loading Data Using an External Table**

Use SQL commands such as `INSERT` and `SELECT` to query a readable external table, the same way that you query a regular database table. For example, to load travel expense data from an external table, `ext_expenses`, into a database table, `expenses_travel`:

```sql
=# INSERT INTO expenses_travel
    SELECT * FROM ext_expenses WHERE category='travel';
```

To load all data into a new database table:

```sql
=# CREATE TABLE expenses AS SELECT * FROM ext_expenses;
```

**Loading and Writing Non-HDFS Custom Data**

Greenplum supports `TEXT` and `CSV` formats for importing and exporting data. You can load and write the data in other formats by defining and using a custom format or custom protocol.

For information about importing custom data from HDFS, see Reading and Writing Custom-Formatted HDFS Data.

**Using a Custom Format**

You specify a custom data format in the `FORMAT` clause of `CREATE EXTERNAL TABLE`.

```sql
FORMAT 'CUSTOM' (formatter=format_function, key1=val1,...keyn=valn)
```

Where the `'CUSTOM'` keyword indicates that the data has a custom format and `formatter` specifies the function to use to format the data, followed by comma-separated parameters to the formatter function.

Greenplum Database provides functions for formatting fixed-width data, but you must author the formatter functions for variable-width data. The steps are as follows.

1. Author and compile input and output functions as a shared library.
2. Specify the shared library function with `CREATE FUNCTION` in Greenplum Database.
3. Use the `formatter` parameter of `CREATE EXTERNAL TABLE`'s `FORMAT` clause to call the function.

**Importing and Exporting Fixed Width Data**

Specify custom formats for fixed-width data with the Greenplum Database functions `fixedwith_in` and `fixedwidth_out`. These functions already exist in the file `$GPHOME/share/postgresql/cdb_external_extensions.sql`. The following example declares a custom format, then calls the `fixedwidth_in` function to format the data.

```sql
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');
```

The following options specify how to import fixed width data.

- Read all the data.

To load all the fields on a line of fixed with data, you must load them in their physical order. You must specify the field length, but cannot specify a starting and ending position. The fields names in the fixed width arguments must match the order in the field list at the beginning of the `CREATE TABLE` command.
• Set options for blank and null characters.

  Trailing blanks are trimmed by default. To keep trailing blanks, use the `preserve_blanks=on` option. You can reset the trailing blanks option to the default with the `preserve_blanks=off` option.

  Use the `null='null_string_value'` option to specify a value for null characters.
  • If you specify `preserve_blanks=on`, you must also define a value for null characters.
  • If you specify `preserve_blanks=off`, null is not defined, and the field contains only blanks, Greenplum writes a null to the table. If null is defined, Greenplum writes an empty string to the table.

  Use the `line_delim='line_ending'` parameter to specify the line ending character. The following examples cover most cases. The `E` specifies an escape string constant.

  ```sql
  line_delim=E'\n'
  line_delim=E'\r'
  line_delim=E'\r\n'
  line_delim='abc'
  ```

**Examples: Read Fixed-Width Data**

The following examples show how to read fixed-width data.

**Example 1 – Loading a table with all fields defined**

```sql
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);
```

**Example 2 – Loading a table with PRESERVED_BLANKS on**

```sql
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');
```

**Example 3 – Loading data with no line delimiter**

```sql
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='?@')
```

**Example 4 – Create a writable external table with a \n line delimiter**

```sql
CREATE WRITABLE EXTERNAL TABLE students_out (  
  name varchar(20), address varchar(30), age int)  
LOCATION ('gpfdist://<host>:<portNum>/file/path/students_out.txt')  
FORMAT 'CUSTOM' (formatter=fixedwidth_out,  
  name=20, address=30, age=4, line_delim=E'\r\n');
```
Using a Custom Protocol

Greenplum provides protocols such as gpfdist, http, and file for accessing data over a network, or you can author a custom protocol. You can use the standard data formats, TEXT and CSV, or a custom data format with custom protocols.

You can create a custom protocol whenever the available built-in protocols do not suffice for a particular need. For example, if you need to connect Greenplum Database in parallel to another system directly, and stream data from one to the other without the need to materialize the system data on disk or use an intermediate process such as gpfdist.

1. Author the send, receive, and (optionally) validator functions in C, with a predefined API. These functions are compiled and registered with the Greenplum Database. For an example custom protocol, see Example Custom Data Access Protocol.

2. After writing and compiling the read and write functions into a shared object (.so), declare a database function that points to the .so file and function names.

The following examples use the compiled import and export code.

```
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;

CREATE OR REPLACE FUNCTION myvalidate() RETURNS void
    AS '$libdir/gpextprotocol.so', 'myprot_validate'
    LANGUAGE C STABLE;
```

3. Create a protocol that accesses these functions. Validatorfunc is optional.

```
CREATE TRUSTED PROTOCOL myprot(
    writefunc='mywrite',
    readfunc='myread',
    validatorfunc='myvalidate');
```

4. Grant access to any other users, as necessary.

```
GRANT ALL ON PROTOCOL myprot TO otheruser
```

5. Use the protocol in readable or writable external tables.

```
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';
```

Declare custom protocols with the SQL command CREATE TRUSTED PROTOCOL, then use the GRANT command to grant access to your users. For example:

- Allow a user to create a readable external table with a trusted protocol

```
GRANT SELECT ON PROTOCOL <protocol name> TO <user name>
```
• Allow a user to create a writable external table with a trusted protocol

```
GRANT INSERT ON PROTOCOL <protocol name> TO <user name>
```

• Allow a user to create readable and writable external tables with a trusted protocol

```
GRANT ALL ON PROTOCOL <protocol name> TO <user name>
```

## Handling Load Errors

Readable external tables are most commonly used to select data to load into regular database tables. You use the `CREATE TABLE AS SELECT` or `INSERT INTO` commands to query the external table data. By default, if the data contains an error, the entire command fails and the data is not loaded into the target database table.

The `SEGMENT REJECT LIMIT` clause allows you to isolate format errors in external table data and to continue loading correctly formatted rows. Use `SEGMENT REJECT LIMIT` to set an error threshold, specifying the reject limit as number of rows (the default) or as a percent of total rows (1-100).

The entire external table operation is aborted, and no rows are processed, if the number of error rows reaches the `SEGMENT REJECT LIMIT`. The limit of error rows is per-segment, not per entire operation. The operation processes all good rows, and it discards and optionally logs formatting errors for erroneous rows, if the number of error rows does not reach the `SEGMENT REJECT LIMIT`.

The `LOG ERRORS` clause allows you to keep error rows for further examination. For information about the `LOG ERRORS` clause, see the `CREATE EXTERNAL TABLE` command in the [*Greenplum Database Reference Guide*](#).

When you set `SEGMENT REJECT LIMIT`, Greenplum scans the external data in single row error isolation mode. Single row error isolation mode applies to external data rows with format errors such as extra or missing attributes, attributes of a wrong data type, or invalid client encoding sequences. Greenplum does not check constraint errors, but you can filter constraint errors by limiting the `SELECT` from an external table at runtime. For example, to eliminate duplicate key errors:

```
=# INSERT INTO table_with_pkeys
  SELECT DISTINCT * FROM external_table;
```

**Note:** When loading data with the `COPY` command or an external table, the value of the server configuration parameter `gp_initial_bad_row_limit` limits the initial number of rows that are processed that are not formatted properly. The default is to stop processing if the first 1000 rows contain formatting errors. See the [*Greenplum Database Reference Guide*](#) for information about the parameter.

## Define an External Table with Single Row Error Isolation

The following example logs errors internally in Greenplum Database and sets an error threshold of 10 errors.

```
=# 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;
```
Use the built-in SQL function `gp_read_error_log('external_table')` to read the error log data. This example command displays the log errors for `ext_expenses`:

```sql
SELECT gp_read_error_log('ext_expenses');
```

For information about the format of the error log, see Viewing Bad Rows in the Error Log.

The built-in SQL function `gp_truncate_error_log('external_table')` deletes the error data. This example deletes the error log data created from the previous external table example:

```sql
SELECT gp_truncate_error_log('ext_expenses');
```

**Capture Row Formatting Errors and Declare a Reject Limit**

The following SQL fragment captures formatting errors internally in Greenplum Database and declares a reject limit of 10 rows.

```sql
LOG ERRORS SEGMENT REJECT LIMIT 10 ROWS
```

Use the built-in SQL function `gp_read_error_log()` to read the error log data. For information about viewing log errors, see Viewing Bad Rows in the Error Log.

**Viewing Bad Rows in the Error Log**

If you use single row error isolation (see Define an External Table with Single Row Error Isolation or Running COPY in Single Row Error Isolation Mode), any rows with formatting errors are logged internally by Greenplum Database.

Greenplum Database captures the following error information in a table format:

**Table 62: Error Log Format**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmdtime</td>
<td>timestamptz</td>
<td>Timestamp when the error occurred.</td>
</tr>
<tr>
<td>relname</td>
<td>text</td>
<td>The name of the external table or the target table of a COPY command.</td>
</tr>
<tr>
<td>filename</td>
<td>text</td>
<td>The name of the load file that contains the error.</td>
</tr>
<tr>
<td>linenum</td>
<td>int</td>
<td>If COPY was used, the line number in the load file where the error occurred. For external tables using file:// protocol or gpfdist:// protocol and CSV format, the file name and line number is logged.</td>
</tr>
<tr>
<td>bytenum</td>
<td>int</td>
<td>For external tables with the gpfdist:// protocol and data in TEXT format: the byte offset in the load file where the error occurred. gpfdist parses TEXT files in blocks, so logging a line number is not possible. CSV files are parsed a line at a time so line number tracking is possible for CSV files.</td>
</tr>
<tr>
<td>errmsg</td>
<td>text</td>
<td>The error message text.</td>
</tr>
<tr>
<td>rawdata</td>
<td>text</td>
<td>The raw data of the rejected row.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>description</td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rawbytes</td>
<td>bytea</td>
<td>In cases where there is a database encoding error (the client encoding used cannot be converted to a server-side encoding), it is not possible to log the encoding error as rawdata. Instead the raw bytes are stored and you will see the octal code for any non seven bit ASCII characters.</td>
</tr>
</tbody>
</table>

You can use the Greenplum Database built-in SQL function `gp_read_error_log()` to display formatting errors that are logged internally. For example, this command displays the error log information for the table `ext_expenses`:

```
SELECT gp_read_error_log('ext_expenses');
```

For information about managing formatting errors that are logged internally, see the command `COPY` or `CREATE EXTERNAL TABLE` in the Greenplum Database Reference Guide.

### Identifying Invalid CSV Files in Error Table Data

If a CSV file contains invalid formatting, the rawdata field in the error table can contain several combined rows. For example, if a closing quote for a specific field is missing, all the following newlines are treated as embedded newlines. When this happens, Greenplum stops parsing a row when it reaches 64K, puts that 64K of data into the error table as a single row, resets the quote flag, and continues. If this happens three times during load processing, the load file is considered invalid and the entire load fails with the message "rejected N or more rows". See Escaping in CSV Formatted Files for more information on the correct use of quotes in CSV files.

### Moving Data between Tables

You can use `CREATE TABLE AS` or `INSERT...SELECT` to load external and external web table data into another (non-external) database table, and the data will be loaded in parallel according to the external or external web table definition.

If an external table file or external web table data source has an error, one of the following will happen, depending on the isolation mode used:

- **Tables without error isolation mode**: any operation that reads from that table fails. Loading from external and external web tables without error isolation mode is an all or nothing operation.
- **Tables with error isolation mode**: the entire file will be loaded, except for the problematic rows (subject to the configured REJECT_LIMIT)

### Loading Data with `gpload`

The Greenplum `gpload` utility loads data using readable external tables and the Greenplum parallel file server (`gpfdist` or `gpfdists`). It handles parallel file-based external table setup and allows users to configure their data format, external table definition, and `gpfdist` or `gpfdists` setup in a single configuration file.

#### To use `gpload`

1. Ensure that your environment is set up to run `gpload`. Some dependent files from your Greenplum Database installation are required, such as `gpfdist` and Python, as well as network access to the Greenplum segment hosts.

   See the Greenplum Database Reference Guide for details.

2. Create your load control file. This is a YAML-formatted file that specifies the Greenplum Database connection information, `gpfdist` configuration information, external table options, and data format.
See the Greenplum Database Reference Guide for details.

For example:

```---
VERSION: 1.0.0.1
DATABASE: ops
USER: gpadmin
HOST: mdw-1
PORT: 5432
GPLOAD:
  INPUT:
    - SOURCE:
      LOCAL_HOSTNAME:
        - etl1-1
        - etl1-2
        - etl1-3
        - etl1-4
      PORT: 8081
    FILE:
      - /var/load/data/*
  COLUMNS:
    - name: text
    - amount: float4
    - category: text
    - desc: text
    - date: date
  FORMAT: text
  DELIMITER: '|'
  ERROR_LIMIT: 25
  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. Run `gpload`, passing in the load control file. For example:

```
gpload -f my_load.yml
```

**Loading Data with COPY**

`COPY FROM` copies data from a file or standard input into a table and appends the data to the table contents. `COPY` is non-parallel: data is loaded in a single process using the Greenplum master instance. Using `COPY` is only recommended for very small data files.

The `COPY` source file must be accessible to the master host. Specify the `COPY` source file name relative to the master host location.

Greenplum copies data from `STDIN` or `STDOUT` using the connection between the client and the master server.

**Running COPY in Single Row Error Isolation Mode**

By default, `COPY` stops an operation at the first error: if the data contains an error, the operation fails and no data loads. If you run `COPY FROM` in *single row error isolation mode*, Greenplum skips rows that contain format errors and loads properly formatted rows. Single row error isolation mode applies only to rows in the
input file that contain format errors. If the data contains a constraint error such as violation of a \texttt{NOT NULL, CHECK}, or \texttt{UNIQUE} constraint, the operation fails and no data loads.

Specifying \texttt{SEGMENT REJECT LIMIT} runs the \texttt{COPY} operation in single row error isolation mode. Specify the acceptable number of error rows on each segment, after which the entire \texttt{COPY FROM} operation fails and no rows load. The error row count is for each Greenplum Database segment, not for the entire load operation.

If the \texttt{COPY} operation does not reach the error limit, Greenplum loads all correctly-formatted rows and discards the error rows. Use the \texttt{LOG ERRORS} clause to capture data formatting errors internally in Greenplum Database. For example:

\begin{verbatim}
=> COPY country FROM '/data/gpdb/country_data'
   WITH DELIMITER '|' LOG ERRORS
   SEGMENT REJECT LIMIT 10 ROWS;
\end{verbatim}

See \url{Viewing Bad Rows in the Error Log} for information about investigating error rows.

\section*{Optimizing Data Load and Query Performance}

Use the following tips to help optimize your data load and subsequent query performance.

\begin{itemize}
\item Drop indexes before loading data into existing tables.
\end{itemize}

Creating an index on pre-existing data is faster than updating it incrementally as each row is loaded. You can temporarily increase the \texttt{maintenance_work_mem} server configuration parameter to help speed up \texttt{CREATE INDEX} commands, though load performance is affected. Drop and recreate indexes only when there are no active users on the system.

\begin{itemize}
\item Create indexes last when loading data into new tables. Create the table, load the data, and create any required indexes.
\item Run \texttt{ANALYZE} after loading data. If you significantly altered the data in a table, run \texttt{ANALYZE} or \texttt{VACUUM ANALYZE} to update table statistics for the query optimizer. Current statistics ensure that the optimizer makes the best decisions during query planning and avoids poor performance due to inaccurate or nonexistent statistics.
\item Run \texttt{VACUUM} after load errors. If the load operation does not run in single row error isolation mode, the operation stops at the first error. The target table contains the rows loaded before the error occurred. You cannot access these rows, but they occupy disk space. Use the \texttt{VACUUM} command to recover the wasted space.
\end{itemize}

\section*{Unloading Data from Greenplum Database}

A writable external table allows you to select rows from other database tables and output the rows to files, named pipes, to applications, or as output targets for Greenplum parallel MapReduce calculations. You can define file-based and web-based writable external tables.

This topic describes how to unload data from Greenplum Database using parallel unload (writable external tables) and non-parallel unload (\texttt{COPY}).

\section*{Defining a File-Based Writable External Table}

Writable external tables that output data to files use the Greenplum parallel file server program, \texttt{gpfdist}, or the Hadoop Distributed File System interface, \texttt{gphdfs}.

Use the \texttt{CREATE WRITABLE EXTERNAL TABLE} command to define the external table and specify the location and format of the output files. See \url{Using the Greenplum Parallel File Server (gpfdist)} for instructions on setting up \texttt{gpfdist} for use with an external table and \url{Accessing HDFS Data with gphdfs} for instructions on setting up \texttt{gphdfs} for use with an external table.
With a writable external table using the `gpfdist` protocol, the Greenplum segments send their data to `gpfdist`, which writes the data to the named file. `gpfdist` must run on a host that the Greenplum segments can access over the network. `gpfdist` points to a file location on the output host and writes data received from the Greenplum segments to the file. To divide the output data among multiple files, list multiple `gpfdist` URIs in your writable external table definition.

A writable external web table sends data to an application as a stream of data. For example, unload data from Greenplum Database and send it to an application that connects to another database or ETL tool to load the data elsewhere. Writable external web tables use the `EXECUTE` clause to specify a shell command, script, or application to run on the segment hosts and accept an input stream of data. See Defining a Command-Based Writable External Web Table for more information about using `EXECUTE` commands in a writable external table definition.

You can optionally declare a distribution policy for your writable external tables. By default, writable external tables use a random distribution policy. If the source table you are exporting data from has a hash distribution policy, defining the same distribution key column(s) for the writable external table improves unload performance by eliminating the requirement to move rows over the interconnect. If you unload data from a particular table, you can use the `LIKE` clause to copy the column definitions and distribution policy from the source table.

**Example 1—Greenplum file server (gpfdist)**

```sql
=# 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);
```

**Example 2—Hadoop file server (gphdfs)**

```sql
=# CREATE WRITABLE EXTERNAL TABLE unload_expenses
   ( LIKE expenses )
   LOCATION ('gphdfs://hdfslhost-1:8081/path')
   FORMAT 'TEXT' (DELIMITER ',')
   DISTRIBUTED BY (exp_id);
```

You can only specify a directory for a writable external table with the `gphdfs` protocol. (You can only specify one file for a readable external table with the `gphdfs` protocol)

**Note:** The default port number is 9000.

**Defining a Command-Based Writable External Web Table**

You can define writable external web tables to send output rows to an application or script. The application must accept an input stream, reside in the same location on all of the Greenplum segment hosts, and be executable by the `gpadmin` user. All segments in the Greenplum system run the application or script, whether or not a segment has output rows to process.

Use `CREATE WRITABLE EXTERNAL WEB TABLE` to define the external table and specify the application or script to run on the segment hosts. Commands execute from within the database and cannot access environment variables (such as `$PATH`). Set environment variables in the `EXECUTE` clause of your writable external table definition. For example:

```sql
=# CREATE WRITABLE EXTERNAL WEB TABLE output (output text)
   EXECUTE 'export PATH=$PATH:/home/gpadmin
             /programs;
             myprogram.sh'
   FORMAT 'TEXT'
```
The following Greenplum Database variables are available for use in OS commands executed by a web or writable external table. Set these variables as environment variables in the shell that executes the command(s). They can be used to identify a set of requests made by an external table statement across the Greenplum Database array of hosts and segment instances.

**Table 63: External Table EXECUTE Variables**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$GP_CID</td>
<td>Command count of the transaction executing the external table statement.</td>
</tr>
<tr>
<td>$GP_DATABASE</td>
<td>The database in which the external table definition resides.</td>
</tr>
<tr>
<td>$GP_DATE</td>
<td>The date on which the external table command ran.</td>
</tr>
<tr>
<td>$GP_MASTER_HOST</td>
<td>The host name of the Greenplum master host from which the external</td>
</tr>
<tr>
<td></td>
<td>table statement was dispatched.</td>
</tr>
<tr>
<td>$GP_MASTER_PORT</td>
<td>The port number of the Greenplum master instance from which the external</td>
</tr>
<tr>
<td></td>
<td>table statement was dispatched.</td>
</tr>
<tr>
<td>$GP_SEG_DATADIR</td>
<td>The location of the data directory of the segment instance executing</td>
</tr>
<tr>
<td></td>
<td>the external table command.</td>
</tr>
<tr>
<td>$GP_SEG_PG_CONF</td>
<td>The location of the <code>postgresql.conf</code> file of the segment instance</td>
</tr>
<tr>
<td></td>
<td>executing the external table command.</td>
</tr>
<tr>
<td>$GP_SEG_PORT</td>
<td>The port number of the segment instance executing the external table command.</td>
</tr>
<tr>
<td>$GP_SEGMENT_COUNT</td>
<td>The total number of primary segment instances in the Greenplum</td>
</tr>
<tr>
<td></td>
<td>Database system.</td>
</tr>
<tr>
<td>$GP_SEGMENT_ID</td>
<td>The ID number of the segment instance executing the external table command</td>
</tr>
<tr>
<td></td>
<td>(same as <code>dbid</code> in <code>gp_segment_configuration</code>).</td>
</tr>
<tr>
<td>$GP_SESSION_ID</td>
<td>The database session identifier number associated with the external</td>
</tr>
<tr>
<td></td>
<td>table statement.</td>
</tr>
<tr>
<td>$GP_SN</td>
<td>Serial number of the external table scan node in the query plan of the</td>
</tr>
<tr>
<td></td>
<td>external table statement.</td>
</tr>
<tr>
<td>$GP_TIME</td>
<td>The time the external table command was executed.</td>
</tr>
<tr>
<td>$GP_USER</td>
<td>The database user executing the external table statement.</td>
</tr>
<tr>
<td>$GP_XID</td>
<td>The transaction ID of the external table statement.</td>
</tr>
</tbody>
</table>

**Disabling EXECUTE for Web or Writable External Tables**

There is a security risk associated with allowing external tables to execute OS commands or scripts. To disable the use of EXECUTE in web and writable external table definitions, set the `gp_external_enable_exec` server configuration parameter to `off` in your master `postgresql.conf` file:

```
gp_external_enable_exec = off
```
Unloading Data Using a Writable External Table

Writable external tables allow only INSERT operations. You must grant INSERT permission on a table to enable access to users who are not the table owner or a superuser. For example:

```
GRANT INSERT ON writable_ext_table TO admin;
```

To unload data using a writable external table, select the data from the source table(s) and insert it into the writable external table. The resulting rows are output to the writable external table. For example:

```
INSERT INTO writable_ext_table SELECT * FROM regular_table;
```

Unloading Data Using COPY

COPY TO copies data from a table to a file (or standard input) on the Greenplum master host using a single process on the Greenplum master instance. Use COPY to output a table's entire contents, or filter the output using a SELECT statement. For example:

```
COPY (SELECT * FROM country WHERE country_name LIKE 'A%')
TO '/home/gpadmin/a_list_countries.out';
```

Transforming XML Data

The Greenplum Database data loader gpfdist provides transformation features to load XML data into a table and to write data from the Greenplum Database to XML files. The following diagram shows gpfdist performing an XML transform.

![Diagram of XML Transformations](image)

Figure 32: External Tables using XML Transformations

Use these steps to load or extract XML data. Keep in mind that the first three steps comprise most of the development effort. The last two steps are straightforward and repeatable, suitable for production.
**Determine the Transformation Schema**

To prepare for the transformation project:

1. Determine the goal of the project, such as indexing data, analyzing data, combining data, and so on.
2. Examine the XML file and note the file structure and element names.
3. Choose the elements to import and decide if any other limits are appropriate.

For example, the following XML file, `prices.xml`, is a simple, short file that contains price records. Each price record contains two fields: an item number and a price.

```xml
<?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>
```

The goal is to import all the data into a Greenplum Database table with an integer `itemnumber` column and a decimal `price` column.

**Write a Transform**

The transform specifies what to extract from the data. You can use any authoring environment and language appropriate for your project. For XML transformations choose a technology such as XSLT, Joost (STX), Java, Python, or Perl, based on the goals and scope of the project.

In the price example, the next step is to transform the XML data into a simple two-column delimited format.

```
708421|19.99
708466|59.25
711121|24.99
```

The following STX transform, called `input_transform.stx`, completes the data transformation.

```xml
<?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:template match="itemnumber">
      <stx:assign name="itemnumber" select="."/>
    </stx:template>
    <stx:template match="price">
      <stx:assign name="price" select="."/>
    </stx:template>
</stx:transform>
```
This STX transform declares two temporary variables, `itemnumber` and `price`, and the following rules.

1. When an element that satisfies the XPath expression `/prices/pricerecord` is found, examine the child elements and generate output that contains the value of the `itemnumber` variable, a `|` character, the value of the `price` variable, and a newline.

2. When an `<itemnumber>` element is found, store the content of that element in the variable `itemnumber`.

3. When a `<price>` element is found, store the content of that element in the variable `price`.

### Write the gpfdist Configuration

The `gpfdist` configuration is specified as a YAML 1.1 document. It specifies rules that `gpfdist` uses to select a Transform to apply when loading or extracting data.

This example `gpfdist` configuration contains the following items:

- the `config.yaml` file defining `TRANSFORMATIONS`
- the `input_transform.sh` wrapper script, referenced in the `config.yaml` file
- the `input_transform.stx` joost transformation, called from `input_transform.sh`

Aside from the ordinary YAML rules, such as starting the document with three dashes (`---`), a `gpfdist` configuration must conform to the following restrictions:

1. a `VERSION` setting must be present with the value `1.0.0.1`.
2. a `TRANSFORMATIONS` setting must be present and contain one or more mappings.
3. Each mapping in the `TRANSFORMATION` must contain:
   - a `TYPE` with the value 'input' or 'output'
   - a `COMMAND` indicating how the transform is run.
4. Each mapping in the `TRANSFORMATION` can contain optional `CONTENT`, `SAFE`, and `STDERR` settings.

The following `gpfdist` configuration called `config.yaml` applies to the prices example. The initial indentation on each line is significant and reflects the hierarchical nature of the specification. The name `prices_input` in the following example will be referenced later when creating the table in SQL.

```yaml
---
VERSION: 1.0.0.1
TRANSFORMATIONS:
  prices_input:
    TYPE: input
    COMMAND: /bin/bash input_transform.sh %filename%
```

The `COMMAND` setting uses a wrapper script called `input_transform.sh` with a `%filename%` placeholder. When `gpfdist` runs the `prices_input` transform, it invokes `input_transform.sh` with `/bin/bash` and replaces the `%filename%` placeholder with the path to the input file to transform. The wrapper script called `input_transform.sh` contains the logic to invoke the STX transformation and return the output.

If Joost is used, the Joost STX engine must be installed.

```bash
#!/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:
```
The `input_transform.sh` file uses the Joost STX engine with the AWK interpreter. The following diagram shows the process flow as `gpfdist` runs the transformation.

Load the Data

Create the tables with SQL statements based on the appropriate schema.

There are no special requirements for the Greenplum Database tables that hold loaded data. In the prices example, the following command creates the appropriate table.

```sql
CREATE TABLE prices (  
    itemnumber integer,  
    price       decimal  
)  
DISTRIBUTED BY (itemnumber);
```

Transfer and Store the Data

Use one of the following approaches to transform the data with `gpfdist`.

- `GPLOAD` supports only input transformations, but is easier to implement in many cases.
- `INSERT INTO SELECT FROM` supports both input and output transformations, but exposes more details.
**Transforming with GPLOAD**

Transforming data with **GPLOAD** requires that the settings **TRANSFORM** and **TRANSFORM_CONFIG** appear in the **INPUT** section of the **GPLOAD** control file.

For more information about the syntax and placement of these settings in the **GPLOAD** control file, see the **Greenplum Database Reference Guide**.

- **TRANSFORM_CONFIG** specifies the name of the **gpfdist** configuration file.
- The **TRANSFORM** setting indicates the name of the transformation that is described in the file named in **TRANSFORM_CONFIG**.

```bash
---
VERSION: 1.0.0.1
DATABASE: ops
USER: gpadmin
GPLOAD:
  INPUT:  
    - TRANSFORM_CONFIG: config.yaml
    - TRANSFORM: prices_input
    - SOURCE: 
      FILE: prices.xml
```

The transformation name must appear in two places: in the **TRANSFORM** setting of the **gpfdist** configuration file and in the **TRANSFORMATIONS** section of the file named in the **TRANSFORM_CONFIG** section.

In the **GPLOAD** control file, the optional parameter **MAX_LINE_LENGTH** specifies the maximum length of a line in the XML transformation data that is passed to **gpload**.

The following diagram shows the relationships between the **GPLOAD** control file, the **gpfdist** configuration file, and the XML data file.
Transforming with INSERT INTO SELECT FROM

Specify the transformation in the CREATE EXTERNAL TABLE definition’s LOCATION clause. For example, the transform is shown in bold in the following command. (Run gpfdist first, using the command gpfdist -c config.yaml).

```
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;
```

In the command above, change `hostname` to your hostname. `prices_input` comes from the configuration file.

The following query loads data into the `prices` table.

```
INSERT INTO prices SELECT * FROM prices_readable;
```

Configuration File Format

The `gpfdist` configuration file uses the YAML 1.1 document format and implements a schema for defining the transformation parameters. The configuration file must be a valid YAML document.

The `gpfdist` program processes the document in order and uses indentation (spaces) to determine the document hierarchy and relationships of the sections to one another. The use of white space is significant. Do not use white space for formatting and do not use tabs.

The following is the basic structure of a configuration file.

```yaml
---
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**

Required. The version of the `gpfdist` configuration file schema. The current version is 1.0.0.1.

**TRANSFORMATIONS**

Required. Begins the transformation specification section. A configuration file must have at least one transformation. When `gpfdist` receives a transformation request, it looks in this section for an entry with the matching transformation name.

**TYPE**

Required. Specifies the direction of transformation. Values are *input* or *output*.

- *input*: `gpfdist` treats the standard output of the transformation process as a stream of records to load into Greenplum Database.
- *output*: `gpfdist` treats the standard input of the transformation process as a stream of records from Greenplum Database to transform and write to the appropriate output.
COMMAND

Required. Specifies the command gpfdist will execute to perform the transformation.

For input transformations, gpfdist invokes the command specified in the CONTENT setting. The command is expected to open the underlying file(s) as appropriate and produce one line of TEXT for each row to load into Greenplum Database. The input transform determines whether the entire content should be converted to one row or to multiple rows.

For output transformations, gpfdist invokes this command as specified in the CONTENT setting. The output command is expected to open and write to the underlying file(s) as appropriate. The output transformation determines the final placement of the converted output.

CONTENT

Optional. The values are data and paths. The default value is data.

- When CONTENT specifies data, the text %filename% in the COMMAND section is replaced by the path to the file to read or write.
- When CONTENT specifies paths, the text %filename% in the COMMAND section is replaced by the path to the temporary file that contains the list of files to read or write.

The following is an example of a COMMAND section showing the text %filename% that is replaced.

```
COMMAND: /bin/bash input_transform.sh %filename%
```

SAFE

Optional. A POSIX regular expression that the paths must match to be passed to the transformation. Specify SAFE when there is a concern about injection or improper interpretation of paths passed to the command. The default is no restriction on paths.

STDOUT

Optional. The values are server and console.

This setting specifies how to handle standard error output from the transformation. The default, server, specifies that gpfdist will capture the standard error output from the transformation in a temporary file and send the first 8k of that file to Greenplum Database as an error message. The error message will appear as a SQL error. Console specifies that gpfdist does not redirect or transmit the standard error output from the transformation.

XML Transformation Examples

The following examples demonstrate the complete process for different types of XML data and STX transformations. Files and detailed instructions associated with these examples are in the GitHub greenplum-db/gpdb repo in gpMgmt/demo/gpfdist_transform. Read the README file in the Before You Begin section before you run the examples. The README file explains how to download the example data file used in the examples.

Command-based External Web Tables

The output of a shell command or script defines command-based web table data. Specify the command in the EXECUTE clause of CREATE EXTERNAL WEB TABLE. The data is current as of the time the command runs. The EXECUTE clause runs the shell command or script on the specified master, and/or segment host or hosts. The command or script must reside on the hosts corresponding to the host(s) defined in the EXECUTE clause.
By default, the command is run on segment hosts when active segments have output rows to process. For example, if each segment host runs four primary segment instances that have output rows to process, the command runs four times per segment host. You can optionally limit the number of segment instances that execute the web table command. All segments included in the web table definition in the `ON` clause run the command in parallel.

The command that you specify in the external table definition executes from the database and cannot access environment variables from `.bashrc` or `.profile`. Set environment variables in the `EXECUTE` clause. For example:

```
=# CREATE EXTERNAL WEB TABLE output (output text)
   EXECUTE 'PATH=/home/gpadmin/programs; export PATH; myprogram.sh'
   FORMAT 'TEXT';
```

Scripts must be executable by the `gpadmin` user and reside in the same location on the master or segment hosts.

The following command defines a web table that runs a script. The script runs on each segment host where a segment has output rows to process.

```
=# CREATE EXTERNAL WEB TABLE log_output
   (linenum int, message text)
   EXECUTE '/var/load_scripts/get_log_data.sh' ON HOST
   FORMAT 'TEXT' (DELIMITER '|');
```

**Example 2 - IRS MeF XML Files (In demo Directory)**

This example demonstrates loading a sample IRS Modernized eFile tax return using a Joost STX transformation. The data is in the form of a complex XML file.

The U.S. Internal Revenue Service (IRS) made a significant commitment to XML and specifies its use in its Modernized e-File (MeF) system. In MeF, each tax return is an XML document with a deep hierarchical structure that closely reflects the particular form of the underlying tax code.

XML, XML Schema and stylesheets play a role in their data representation and business workflow. The actual XML data is extracted from a ZIP file attached to a MIME "transmission file" message. For more information about MeF, see [Modernized e-File (Overview)] on the IRS web site.

The sample XML document, `RET990EZ_2006.xml`, is about 350KB in size with two elements:

- ReturnHeader
- ReturnData

The `<ReturnHeader>` element contains general details about the tax return such as the taxpayer's name, the tax year of the return, and the preparer. The `<ReturnData>` element contains multiple sections with specific details about the tax return and associated schedules.

The following is an abridged sample of the XML file.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Return returnVersion="2006v2.0"
xmlns="https://www.irs.gov/efile"
xmlns:efile="https://www.irs.gov/efile"
xmlns:xsi:schemaLocation="https://www.irs.gov/efile"
xlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<ReturnHeader binaryAttachmentCount="1">
  <ReturnId>AAAAAAAAAAAAAAAAAAAA</ReturnId>
  <Timestamp>1999-05-30T12:01:01+05:01</Timestamp>
  <ReturnType>990EZ</ReturnType>
  <TaxPeriodBeginDate>2005-01-01</TaxPeriodBeginDate>
  <TaxPeriodEndDate>2005-12-31</TaxPeriodEndDate>
  <Filer>
```


The goal is to import all the data into a Greenplum database. First, convert the XML document into text with newlines "escaped", with two columns: ReturnId and a single column on the end for the entire MeF tax return. For example:

```
AAAAAAAAAAAAAAAAAAAAA|<Return returnVersion="2006v2.0"...
```

Load the data into Greenplum Database.

**Example 3 - WITSML™ Files (In demo Directory)**

This example demonstrates loading sample data describing an oil rig using a Joost STX transformation. The data is in the form of a complex XML file downloaded from energistics.org.

The Wellsite Information Transfer Standard Markup Language (WITSML™) is an oil industry initiative to provide open, non-proprietary, standard interfaces for technology and software to share information among oil companies, service companies, drilling contractors, application vendors, and regulatory agencies. For more information about WITSML™, see [http://www.energistics.org/](http://www.energistics.org/).

The oil rig information consists of a top level `<rigs>` element with multiple child elements such as `<documentInfo>`, `<rig>`, and so on. The following excerpt from the file shows the type of information in the `<rig>` tag.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="../stylesheets/rig.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 ...
    </rig>
  </rigs>
```

The goal is to import the information for this rig into Greenplum Database.

The sample document, `rig.xml`, is about 11KB in size. The input does not contain tabs so the relevant information can be converted into records delimited with a pipe (|).
With the columns:

- 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

Then, load the data into Greenplum Database.

**Formatting Data Files**

When you use the Greenplum tools for loading and unloading data, you must specify how your data is formatted. `COPY`, `CREATE EXTERNAL TABLE`, and `gpload` have clauses that allow you to specify how your data is formatted. Data can be delimited text (`TEXT`) or comma separated values (`CSV`) format. External data must be formatted correctly to be read by Greenplum Database. This topic explains the format of data files expected by Greenplum Database.

**Formatting Rows**

Greenplum Database expects rows of data to be separated by the `LF` character (Line feed, 0x0A), `CR` (Carriage return, 0x0D), or `CR` followed by `LF` (CR+LF, 0x0D 0x0A). `LF` is the standard newline representation on UNIX or UNIX-like operating systems. Operating systems such as Windows or Mac OS X use `CR` or `CR+LF`. All of these representations of a newline are supported by Greenplum Database as a row delimiter. For more information, see [Importing and Exporting Fixed Width Data](#).

**Formatting Columns**

The default column or field delimiter is the horizontal `TAB` character (0x09) for text files and the comma character (0x2C) for CSV files. You can declare a single character delimiter using the `DELIMITER` clause of `COPY`, `CREATE EXTERNAL TABLE` or `gpload` when you define your data format. The delimiter character must appear between any two data value fields. Do not place a delimiter at the beginning or end of a row. For example, if the pipe character ( | ) is your delimiter:

```
data value 1|data value 2|data value 3
```

The following command shows the use of the pipe character as a column delimiter:

```sql
=# CREATE EXTERNAL TABLE ext_table (name text, date date)
   LOCATION ('gpfdist://<hostname>/filename.txt')
   FORMAT 'TEXT' (DELIMITER '|');
```

**Representing NULL Values**

`NULL` represents an unknown piece of data in a column or field. Within your data files you can designate a string to represent null values. The default string is `\N` (backslash-N) in `TEXT` mode, or an empty value with no quotations in `CSV` mode. You can also declare a different string using the `NULL` clause of `COPY`, `CREATE EXTERNAL TABLE` or `gpload` when defining your data format. For example, you can use an empty string if you do not want to distinguish nulls from empty strings. When using the Greenplum Database loading tools, any data item that matches the designated null string is considered a null value.
Escaping

There are two reserved characters that have special meaning to Greenplum Database:

- The designated delimiter character separates columns or fields in the data file.
- The newline character designates a new row in the data file.

If your data contains either of these characters, you must escape the character so that Greenplum treats it as data and not as a field separator or new row. By default, the escape character is a \ (backslash) for text-formatted files and a double quote (") for csv-formatted files.

**Escaping in Text Formatted Files**

By default, the escape character is a \ (backslash) for text-formatted files. You can declare a different escape character in the `ESCAPE` clause of `COPY`, `CREATE EXTERNAL TABLE` or `gpload`. If your escape character appears in your data, use it to escape itself.

For example, suppose you have a table with three columns and you want to load the following three fields:

- `backslash = \`
- `vertical bar = |`
- `exclamation point = !`

Your designated delimiter character is | (pipe character), and your designated escape character is \ (backslash). The formatted row in your data file looks like this:

```
backslash = \ | vertical bar = | | exclamation point = !
```

Notice how the backslash character that is part of the data is escaped with another backslash character, and the pipe character that is part of the data is escaped with a backslash character.

You can use the escape character to escape octal and hexadecimal sequences. The escaped value is converted to the equivalent character when loaded into Greenplum Database. For example, to load the ampersand character (&), use the escape character to escape its equivalent hexadecimal (\0x26) or octal (\046) representation.

You can disable escaping in TEXT-formatted files using the `ESCAPE` clause of `COPY`, `CREATE EXTERNAL TABLE` or `gpload` as follows:

```
ESCAPE 'OFF'
```

This is useful for input data that contains many backslash characters, such as web log data.

**Escaping in CSV Formatted Files**

By default, the escape character is a " (double quote) for CSV-formatted files. If you want to use a different escape character, use the `ESCAPE` clause of `COPY`, `CREATE EXTERNAL TABLE` or `gpload` to declare a different escape character. In cases where your selected escape character is present in your data, you can use it to escape itself.

For example, suppose you have a table with three columns and you want to load the following three fields:

- Free trip to A,B
- 5.89
- Special rate "1.79"

Your designated delimiter character is , (comma), and your designated escape character is " (double quote). The formatted row in your data file looks like this:

```
"Free trip to A,B","5.89","Special rate "1.79"
```

"Free trip to A,B","5.89","Special rate "1.79""
The data value with a comma character that is part of the data is enclosed in double quotes. The double quotes that are part of the data are escaped with a double quote even though the field value is enclosed in double quotes.

Embedding the entire field inside a set of double quotes guarantees preservation of leading and trailing whitespace characters:
"Free trip to A,B ","5.89 ","Special rate ","1.79" 

Note: In CSV mode, all characters are significant. A quoted value surrounded by white space, or any characters other than DELIMITER, includes those characters. This can cause errors if you import data from a system that pads CSV lines with white space to some fixed width. In this case, preprocess the CSV file to remove the trailing white space before importing the data into Greenplum Database.

Character Encoding

Character encoding systems consist of a code that pairs each character from a character set with something else, such as a sequence of numbers or octets, to facilitate data stransmission and storage. Greenplum Database supports a variety of character sets, including single-byte character sets such as the ISO 8859 series and multiple-byte character sets such as EUC (Extended UNIX Code), UTF-8, and Mule internal code. The server-side character set is defined during database initialization, UTF-8 is the default and can be changed. Clients can use all supported character sets transparently, but a few are not supported for use within the server as a server-side encoding. When loading or inserting data into Greenplum Database, Greenplum transparently converts the data from the specified client encoding into the server encoding. When sending data back to the client, Greenplum converts the data from the server character encoding into the specified client encoding.

Data files must be in a character encoding recognized by Greenplum Database. See the Greenplum Database Reference Guide for the supported character sets. Data files that contain invalid or unsupported encoding sequences encounter errors when loading into Greenplum Database.

Note: On data files generated on a Microsoft Windows operating system, run the dos2unix system command to remove any Windows-only characters before loading into Greenplum Database.

Changing the Client-Side Character Encoding

The client-side character encoding can be changed for a session by setting the server configuration parameter client_encoding

SET client_encoding TO 'latin1';

Change the client-side character encoding back to the default value:

RESET client_encoding;

Show the current client-side character encoding setting:

SHOW client_encoding;
Example Custom Data Access Protocol

The following is the API for the Greenplum Database custom data access protocol. The example protocol implementation `gpextprotocol.c` is written in C and shows how the API can be used. For information about accessing a custom data access protocol, see Using a Custom Protocol.

```c
/* ---- 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)
```

Notes

The protocol corresponds to the example described in Using a Custom Protocol. The source code file name and shared object are `gpextprotocol.c` and `gpextprotocol.so`.

The protocol has the following properties:

- The name defined for the protocol is `myprot`.
- The protocol has the following simple form: the protocol name and a path, separated by `://`
  - `myprot://path`
- Three functions are implemented:
  - `myprot_import()` a read function
  - `myprot_export()` a write function
  - `myprot_validate_urls()` a validation function

  These functions are referenced in the `CREATE PROTOCOL` statement when the protocol is created and declared in the database.

The example implementation `gpextprotocol.c` uses `fopen()` and `fread()` to simulate a simple protocol that reads local files. In practice, however, the protocol would implement functionality such as a remote connection to some process over the network.

Installing the External Table Protocol

To use the example external table protocol, you use the C compiler `cc` to compile and link the source code to create a shared object that can be dynamically loaded by Greenplum Database. The commands to compile and link the source code on a Linux system are similar to this:

```
cc -fpic -c gpextproto.c cc -shared -o gpextprotocol.so gpextprotocol.o
```

The option `-fpic` specifies creating position-independent code (PIC) and the `-c` option compiles the source code without linking and creates an object file. The object file needs to be created as position-independent code (PIC) so that it can be loaded at any arbitrary location in memory by Greenplum Database.
The flag `-shared` specifies creating a shared object (shared library) and the `-o` option specifies the shared object file name `gpextprotocol.so`. Refer to the GCC manual for more information on the `cc` options.

The header files that are declared as include files in `gpextprotocol.c` are located in subdirectories of `$GPHOME/include/postgresql/`.

For more information on compiling and linking dynamically-loaded functions and examples of compiling C source code to create a shared library on other operating systems, see the Postgres documentation at https://www.postgresql.org/docs/8.4/static/xfunc-c.html#DFUNC.

The manual pages for the C compiler `cc` and the link editor `ld` for your operating system also contain information on compiling and linking source code on your system.

The compiled code (shared object file) for the custom protocol must be placed in the same location on every host in your Greenplum Database array (master and all segments). This location must also be in the `LD_LIBRARY_PATH` so that the server can locate the files. It is recommended to locate shared libraries either relative to `$libdir` (which is located at `$GPHOME/lib`) or through the dynamic library path (set by the `dynamic_library_path` server configuration parameter) on all master segment instances in the Greenplum Database array. You can use the Greenplum Database utilities `gpssh` and `gpscp` to update segments.

### gpextprotocol.c

```c
#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 = EXTPROTOCOL_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_EXTPROTOCOL(fcinfo))
        elog(ERROR, "myprot_export: 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 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 = 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, "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)));

EXTPROTOCOL_SET_USER_CTX(fcinfo, myData);
}

/* DO THE EXPORT
* *******************************************************/
data = EXTPROTOCOL_GET_DATABUF(fcinfo);
datlen = EXTPROTOCOL_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);
}


Managing Performance

The topics in this section cover Greenplum Database performance management, including how to monitor performance and how to configure workloads to prioritize resource utilization.

This section contains the following topics:

• Defining Database Performance
• Common Causes of Performance Issues
• Using Resource Queues
• Investigating a Performance Problem

Defining Database Performance

Managing system performance includes measuring performance, identifying the causes of performance problems, and applying the tools and techniques available to you to remedy the problems.

Greenplum measures database performance based on the rate at which the database management system (DBMS) supplies information to requesters.

Understanding the Performance Factors

Several key performance factors influence database performance. Understanding these factors helps identify performance opportunities and avoid problems:

• System Resources
• Workload
• Throughput
• Contention
• Optimization

System Resources

Database performance relies heavily on disk I/O and memory usage. To accurately set performance expectations, you need to know the baseline performance of the hardware on which your DBMS is deployed. Performance of hardware components such as CPUs, hard disks, disk controllers, RAM, and network interfaces will significantly affect how fast your database performs.

Workload

The workload equals the total demand from the DBMS, and it varies over time. The total workload is a combination of user queries, applications, batch jobs, transactions, and system commands directed through the DBMS at any given time. For example, it can increase when month-end reports are run or decrease on weekends when most users are out of the office. Workload strongly influences database performance. Knowing your workload and peak demand times helps you plan for the most efficient use of your system resources and enables processing the largest possible workload.

Throughput

A system’s throughput defines its overall capability to process data. DBMS throughput is measured in queries per second, transactions per second, or average response times. DBMS throughput is closely related to the processing capacity of the underlying systems (disk I/O, CPU speed, memory bandwidth, and so on), so it is important to know the throughput capacity of your hardware when setting DBMS throughput goals.
Contention

Contention is the condition in which two or more components of the workload attempt to use the system in a conflicting way — for example, multiple queries that try to update the same piece of data at the same time or multiple large workloads that compete for system resources. As contention increases, throughput decreases.

Optimization

DBMS optimizations can affect the overall system performance. SQL formulation, database configuration parameters, table design, data distribution, and so on enable the database query optimizer to create the most efficient access plans.

Determining Acceptable Performance

When approaching a performance tuning initiative, you should know your system’s expected level of performance and define measurable performance requirements so you can accurately evaluate your system’s performance. Consider the following when setting performance goals:

• Baseline Hardware Performance
• Performance Benchmarks

Baseline Hardware Performance

Most database performance problems are caused not by the database, but by the underlying systems on which the database runs. I/O bottlenecks, memory problems, and network issues can notably degrade database performance. Knowing the baseline capabilities of your hardware and operating system (OS) will help you identify and troubleshoot hardware-related problems before you explore database-level or query-level tuning initiatives.

See the Greenplum Database Reference Guide for information about running the gpcheckperf utility to validate hardware and network performance.

Performance Benchmarks

To maintain good performance or fix performance issues, you should know the capabilities of your DBMS on a defined workload. A benchmark is a predefined workload that produces a known result set. Periodically run the same benchmark tests to help identify system-related performance degradation over time. Use benchmarks to compare workloads and identify queries or applications that need optimization.

Many third-party organizations, such as the Transaction Processing Performance Council (TPC), provide benchmark tools for the database industry. TPC provides TPC-H, a decision support system that examines large volumes of data, executes queries with a high degree of complexity, and gives answers to critical business questions. For more information about TPC-H, go to:

http://www.tpc.org/tpch

Common Causes of Performance Issues

This section explains the troubleshooting processes for common performance issues and potential solutions to these issues.

Identifying Hardware and Segment Failures

The performance of Greenplum Database depends on the hardware and IT infrastructure on which it runs. Greenplum Database is comprised of several servers (hosts) acting together as one cohesive system (array); as a first step in diagnosing performance problems, ensure that all Greenplum Database segments are online. Greenplum Database's performance will be as fast as the slowest host in the array. Problems
with CPU utilization, memory management, I/O processing, or network load affect performance. Common
hardware-related issues are:

- **Disk Failure** – Although a single disk failure should not dramatically affect database performance if
  you are using RAID, disk resynchronization does consume resources on the host with failed disks. The
  `gpcheckperf` utility can help identify segment hosts that have disk I/O issues.

- **Host Failure** – When a host is offline, the segments on that host are nonoperational. This means
  other hosts in the array must perform twice their usual workload because they are running the primary
  segments and multiple mirrors. If mirrors are not enabled, service is interrupted. Service is temporarily
  interrupted to recover failed segments. The `gpstate` utility helps identify failed segments.

- **Network Failure** – Failure of a network interface card, a switch, or DNS server can bring down
  segments. If host names or IP addresses cannot be resolved within your Greenplum array, these
  manifest themselves as interconnect errors in Greenplum Database. The `gpcheckperf` utility helps
  identify segment hosts that have network issues.

- **Disk Capacity** – Disk capacity on your segment hosts should never exceed 70 percent full. Greenplum
  Database needs some free space for runtime processing. To reclaim disk space that deleted rows
  occupy, run `VACUUM` after loads or updates. The `gp_toolkit` administrative schema has many views for
  checking the size of distributed database objects.

  See the *Greenplum Database Reference Guide* for information about checking database object sizes
  and disk space.

### Managing Workload

A database system has a limited CPU capacity, memory, and disk I/O resources. When multiple workloads
compete for access to these resources, database performance suffers. Resource management maximizes
system throughput while meeting varied business requirements. Greenplum Database provides resource
queues and resource groups to help you manage these system resources.

Resource queues and resource groups limit resource usage and the total number of concurrent queries
executing in the particular queue or group. By assigning database roles to the appropriate queue or
  
  group, administrators can control concurrent user queries and prevent system overload. For more
  information about resource queues and resource groups, including selecting the appropriate scheme for
  your Greenplum Database environment, see *Managing Resources*.

Greenplum Database administrators should run maintenance workloads such as data loads and `VACUUM`
`ANALYZE` operations after business hours. Do not compete with database users for system resources;
perform administrative tasks at low-usage times.

### Avoiding Contention

Contention arises when multiple users or workloads try to use the system in a conflicting way; for example,
contention occurs when two transactions try to update a table simultaneously. A transaction that seeks a
  
  table-level or row-level lock will wait indefinitely for conflicting locks to be released. Applications should not
  hold transactions open for long periods of time, for example, while waiting for user input.

### Maintaining Database Statistics

Greenplum Database uses a cost-based query optimizer that relies on database statistics. Accurate
statistics allow the query optimizer to better estimate the number of rows retrieved by a query to choose
the most efficient query plan. Without database statistics, the query optimizer cannot estimate how many
records will be returned. The optimizer does not assume it has sufficient memory to perform certain
operations such as aggregations, so it takes the most conservative action and does these operations by
reading and writing from disk. This is significantly slower than doing them in memory. `ANALYZE` collects
statistics about the database that the query optimizer needs.

**Note:** When executing an SQL command with GPOPCA, Greenplum Database issues a warning if
the command performance could be improved by collecting statistics on a column or set of columns
referenced by the command. The warning is issued on the command line and information is added to the Greenplum Database log file. For information about collecting statistics on table columns, see the ANALYZE command in the Greenplum Database Reference Guide

**Identifying Statistics Problems in Query Plans**

Before you interpret a query plan for a query using EXPLAIN or EXPLAIN ANALYZE, familiarize yourself with the data to help identify possible statistics problems. Check the plan for the following indicators of inaccurate statistics:

- **Are the optimizer's estimates close to reality?** Run EXPLAIN ANALYZE and see if the number of rows the optimizer estimated is close to the number of rows the query operation returned.
- **Are selective predicates applied early in the plan?** The most selective filters should be applied early in the plan so fewer rows move up the plan tree.
- **Is the optimizer choosing the best join order?** When you have a query that joins multiple tables, make sure the optimizer chooses the most selective join order. Joins that eliminate the largest number of rows should be done earlier in the plan so fewer rows move up the plan tree.

See Query Profiling for more information about reading query plans.

**Tuning Statistics Collection**

The following configuration parameters control the amount of data sampled for statistics collection:

- `default_statistics_target`
- `gp_analyze_relative_error`

These parameters control statistics sampling at the system level. It is better to sample only increased statistics for columns used most frequently in query predicates. You can adjust statistics for a particular column using the command:

```
ALTER TABLE...SET STATISTICS
```

For example:

```
ALTER TABLE sales ALTER COLUMN region SET STATISTICS 50;
```

This is equivalent to increasing `default_statistics_target` for a particular column. Subsequent ANALYZE operations will then gather more statistics data for that column and produce better query plans as a result.

**Optimizing Data Distribution**

When you create a table in Greenplum Database, you must declare a distribution key that allows for even data distribution across all segments in the system. Because the segments work on a query in parallel, Greenplum Database will always be as fast as the slowest segment. If the data is unbalanced, the segments that have more data will return their results slower and therefore slow down the entire system.

**Optimizing Your Database Design**

Many performance issues can be improved by database design. Examine your database design and consider the following:

- Does the schema reflect the way the data is accessed?
- Can larger tables be broken down into partitions?
- Are you using the smallest data type possible to store column values?
- Are columns used to join tables of the same datatype?
- Are your indexes being used?
**Greenplum Database Maximum Limits**

To help optimize database design, review the maximum limits that Greenplum Database supports:

**Table 64: Maximum Limits of Greenplum Database**

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Size</td>
<td>Unlimited</td>
</tr>
<tr>
<td>Table Size</td>
<td>Unlimited, 128 TB per partition per segment</td>
</tr>
<tr>
<td>Row Size</td>
<td>1.6 TB (1600 columns * 1 GB)</td>
</tr>
<tr>
<td>Field Size</td>
<td>1 GB</td>
</tr>
<tr>
<td>Rows per Table</td>
<td>281474976710656 (2^48)</td>
</tr>
<tr>
<td>Columns per Table/View</td>
<td>1600</td>
</tr>
<tr>
<td>Indexes per Table</td>
<td>Unlimited</td>
</tr>
<tr>
<td>Columns per Index</td>
<td>32</td>
</tr>
<tr>
<td>Table-level Constraints per Table</td>
<td>Unlimited</td>
</tr>
<tr>
<td>Table Name Length</td>
<td>63 Bytes (Limited by name data type)</td>
</tr>
</tbody>
</table>

Dimensions listed as unlimited are not intrinsically limited by Greenplum Database. However, they are limited in practice to available disk space and memory/swap space. Performance may suffer when these values are unusually large.

**Note:**

There is a maximum limit on the number of objects (tables, indexes, and views, but not rows) that may exist at one time. This limit is 4294967296 (2^32).

**Greenplum Database Memory Overview**

Memory is a key resource for a Greenplum Database system and, when used efficiently, can ensure high performance and throughput. This topic describes how segment host memory is allocated between segments and the options available to administrators to configure memory.

A Greenplum Database segment host runs multiple PostgreSQL instances, all sharing the host's memory. The segments have an identical configuration and they consume similar amounts of memory, CPU, and disk IO simultaneously, while working on queries in parallel.

For best query throughput, the memory configuration should be managed carefully. There are memory configuration options at every level in Greenplum Database, from operating system parameters, to managing resources with resource queues and resource groups, to setting the amount of memory allocated to an individual query.

**Segment Host Memory**

On a Greenplum Database segment host, the available host memory is shared among all the processes executing on the computer, including the operating system, Greenplum Database segment instances, and other application processes. Administrators must determine what Greenplum Database and non-Greenplum Database processes share the hosts' memory and configure the system to use the memory efficiently. It is equally important to monitor memory usage regularly to detect any changes in the way host memory is consumed by Greenplum Database or other processes.

The following figure illustrates how memory is consumed on a Greenplum Database segment host.
Figure 33: Greenplum Database Segment Host Memory

Beginning at the bottom of the illustration, the line labeled A represents the total host memory. The line directly above line A shows that the total host memory comprises both physical RAM and swap space.

The line labelled B shows that the total memory available must be shared by Greenplum Database and all other processes on the host. Non-Greenplum Database processes include the operating system and any other applications, for example system monitoring agents. Some applications may use a significant portion of memory and, as a result, you may have to adjust the number of segments per Greenplum Database host or the amount of memory per segment.

The segments (C) each get an equal share of the Greenplum Database Memory (B).

Within a segment, the currently active resource management scheme, Resource Queues or Resource Groups, governs how memory is allocated to execute a SQL statement. These constructs allow you to translate business requirements into execution policies in your Greenplum Database system and to guard against queries that could degrade performance. For an overview of resource groups and resource queues, refer to Managing Resources.

Options for Configuring Segment Host Memory

Host memory is the total memory shared by all applications on the segment host. You can configure the amount of host memory using any of the following methods:

- Add more RAM to the nodes to increase the physical memory.
- Allocate swap space to increase the size of virtual memory.
- Set the kernel parameters `vm.overcommit_memory` and `vm.overcommit_ratio` to configure how the operating system handles large memory allocation requests.

The physical RAM and OS configuration are usually managed by the platform team and system administrators. See the Greenplum Database Installation Guide for the recommended kernel parameter settings.

The amount of memory to reserve for the operating system and other processes is workload dependent. The minimum recommendation for operating system memory is 32GB, but if there is much concurrency in Greenplum Database, increasing to 64GB of reserved memory may be required. The largest user of operating system memory is SLAB, which increases as Greenplum Database concurrency and the number of sockets used increases.
The `vm.overcommit_memory` kernel parameter should always be set to 2, the only safe value for Greenplum Database.

The `vm.overcommit_ratio` kernel parameter sets the percentage of RAM that is used for application processes, the remainder reserved for the operating system. The default for Red Hat is 50 (50%). Setting this parameter too high may result in insufficient memory reserved for the operating system, which can cause segment host failure or database failure. Leaving the setting at the default of 50 is generally safe, but conservative. Setting the value too low reduces the amount of concurrency and the complexity of queries you can run at the same time by reducing the amount of memory available to Greenplum Database. When increasing `vm.overcommit_ratio`, it is important to remember to always reserve some memory for operating system activities.

To calculate a safe value for `vm.overcommit_ratio`, first determine the total memory available to Greenplum Database processes, called `gp_vmem`, with this formula:

\[
gp_vmem = \frac{((SWAP + RAM) - (7.5GB + 0.05 \times RAM))}{1.7}
\]

where `SWAP` is the swap space on the host in GB, and `RAM` is the number of GB of RAM installed on the host.

Calculate the `vm.overcommit_ratio` value with this formula:

\[
vm.overcommit_ratio = \frac{(RAM - 0.026 \times gp_vmem)}{RAM}
\]

### Configuring Greenplum Database Memory

Greenplum Database Memory is the amount of memory available to all Greenplum Database segment instances.

When you set up the Greenplum Database cluster, you determine the number of primary segments to run per host and the amount of memory to allocate for each segment. Depending on the CPU cores, amount of physical RAM, and workload characteristics, the number of segments is usually a value between 4 and 8. With segment mirroring enabled, it is important to allocate memory for the maximum number of primary segments executing on a host during a failure. For example, if you use the default grouping mirror configuration, a segment host failure doubles the number of acting primaries on the host that has the failed host’s mirrors. Mirror configurations that spread each host’s mirrors over multiple other hosts can lower the maximum, allowing more memory to be allocated for each segment. For example, if you use a block mirroring configuration with 4 hosts per block and 8 primary segments per host, a single host failure would cause other hosts in the block to have a maximum of 11 active primaries, compared to 16 for the default grouping mirror configuration.

The `gp_vmem_protect_limit` server configuration parameter value identifies the amount of memory to allocate to each segment. This value is estimated by calculating the memory available for all Greenplum Database processes and dividing by the maximum number of primary segments during a failure. If `gp_vmem_protect_limit` is set too high, queries can fail. Use the following formula to calculate a safe value for `gp_vmem_protect_limit`. Provide the `gp_vmem` value you calculated earlier.

\[
gp_vmem_protect_limit = \frac{gp_vmem}{max_acting_primary_segments}
\]

where `max_acting_primary_segments` is the maximum number of primary segments that could be running on a host when mirror segments are activated due to a host or segment failure.

**Note:** The resource management scheme currently active (resource queues or resource groups) for your Greenplum Database cluster may impose additional considerations on `gp_vmem_protect_limit`.

Resource queues and resource groups support additional configuration parameters that enable you to further control and refine the amount of memory allocated for queries.
Managing Resources

Greenplum Database provides features to help you prioritize and allocate resources to queries according to business requirements and to prevent queries from starting when resources are unavailable.

You can use resource management features to limit the number of concurrent queries, the amount of memory used to execute a query, and the relative amount of CPU devoted to processing a query. Greenplum Database provides two schemes to manage resources - Resource Queues and Resource Groups.

**Warning:** Resource groups are an experimental feature and are not intended for use in a production environment. Experimental features are subject to change without notice in future releases.

Significant Greenplum Database performance degradation has been observed when enabling resource group-based workload management on RedHat 6.x, CentOS 6.x, and SuSE 11 systems. This issue is caused by a Linux cgroup kernel bug. This kernel bug has been fixed in CentOS 7.x and Red Hat 7.x systems. The performance problem is mitigated in Red Hat 6 systems that use kernel version 2.6.32-696 or higher. SuSE 11 does not have a kernel version that resolves this issue.

Either the resource queue or the resource group management scheme can be active in Greenplum Database; both schemes cannot be active at the same time.

Resource queues are enabled by default when you install your Greenplum Database cluster. While you can create and assign resource groups when resource queues are active, you must explicitly enable resource groups to start using that management scheme.

The following table summarizes some of the differences between Resource Queues and Resource Groups.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Resource Queues</th>
<th>Resource Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrency</td>
<td>Managed at the query level</td>
<td>Managed at the transaction level</td>
</tr>
<tr>
<td>CPU</td>
<td>Specify query priority</td>
<td>Specify percentage of CPU resources; uses Linux Control Groups</td>
</tr>
<tr>
<td>Memory</td>
<td>Managed at the queue and operator level; users can over-subscribe</td>
<td>Managed at the transaction level, with enhanced allocation and tracking; users cannot over-subscribe</td>
</tr>
<tr>
<td>Memory Isolation</td>
<td>None</td>
<td>Memory is isolated between resource groups and between transactions within the same resource group</td>
</tr>
<tr>
<td>Users</td>
<td>Limits are applied only to non-admin users</td>
<td>Limits are applied to <strong>SUPERUSER</strong> and non-admin users alike</td>
</tr>
<tr>
<td>Queueing</td>
<td>Queue only when no slot available</td>
<td>Queue when no slot is available or not enough available memory</td>
</tr>
<tr>
<td>Query Failure</td>
<td>Query may fail immediately if not enough memory</td>
<td>Query may fail after reaching transaction memory limit</td>
</tr>
<tr>
<td>Limit Bypass</td>
<td>Limits are not enforced for <strong>SUPERUSER</strong> roles and certain operators and functions</td>
<td>Limits are not enforced on <strong>SET</strong> and <strong>SHOW</strong> commands</td>
</tr>
</tbody>
</table>
Using Resource Queues

Use Greenplum Database resource queues to prioritize and allocate resources to queries according to business requirements and to prevent queries from starting when resources are unavailable.

Resource queues are one tool to manage the degree of concurrency in a Greenplum Database system. Resource queues are database objects that you create with the `CREATE RESOURCE QUEUE` SQL statement. You can use them to manage the number of active queries that may execute concurrently, the amount of memory each type of query is allocated, and the relative priority of queries. Resource queues can also guard against queries that would consume too many resources and degrade overall system performance.

Each database role is associated with a single resource queue; multiple roles can share the same resource queue. Roles are assigned to resource queues using the `RESOURCE QUEUE` phrase of the `CREATE ROLE` or `ALTER ROLE` statements. If a resource queue is not specified, the role is associated with the default resource queue, `pg_default`.

When the user submits a query for execution, the query is evaluated against the resource queue’s limits. If the query does not cause the queue to exceed its resource limits, then that query will run immediately. If the query causes the queue to exceed its limits (for example, if the maximum number of active statement slots are currently in use), then the query must wait until queue resources are free before it can run.

Queries are evaluated on a first in, first out basis. If query prioritization is enabled, the active workload on the system is periodically assessed and processing resources are reallocated according to query priority (see How Priorities Work). Roles with the `SUPERUSER` attribute are exempt from resource queue limits. Superuser queries always run immediately regardless of limits imposed by their assigned resource queue.

Figure 34: Resource Queue Process

Resource queues define classes of queries with similar resource requirements. Administrators should create resource queues for the various types of workloads in their organization. For example, you could create resource queues for the following classes of queries, corresponding to different service level agreements:

- ETL queries
- Reporting queries
- Executive queries

A resource queue has the following characteristics:

```.memory_limit```

The amount of memory used by all the queries in the queue (per segment). For example, setting `MEMORY_LIMIT` to 2GB on the ETL queue allows ETL queries to use up to 2GB of memory in each segment.
**ACTIVE_STATEMENTS**

The number of slots for a queue; the maximum concurrency level for a queue. When all slots are used, new queries must wait. Each query uses an equal amount of memory by default.

For example, the `pg_default` resource queue has `ACTIVE_STATEMENTS = 20`.

**PRIORITY**

The relative CPU usage for queries. This may be one of the following levels: LOW, MEDIUM, HIGH, MAX. The default level is MEDIUM. The query prioritization mechanism monitors the CPU usage of all the queries running in the system, and adjusts the CPU usage for each to conform to its priority level. For example, you could set MAX priority to the executive resource queue and MEDIUM to other queues to ensure that executive queries receive a greater share of CPU.

**MAX_COST**

Query plan cost limit.

The Greenplum Database optimizer assigns a numeric cost to each query. If the cost exceeds the `MAX_COST` value set for the resource queue, the query is rejected as too expensive.

*Note:* Use `MEMORY_LIMIT` and `ACTIVE_STATEMENTS` to set limits for resource queues rather than `MAX_COST`.

The default configuration for a Greenplum Database system has a single default resource queue named `pg_default`. The `pg_default` resource queue has an `ACTIVE_STATEMENTS` setting of 20, no `MEMORY_LIMIT`, medium `PRIORITY`, and no set `MAX_COST`. This means that all queries are accepted and run immediately, at the same priority and with no memory limitations; however, only twenty queries may execute concurrently.

The number of concurrent queries a resource queue allows depends on whether the `MEMORY_LIMIT` parameter is set:

- If no `MEMORY_LIMIT` is set for a resource queue, the amount of memory allocated per query is the value of the `statement_mem` server configuration parameter. The maximum memory the resource queue can use is the product of `statement_mem` and `ACTIVE_STATEMENTS`.
- When a `MEMORY_LIMIT` is set on a resource queue, the number of queries that the queue can execute concurrently is limited by the queue's available memory.

A query admitted to the system is allocated an amount of memory and a query plan tree is generated for it. Each node of the tree is an operator, such as a sort or hash join. Each operator is a separate execution thread and is allocated a fraction of the overall statement memory, at minimum 100KB. If the plan has a large number of operators, the minimum memory required for operators can exceed the available memory and the query will be rejected with an insufficient memory error. Operators determine if they can complete their tasks in the memory allocated, or if they must spill data to disk, in work files. The mechanism that allocates and controls the amount of memory used by each operator is called memory quota.

Not all SQL statements submitted through a resource queue are evaluated against the queue limits. By default only `SELECT`, `SELECT INTO`, `CREATE TABLE AS SELECT`, and `DECLARE CURSOR` statements are evaluated. If the server configuration parameter `resource_select_only` is set to `off`, then `INSERT`, `UPDATE`, and `DELETE` statements will be evaluated as well.

Also, an SQL statement that is run during the execution of an `EXPLAIN ANALYZE` command is excluded from resource queues.

**Resource Queue Example**

The default resource queue, `pg_default`, allows a maximum of 20 active queries and allocates the same amount of memory to each. This is generally not adequate resource control for production systems. To ensure that the system meets performance expectations, you can define classes of queries and assign
them to resource queues configured to execute them with the concurrency, memory, and CPU resources best suited for that class of query.

The following illustration shows an example resource queue configuration for a Greenplum Database system with gp_vmem_protect_limit set to 8GB:

![Resource Queue Configuration Example](image)

**Figure 35: Resource Queue Configuration Example**

This example has three classes of queries with different characteristics and service level agreements (SLAs). Three resource queues are configured for them. A portion of the segment memory is reserved as a safety margin.

<table>
<thead>
<tr>
<th>Resource Queue Name</th>
<th>Active Statements</th>
<th>Memory Limit</th>
<th>Memory per Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETL</td>
<td>3</td>
<td>2GB</td>
<td>667MB</td>
</tr>
<tr>
<td>Reporting</td>
<td>7</td>
<td>3GB</td>
<td>429MB</td>
</tr>
<tr>
<td>Executive</td>
<td>1</td>
<td>1.4GB</td>
<td>1.4GB</td>
</tr>
</tbody>
</table>

The total memory allocated to the queues is 6.4GB, or 80% of the total segment memory defined by the gp_vmem_protect_limit server configuration parameter. Allowing a safety margin of 20% accommodates some operators and queries that are known to use more memory than they are allocated by the resource queue.

See the CREATE RESOURCE QUEUE and CREATE/ALTER ROLE statements in the Greenplum Database Reference Guide for help with command syntax and detailed reference information.

**How Memory Limits Work**

Setting MEMORY_LIMIT on a resource queue sets the maximum amount of memory that all active queries submitted through the queue can consume for a segment instance. The amount of memory allotted to a query is the queue memory limit divided by the active statement limit. (Use the memory limits in conjunction with statement-based queues rather than cost-based queues.) For example, if a queue has a memory limit of 2000MB and an active statement limit of 10, each query submitted through the queue is allotted 200MB of memory by default. The default memory allotment can be overridden on a per-query basis using the statement_mem server configuration parameter (up to the queue memory limit). Once a query has started executing, it holds its allotted memory in the queue until it completes, even if during execution it actually consumes less than its allotted amount of memory.

You can use the statement_mem server configuration parameter to override memory limits set by the current resource queue. At the session level, you can increase statement_mem up to the resource queue’s MEMORY_LIMIT. This will allow an individual query to use all of the memory allocated for the entire queue without affecting other resource queues.

The value of statement_mem is capped using the max_statement_mem configuration parameter (a superuser parameter). For a query in a resource queue with MEMORY_LIMIT set, the maximum value for statement_mem is min(MEMORY_LIMIT, max_statement_mem). When a query is admitted, the memory allocated to it is subtracted from MEMORY_LIMIT. If MEMORY_LIMIT is exhausted, new queries in the same resource queue must wait. This happens even if ACTIVE_STATEMENTS has not yet been
reached. Note that this can happen only when `statement_mem` is used to override the memory allocated by the resource queue.

For example, consider a resource queue named `adhoc` with the following settings:

- **MEMORY_LIMIT** is 1.5GB
- **ACTIVE_STATEMENTS** is 3

By default each statement submitted to the queue is allocated 500MB of memory. Now consider the following series of events:

1. User `ADHOC_1` submits query `Q1`, overriding `STATEMENT_MEM` to 800MB. The `Q1` statement is admitted into the system.
2. User `ADHOC_2` submits query `Q2`, using the default 500MB.
3. With `Q1` and `Q2` still running, user `ADHOC3` submits query `Q3`, using the default 500MB.

Queries `Q1` and `Q2` have used 1300MB of the queue’s 1500MB. Therefore, `Q3` must wait for `Q1` or `Q2` to complete before it can run.

If **MEMORY_LIMIT** is not set on a queue, queries are admitted until all of the **ACTIVE_STATEMENTS** slots are in use, and each query can set an arbitrarily high `statement_mem`. This could lead to a resource queue using unbounded amounts of memory.

For more information on configuring memory limits on a resource queue, and other memory utilization controls, see Creating Queues with Memory Limits.

### How Priorities Work

The **PRIORITY** setting for a resource queue differs from the **MEMORY_LIMIT** and **ACTIVE_STATEMENTS** settings, which determine whether a query will be admitted to the queue and eventually executed. The **PRIORITY** setting applies to queries after they become active. Active queries share available CPU resources as determined by the priority settings for its resource queue. When a statement from a high-priority queue enters the group of actively running statements, it may claim a greater share of the available CPU, reducing the share allocated to already-running statements in queues with a lesser priority setting.

The comparative size or complexity of the queries does not affect the allotment of CPU. If a simple, low-cost query is running simultaneously with a large, complex query, and their priority settings are the same, they will be allocated the same share of available CPU resources. When a new query becomes active, the CPU shares will be recalculated, but queries of equal priority will still have equal amounts of CPU.

For example, an administrator creates three resource queues: `adhoc` for ongoing queries submitted by business analysts, `reporting` for scheduled reporting jobs, and `executive` for queries submitted by executive user roles. The administrator wants to ensure that scheduled reporting jobs are not heavily affected by unpredictable resource demands from ad-hoc analyst queries. Also, the administrator wants to make sure that queries submitted by executive roles are allotted a significant share of CPU. Accordingly, the resource queue priorities are set as shown:

- **adhoc** — Low priority
- **reporting** — High priority
- **executive** — Maximum priority

At runtime, the CPU share of active statements is determined by these priority settings. If queries 1 and 2 from the reporting queue are running simultaneously, they have equal shares of CPU. When an ad-hoc query becomes active, it claims a smaller share of CPU. The exact share used by the reporting queries is adjusted, but remains equal due to their equal priority setting:
Figure 36: CPU share readjusted according to priority

Note:

The percentages shown in these illustrations are approximate. CPU usage between high, low and maximum priority queues is not always calculated in precisely these proportions.

When an executive query enters the group of running statements, CPU usage is adjusted to account for its maximum priority setting. It may be a simple query compared to the analyst and reporting queries, but until it is completed, it will claim the largest share of CPU.
Figure 37: CPU share readjusted for maximum priority query

For more information about commands to set priorities, see Setting Priority Levels.

**Steps to Enable Resource Management**

Enabling and using resource management in Greenplum Database involves the following high-level tasks:

2. Create the resource queues and set limits on them. See Creating Resource Queues and Modifying Resource Queues.
3. Assign a queue to one or more user roles. See Assigning Roles (Users) to a Resource Queue.
4. Use the resource management system views to monitor and manage the resource queues. See Checking Resource Queue Status.

**Configuring Resource Management**

Resource scheduling is enabled by default when you install Greenplum Database, and is required for all roles. The default resource queue, pg_default, has an active statement limit of 20, no memory limit, and a medium priority setting. Create resource queues for the various types of workloads.

**To configure resource management**

1. The following parameters are for the general configuration of resource queues:
   - max_resource_queues - Sets the maximum number of resource queues.
   - max_resource_portals_per_transaction - Sets the maximum number of simultaneously open cursors allowed per transaction. Note that an open cursor will hold an active query slot in a resource queue.
• resource_select_only - If set to on, then SELECT, SELECT INTO, CREATE TABLE ASSELECT, and DECLARE CURSOR commands are evaluated. If set to off, INSERT, UPDATE, and DELETE commands will be evaluated as well.
• resource_cleanup_gangs_on_wait - Cleans up idle segment worker processes before taking a slot in the resource queue.
• stats_queue_level - Enables statistics collection on resource queue usage, which can then be viewed by querying the pg_stat_resqueues system view.

2. The following parameters are related to memory utilization:

• gp_resqueue_memory_policy - Enables Greenplum Database memory management features.
  
  In Greenplum Database 4.2 and later, the distribution algorithm eager_free takes advantage of the fact that not all operators execute at the same time. The query plan is divided into stages and Greenplum Database eagerly frees memory allocated to a previous stage at the end of that stage’s execution, then allocates the eagerly freed memory to the new stage.

  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.
• statement_mem and max_statement_mem - Used to allocate memory to a particular query at runtime (override the default allocation assigned by the resource queue). max_statement_mem is set by database superusers to prevent regular database users from over-allocation.
• gp_vmem_protect_limit - Sets the upper boundary that all query processes can consume and should not exceed the amount of physical memory of a segment host. When a segment host reaches this limit during query execution, the queries that cause the limit to be exceeded will be cancelled.
• gp_vmem_idle_resource_timeout and gp_vmem_protect_segworker_cache_limit - used to free memory on segment hosts held by idle database processes. Administrators may want to adjust these settings on systems with lots of concurrency.
• shared_buffers - Sets the amount of memory a Greenplum server instance uses for shared memory buffers. This setting must be at least 128 kilobytes and at least 16 kilobytes times max_connections. The value must not exceed the operating system shared memory maximum allocation request size, shmax on Linux. See the Greenplum Database Installation Guide for recommended OS memory settings for your platform.

3. The following parameters are related to query prioritization. Note that the following parameters are all local parameters, meaning they must be set in the postgresql.conf files of the master and all segments:

• gp_resqueue_priority - The query prioritization feature is enabled by default.
• gp_resqueue_priority_sweeper_interval - Sets the interval at which CPU usage is recalculated for all active statements. The default value for this parameter should be sufficient for typical database operations.
• gp_resqueue_priority_cpucores_per_segment - Specifies the number of CPU cores allocated per segment instance. The default value is 4 for the master and segments. For Greenplum Data Computing Appliance Version 2, the default value is 4 for segments and 25 for the master.

  Each host checks its own postgresql.conf file for the value of this parameter. This parameter also affects the master node, where it should be set to a value reflecting the higher ratio of CPU cores. For example, on a cluster that has 10 CPU cores per host and 4 segments per host, you would specify these values for gp_resqueue_priority_cpucores_per_segment:

  10 for the master and standby master. Typically, only the master instance is on the master host.

  2.5 for segment instances on the segment hosts.

  If the parameter value is not set correctly, either the CPU might not be fully utilized, or query prioritization might not work as expected. For example, if the Greenplum Database cluster has fewer than one segment instance per CPU core on your segment hosts, make sure you adjust this value accordingly.
Actual CPU core utilization is based on the ability of Greenplum Database to parallelize a query and the resources required to execute the query.

Note: Any CPU core that is available to the operating system is included in the number of CPU cores. For example, virtual CPU cores are included in the number of CPU cores.

4. If you wish to view or change any of the resource management parameter values, you can use the `gpconfig` utility.

5. For example, to see the setting of a particular parameter:

   ```bash
   $ gpconfig --show gp_vmem_protect_limit
   ```

6. For example, to set one value on all segment instances and a different value on the master:

   ```bash
   $ gpconfig -c gp_resqueue_priority_cpuscores_per_segment -v 2 -m 8
   ```

7. Restart Greenplum Database to make the configuration changes effective:

   ```bash
   $ gpstop -r
   ```

Creating Resource Queues

Creating a resource queue involves giving it a name, setting an active query limit, and optionally a query priority on the resource queue. Use the `CREATE RESOURCE QUEUE` command to create new resource queues.

Creating Queues with an Active Query Limit

Resource queues with an `ACTIVE_STATEMENTS` setting limit the number of queries that can be executed by roles assigned to that queue. For example, to create a resource queue named `adhoc` with an active query limit of three:

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

This means that for all roles assigned to the `adhoc` resource queue, only three active queries can be running on the system at any given time. If this queue has three queries running, and a fourth query is submitted by a role in that queue, that query must wait until a slot is free before it can run.

Creating Queues with Memory Limits

Resource queues with a `MEMORY_LIMIT` setting control the amount of memory for all the queries submitted through the queue. The total memory should not exceed the physical memory available per-segment. Set `MEMORY_LIMIT` to 90% of memory available on a per-segment basis. For example, if a host has 48 GB of physical memory and 6 segment instances, then the memory available per segment instance is 8 GB. You can calculate the recommended `MEMORY_LIMIT` for a single queue as 0.90*8=7.2 GB. If there are multiple queues created on the system, their total memory limits must also add up to 7.2 GB.

When used in conjunction with `ACTIVE_STATEMENTS`, the default amount of memory allotted per query is: `MEMORY_LIMIT / ACTIVE_STATEMENTS`. When used in conjunction with `MAX_COST`, the default amount of memory allotted per query is: `MEMORY_LIMIT * (query_cost / MAX_COST)`. Use `MEMORY_LIMIT` in conjunction with `ACTIVE_STATEMENTS` rather than with `MAX_COST`.

For example, to create a resource queue with an active query limit of 10 and a total memory limit of 2000MB (each query will be allocated 200MB of segment host memory at execution time):

```sql
=# CREATE RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=20, MEMORY_LIMIT='2000MB');
```
The default memory allotment can be overridden on a per-query basis using the `statement_mem` server configuration parameter, provided that `MEMORY_LIMIT` or `max_statement_mem` is not exceeded. For example, to allocate more memory to a particular query:

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

As a general guideline, `MEMORY_LIMIT` for all of your resource queues should not exceed the amount of physical memory of a segment host. If workloads are staggered over multiple queues, it may be OK to oversubscribe memory allocations, keeping in mind that queries may be cancelled during execution if the segment host memory limit (`gp_vmem_protect_limit`) is exceeded.

### Setting Priority Levels

To control a resource queue's consumption of available CPU resources, an administrator can assign an appropriate priority level. When high concurrency causes contention for CPU resources, queries and statements associated with a high-priority resource queue will claim a larger share of available CPU than lower priority queries and statements.

Priority settings are created or altered using the `WITH` parameter of the commands `CREATE RESOURCE QUEUE` and `ALTER RESOURCE QUEUE`. For example, to specify priority settings for the `adhoc` and `reporting` queues, an administrator would use the following commands:

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

To create the `executive` queue with maximum priority, an administrator would use the following command:

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

When the query prioritization feature is enabled, resource queues are given a `MEDIUM` priority by default if not explicitly assigned. For more information on how priority settings are evaluated at runtime, see [How Priorities Work](#).

**Important:** In order for resource queue priority levels to be enforced on the active query workload, you must enable the query prioritization feature by setting the associated server configuration parameters. See [Configuring Resource Management](#).

### Assigning Roles (Users) to a Resource Queue

Once a resource queue is created, you must assign roles (users) to their appropriate resource queue. If roles are not explicitly assigned to a resource queue, they will go to the default resource queue, `pg_default`. The default resource queue has an active statement limit of 20, no cost limit, and a medium priority setting.

Use the `ALTER ROLE` or `CREATE ROLE` commands to assign a role to a resource queue. For example:

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

A role can only be assigned to one resource queue at any given time, so you can use the `ALTER ROLE` command to initially assign or change a role's resource queue.

Resource queues must be assigned on a user-by-user basis. If you have a role hierarchy (for example, a group-level role) then assigning a resource queue to the group does not propagate down to the users in that group.

Superusers are always exempt from resource queue limits. Superuser queries will always run regardless of the limits set on their assigned queue.
Removing a Role from a Resource Queue

All users must be assigned to a resource queue. If not explicitly assigned to a particular queue, users will go into the default resource queue, pg_default. If you wish to remove a role from a resource queue and put them in the default queue, change the role's queue assignment to none. For example:

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

Modifying Resource Queues

After a resource queue has been created, you can change or reset the queue limits using the ALTER RESOURCE QUEUE command. You can remove a resource queue using the DROP RESOURCE QUEUE command. To change the roles (users) assigned to a resource queue, Assigning Roles (Users) to a Resource Queue.

Altering a Resource Queue

The ALTER RESOURCE QUEUE command changes the limits of a resource queue. To change the limits of a resource queue, specify the new values you want for the queue. For example:

```sql
=# ALTER RESOURCE QUEUE adhoc WITH (ACTIVE_STATEMENTS=5);
=# ALTER RESOURCE QUEUE exec WITH (PRIORITY=MAX);
```

To reset active statements or memory limit to no limit, enter a value of -1. To reset the maximum query cost to no limit, enter a value of -1.0. For example:

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

You can use the ALTER RESOURCE QUEUE command to change the priority of queries associated with a resource queue. For example, to set a queue to the minimum priority level:

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

Dropping a Resource Queue

The DROP RESOURCE QUEUE command drops a resource queue. To drop a resource queue, the queue cannot have any roles assigned to it, nor can it have any statements waiting in the queue. See Removing a Role from a Resource Queue and Clearing a Waiting Statement From a Resource Queue for instructions on emptying a resource queue. To drop a resource queue:

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

Checking Resource Queue Status

Checking resource queue status involves the following tasks:

- Viewing Queued Statements and Resource Queue Status
- Viewing Resource Queue Statistics
- Viewing the Roles Assigned to a Resource Queue
- Viewing the Waiting Queries for a Resource Queue
- Clearing a Waiting Statement From a Resource Queue
- Viewing the Priority of Active Statements
- Resetting the Priority of an Active Statement
Viewing Queued Statements and Resource Queue Status

The `gp_toolkit.gp_resqueue_status` view allows administrators to see status and activity for a resource queue. It shows how many queries are waiting to run and how many queries are currently active in the system from a particular resource queue. To see the resource queues created in the system, their limit attributes, and their current status:

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

Viewing Resource Queue Statistics

If you want to track statistics and performance of resource queues over time, you can enable statistics collecting for resource queues. This is done by setting the following server configuration parameter in your `master/postgresql.conf` file:

```sql
stats_queue_level = on
```

Once this is enabled, you can use the `pg_stat_resqueues` system view to see the statistics collected on resource queue usage. Note that enabling this feature does incur slight performance overhead, as each query submitted through a resource queue must be tracked. It may be useful to enable statistics collecting on resource queues for initial diagnostics and administrative planning, and then disable the feature for continued use.

See the Statistics Collector section in the PostgreSQL documentation for more information about collecting statistics in Greenplum Database.

Viewing the Roles Assigned to a Resource Queue

To see the roles assigned to a resource queue, perform the following query of the `pg_roles` and `gp_toolkit.gp_resqueue_status` system catalog tables:

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

You may want to create a view of this query to simplify future inquiries. For example:

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

Then you can just query the view:

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

Viewing the Waiting Queries for a Resource Queue

When a slot is in use for a resource queue, it is recorded in the `pg_locks` system catalog table. This is where you can see all of the currently active and waiting queries for all resource queues. To check that statements are being queued (even statements that are not waiting), you can also use the `gp_toolkit.gp_locks_on_resqueue` view. For example:

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

If this query returns no results, then that means there are currently no statements waiting in a resource queue.
Clearing a Waiting Statement From a Resource Queue

In some cases, you may want to clear a waiting statement from a resource queue. For example, you may want to remove a query that is waiting in the queue but has not been executed yet. You may also want to stop a query that has been started if it is taking too long to execute, or if it is sitting idle in a transaction and taking up resource queue slots that are needed by other users. To do this, you must first identify the statement you want to clear, determine its process id (pid), and then, use `pg_cancel_backend` with the process id to end that process, as shown below. An optional message to the process can be passed as the second parameter, to indicate to the user why the process was cancelled.

For example, to see process information about all statements currently active or waiting in all resource queues, run the following query:

```sql
=# 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;
```

If this query returns no results, then that means there are currently no statements in a resource queue. A sample of a resource queue with two statements in it looks something like this:

<table>
<thead>
<tr>
<th>rolname</th>
<th>rsqname</th>
<th>pid</th>
<th>granted</th>
<th>current_query</th>
<th>datname</th>
</tr>
</thead>
<tbody>
<tr>
<td>sammy</td>
<td>webuser</td>
<td>31861</td>
<td>t</td>
<td>&lt;IDLE&gt; in transaction</td>
<td>namesdb</td>
</tr>
<tr>
<td>daria</td>
<td>webuser</td>
<td>31905</td>
<td>f</td>
<td>SELECT * FROM topten;</td>
<td>namesdb</td>
</tr>
</tbody>
</table>

Use this output to identify the process id (pid) of the statement you want to clear from the resource queue. To clear the statement, you would then open a terminal window (as the `gpadmin` database superuser or as root) on the master host and cancel the corresponding process. For example:

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

**Note:**

Do not use any operating system `KILL` command.

Viewing the Priority of Active Statements

The `gp_toolkit` administrative schema has a view called `gp_resq_priority_statement`, which lists all statements currently being executed and provides the priority, session ID, and other information.

This view is only available through the `gp_toolkit` administrative schema. See the Greenplum Database Reference Guide for more information.

Resetting the Priority of an Active Statement

Superusers can adjust the priority of a statement currently being executed using the built-in function `gp_adjust_priority(session_id, statement_count, priority)`. Using this function, superusers can raise or lower the priority of any query. For example:

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

To obtain the session ID and statement count parameters required by this function, superusers can use the `gp_toolkit` administrative schema view, `gp_resq_priority_statement`. From the view, use these values for the function parameters.

- The value of the `rqpsession` column for the `session_id` parameter
• The value of the `rqpcommand` column for the `statement_count` parameter
• The value of `rqppriority` column is the current priority. You can specify a string value of `MAX`, `HIGH`, `MEDIUM`, or `LOW` as the priority.

  Note: The `gp_adjust_priority()` function affects only the specified statement. Subsequent statements in the same resource queue are executed using the queue's normally assigned priority.

### Using Resource Groups

**Warning:** Resource groups are an experimental feature and are not intended for use in a production environment. Experimental features are subject to change without notice in future releases.

You can use resource groups to manage the number of active queries that may execute concurrently in your Greenplum Database cluster. With resource groups, you can also manage the amount of CPU and memory resources Greenplum allocates to each query.

When the user executes a query, Greenplum Database evaluates the query against a set of limits defined for the resource group. Greenplum Database executes the query immediately if the group’s resource limits have not yet been reached and the query does not cause the group to exceed the concurrent transaction limit. If these conditions are not met, Greenplum Database queues the query. For example, if the maximum number of concurrent transactions for the resource group has already been reached, a subsequent query is queued and must wait until other queries complete before it runs. Greenplum Database may also execute a pending query when the resource group’s concurrency and memory limits are altered to large enough values.

Within a resource group, transactions are evaluated on a first in, first out basis. Greenplum Database periodically assesses the active workload of the system, reallocating resources and starting/queuing jobs as necessary.

When you create a resource group, you provide a set of limits that determine the amount of CPU and memory resources available to transactions executed within the group. These limits are:

<table>
<thead>
<tr>
<th>Limit Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONCURRENCY</td>
<td>The maximum number of concurrent transactions, including active and idle transactions, that are permitted for this resource group.</td>
</tr>
<tr>
<td>CPU_RATE_LIMIT</td>
<td>The percentage of CPU resources available to this resource group.</td>
</tr>
<tr>
<td>MEMORY_LIMIT</td>
<td>The percentage of memory resources available to this resource group.</td>
</tr>
<tr>
<td>MEMORY_SHARED_QUOTA</td>
<td>The percentage of memory to share across transactions submitted in this resource group.</td>
</tr>
<tr>
<td>MEMORY_SPILL_RATIO</td>
<td>The memory usage threshold for memory-intensive transactions. When a transaction reaches this threshold, it spills to disk.</td>
</tr>
</tbody>
</table>

  Note: Resource limits are not enforced on `SET` and `SHOW` commands.

### Transaction Concurrency Limit

The `CONCURRENCY` limit controls the maximum number of concurrent transactions permitted for the resource group. Each resource group is logically divided into a fixed number of slots equal to the `CONCURRENCY` limit. Greenplum Database allocates these slots an equal, fixed percentage of memory resources.
The default **CONCURRENCY** limit value for a resource group is 20.

Greenplum Database queues any transactions submitted after the resource group reaches its **CONCURRENCY** limit. When a running transaction completes, Greenplum Database un-queues and executes the earliest queued transaction if sufficient memory resources exist.

**CPU Limit**

The `gp_resource_group_cpu_limit` server configuration parameter identifies the maximum percentage of system CPU resources to allocate to resource groups on each Greenplum Database segment node. The remaining CPU resources are used for the OS kernel and the Greenplum Database auxiliary daemon processes. The default `gp_resource_group_cpu_limit` value is .9 (90%).

**Note:** The default `gp_resource_group_cpu_limit` value may not leave sufficient CPU resources if you are running other workloads on your Greenplum Database cluster nodes, so be sure to adjust this server configuration parameter accordingly.

**Warning:** Avoid setting `gp_resource_group_cpu_limit` to a value higher than .9. Doing so may result in high workload queries taking near all CPU resources, potentially starving Greenplum Database auxiliary processes.

The Greenplum Database node CPU percentage is further divided equally among each segment on the Greenplum node. Each resource group reserves a percentage of the segment CPU for resource management. You identify this percentage via the `CPU_RATE_LIMIT` value you provide when you create the resource group.

The minimum `CPU_RATE_LIMIT` percentage you can specify for a resource group is 1, the maximum is 100.

The sum of `CPU_RATE_LIMITs` specified for all resource groups you define in your Greenplum Database cluster must not exceed 100.

CPU resource assignment is elastic in that Greenplum Database may allocate the CPU resources of an idle resource group to a busier one(s). In such situations, CPU resources are re-allocated to the previously idle resource group when that resource group next becomes active. If multiple resource groups are busy, they are allocated the CPU resources of any idle resource groups based on the ratio of their `CPU_RATE_LIMITs`. For example, a resource group created with a `CPU_RATE_LIMIT` of 40 will be allocated twice as much extra CPU resource as a resource group you create with a `CPU_RATE_LIMIT` of 20.

**Memory Limits**

When resource groups are enabled, memory usage is managed at the Greenplum Database node, segment, resource group, and transaction levels.

The `gp_resource_group_memory_limit` server configuration parameter identifies the maximum percentage of system memory resources to allocate to resource groups on each Greenplum Database segment node. The default `gp_resource_group_memory_limit` value is .9 (90%).

The memory resource available on a Greenplum Database node is further divided equally among each segment on the node. Each resource group reserves a percentage of the segment memory for resource management. You identify this percentage via the `MEMORY_LIMIT` value you specify when you create the resource group. The minimum `MEMORY_LIMIT` percentage you can specify for a resource group is 1, the maximum is 100.

The sum of `MEMORY_LIMITs` specified for all resource groups you define in your Greenplum Database cluster must not exceed 100.

The memory reserved by the resource group is divided into fixed and shared components. The `MEMORY_SHARED_QUOTA` value you specify when you create the resource group identifies the percentage of reserved resource group memory that may be shared among the currently running transactions. This
memory is allotted on a first-come, first-served basis. A running transaction may use none, some, or all of the MEMORY_SHARED_QUOTA.

The minimum MEMORY_SHARED_QUOTA you can specify is 0, the maximum is 100. The default MEMORY_SHARED_QUOTA is 20.

As mentioned previously, CONCURRENCY identifies the maximum number of concurrently running transactions permitted in the resource group. The fixed memory reserved by a resource group is divided into CONCURRENCY number of transaction slots. Each slot is allocated a fixed, equal amount of resource group memory. Greenplum Database guarantees this fixed memory to each transaction.
When a query's memory usage exceeds the fixed per-transaction memory usage amount, Greenplum Database allocates available resource group shared memory to the query. The maximum amount of

CREATE RESOURCE GROUP rg_sample
WITH (CONCURRENCY = 8,
     CPU_LIMIT = 30,
     MEMORY_LIMIT = 60,
     MEMORY_SHARED_QUOTA = 50)

Figure 38: Resource Group Memory Allotments

When a query's memory usage exceeds the fixed per-transaction memory usage amount, Greenplum Database allocates available resource group shared memory to the query. The maximum amount of
resource group memory available to a specific transaction slot is the sum of the transaction’s fixed memory and the full resource group shared memory allotment.

Query Operator Memory

Most query operators are non-memory-intensive; that is, during processing, Greenplum Database can hold their data in allocated memory. When memory-intensive query operators such as join and sort process more data than can be held in memory, data is spilled to disk.

The `gp_resgroup_memory_policy` server configuration parameter governs the memory allocation and distribution algorithm for all query operators. Greenplum Database supports eager-free (the default) and auto memory policies for resource groups. When you specify the auto policy, Greenplum Database uses resource group memory limits to distribute memory across query operators, allocating a fixed size of memory to non-memory-intensive operators and the rest to memory-intensive operators. When the eager_free policy is in place, Greenplum Database distributes memory among operators more optimally by re-allocating memory released by operators that have completed their processing to operators in a later query stage.

`MEMORY_SPILL_RATIO` identifies the memory usage threshold for memory-intensive operators in a transaction. When the transaction reaches this memory threshold, it spills to disk. Greenplum Database uses the `MEMORY_SPILL_RATIO` to determine the initial memory to allocate to a transaction.

The minimum `MEMORY_SPILL_RATIO` percentage you can specify for a resource group is 0. The maximum is 100. The default `MEMORY_SPILL_RATIO` is 20.

You define the `MEMORY_SPILL_RATIO` when you create a resource group. You can selectively set this limit on a per-query basis at the session level with the `memory_spill_ratio` server configuration parameter.

Using Resource Groups

Warning: Significant Greenplum Database performance degradation has been observed when enabling resource group-based workload management on RedHat 6.x, CentOS 6.x, and SuSE 11 systems. This issue is caused by a Linux cgroup kernel bug. This kernel bug has been fixed in CentOS 7.x and Red Hat 7.x systems. The performance problem is mitigated in Red Hat 6 systems that use kernel version 2.6.32-696 or higher. SuSE 11 does not have a kernel version that resolves this issue.

Prerequisite

Greenplum Database resource groups use Linux Control Groups (cgroups) to manage CPU resources. With cgroups, Greenplum isolates the CPU usage of your Greenplum processes from other processes on the node. This allows Greenplum to support CPU usage restrictions on a per-resource-group basis.

For detailed information about cgroups, refer to the Control Groups documentation for your Linux distribution.

Complete the following tasks on each node in your Greenplum Database cluster to set up cgroups for use with resource groups:

1. If you are running the Suse 11+ operating system on your Greenplum Database cluster nodes, you must enable swap accounting on each node and restart your Greenplum Database cluster. The `swapaccount` kernel boot parameter governs the swap accounting setting on Suse 11+ systems. After setting this boot parameter, you must reboot your systems. For details, refer to the Cgroup Swap Control discussion in the Suse 11 release notes. You must be the superuser or have `sudo` access to configure kernel boot parameters and reboot systems.

2. Create the Greenplum Database cgroups configuration file `/etc/cgconfig.d/gpdb.conf`. You must be the superuser or have `sudo` access to create this file:

```
sudo vi /etc/cgconfig.d/gpdb.conf
```
3. Add the following configuration information to /etc/cgconfig.d/gpdb.conf:

```plaintext
group gpdb {
    perm {
        task {
            uid = gpadmin;
            gid = gpadmin;
        }
        admin {
            uid = gpadmin;
            gid = gpadmin;
        }
    }
    cpu {
    }
    cpuacct {
    }
}
```

This content configures CPU and CPU accounting control groups managed by the gpadmin user.

4. If not already installed and running, install the Control Groups operating system package and start the cgroups service on each Greenplum Database node. The commands you run to perform these tasks will differ based on the operating system installed on the node. You must be the superuser or have sudo access to run these commands:

- **Redhat/CentOS 7.x systems:**
  ```plaintext
  sudo yum install libcgroup-tools
  sudo cgconfigparser -l /etc/cgconfig.d/gpdb.conf
  ```

- **Redhat/CentOS 6.x systems:**
  ```plaintext
  sudo yum install libcgroup
  sudo service cgconfig start
  ```

- **Suse 11+ systems:**
  ```plaintext
  sudo zypper install libcgroup-tools
  sudo cgconfigparser -l /etc/cgconfig.d/gpdb.conf
  ```

5. Identify the cgroup directory mount point for the node:

```plaintext
grep cgroup /proc/mounts
```

The first line of output identifies the cgroup mount point.

6. Verify that you set up the Greenplum Database cgroups configuration correctly by running the following commands. Replace `cgroup_mount_point` with the mount point you identified in the previous step:

```plaintext
ls -l cgroup_mount_point/cpu/gpdb
ls -l cgroup_mount_point/cpuacct/gpdb
```

If these directories exist and are owned by gpadmin:gpadmin, you have successfully configured cgroups for Greenplum Database CPU resource management.

**Procedure**

To use resource groups in your Greenplum Database cluster, you:

1. **Enable resource groups for your Greenplum Database cluster.**
2. **Create resource groups.**
3. Assign the resource groups to one or more roles.
4. Use resource management system views to monitor and manage the resource groups.

**Enabling Resource Groups**

When you install Greenplum Database, resource queues are enabled by default. To use resource groups instead of resource queues, you must set the `gp_resource_manager` server configuration parameter.

1. Set the `gp_resource_manager` server configuration parameter to the value "group":

   ```
   gpconfig -s gp_resource_manager
   gpconfig -c gp_resource_manager -v "group"
   ```

2. Restart Greenplum Database:

   ```
   gpstop
   gpstart
   ```

Once enabled, any transaction submitted by a role is directed to the resource group assigned to the role, and is governed by that resource group's concurrency, memory, and CPU limits.

Greenplum Database creates two default resource groups named `admin_group` and `default_group`. When you enable resource groups, any role that was not explicitly assigned a resource group is assigned the default group for the role's capability. SUPERUSER roles are assigned the `admin_group`, non-admin roles are assigned the group named `default_group`.

The default resource groups `admin_group` and `default_group` are created with the following resource limits:

<table>
<thead>
<tr>
<th>Limit Type</th>
<th>admin_group</th>
<th>default_group</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONCURRENCE</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>CPU_RATE_LIMIT</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>MEMORY_LIMIT</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>MEMORY_SHARED_QUOTA</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>MEMORY_SPILL_RATIO</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

**Creating Resource Groups**

When you create a resource group, you provide a name, CPU limit, and memory limit. You can optionally provide a concurrent transaction limit and memory shared quota and spill ratio. Use the `CREATE RESOURCE GROUP` command to create a new resource group.

When you create a resource group, you must provide `CPU_RATE_LIMIT` and `MEMORY_LIMIT` limit values. These limits identify the percentage of Greenplum Database resources to allocate to this resource group. For example, to create a resource group named `rgroup1` with a CPU limit of 20 and a memory limit of 25:

```
=# CREATE RESOURCE GROUP rgroup1 WITH (CPU_RATE_LIMIT=20, MEMORY_LIMIT=25);
```

The CPU limit of 20 is shared by every role to which `rgroup1` is assigned. Similarly, the memory limit of 25 is shared by every role to which `rgroup1` is assigned.

The `ALTER RESOURCE GROUP` command updates the limits of a resource group. To change the limits of a resource group, specify the new values you want for the group. For example:

```
=# ALTER RESOURCE GROUP rg_light SET CONCURRENCE 7;
=# ALTER RESOURCE GROUP exec SET MEMORY_LIMIT 25;
```
Note: You cannot set or alter the CONCURRENCY value for the admin_group to zero (0).

The DROP RESOURCE GROUP command drops a resource group. To drop a resource group, the group cannot be assigned to any role, nor can there be any transactions active or waiting in the resource group. To drop a resource group:

```sql
=## DROP RESOURCE GROUP exec;
```

**Assigning a Resource Group to a Role**

When you create a resource group, the group is available for assignment to one or more roles (users). You assign a resource group to a database role using the RESOURCE GROUP clause of the CREATE ROLE or ALTER ROLE commands. If you do not specify a resource group for a role, the role is assigned the default group for the role's capability. SUPERUSER roles are assigned the admin_group, non-admin roles are assigned the group named default_group.

Use the ALTER ROLE or CREATE ROLE commands to assign a resource group to a role. For example:

```sql
=## ALTER ROLE bill RESOURCE GROUP rg_light;
=## CREATE ROLE mary RESOURCE GROUP exec;
```

You can assign a resource group to one or more roles. If you have defined a role hierarchy, assigning a resource group to a parent role does not propagate down to the members of that role group.

If you wish to remove a resource group assignment from a role and assign the role the default group, change the role's group name assignment to NONE. For example:

```sql
=## ALTER ROLE mary RESOURCE GROUP NONE;
```

**Monitoring Resource Group Status**

Monitoring the status of your resource groups and queries may involve the following tasks:

- Viewing Resource Group Limits
- Viewing Resource Group Query Status and CPU/Memory Usage
- Viewing the Resource Group Assigned to a Role
- Viewing a Resource Group's Running and Pending Queries
- Cancelling a Running or Queued Transaction in a Resource Group

**Viewing Resource Group Limits**

The gp_resgroup_config gp_toolkit system view displays the current and proposed limits for a resource group. The proposed limit differs from the current limit when you alter the limit but the new value can not be immediately applied. To view the limits of all resource groups:

```sql
=## SELECT * FROM gp_toolkit.gp_resgroup_config;
```

**Viewing Resource Group Query Status and CPU/Memory Usage**

The gp_resgroup_status gp_toolkit system view enables you to view the status and activity of a resource group. The view displays the number of running and queued transactions. It also displays the real-time CPU and memory usage of the resource group. To view this information:

```sql
=## SELECT * FROM gp_toolkit.gp_resgroup_status;
```
Viewing the Resource Group Assigned to a Role

To view the resource group-to-role assignments, perform the following query on the `pg_roles` and `pg_resgroup` system catalog tables:

```sql
=# SELECT rolname, rsgname FROM pg_roles, pg_resgroup
    WHERE pg_roles.rolresgroup=pg_resgroup.oid;
```

Viewing a Resource Group's Running and Pending Queries

To view a resource group's running queries, pending queries, and how long the pending queries have been queued, examine the `pg_stat_activity` system catalog table:

```sql
=# SELECT current_query, waiting, rsgname, rsgqueueduration
    FROM pg_stat_activity;
```

Cancelling a Running or Queued Transaction in a Resource Group

There may be cases when you want to cancel a running or queued transaction in a resource group. For example, you may want to remove a query that is waiting in the resource group queue but has not yet been executed. Or, you may want to stop a running query that is taking too long to execute, or one that is sitting idle in a transaction and taking up resource group transaction slots that are needed by other users.

To cancel a running or queued transaction, you must first determine the process id (pid) associated with the transaction. Once you have obtained the process id, you can invoke `pg_cancel_backend()` to end that process, as shown below.

For example, to view the process information associated with all statements currently active or waiting in all resource groups, run the following query. If the query returns no results, then there are no running or queued transactions in any resource group.

```sql
=# 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.usename=pg_roles.rolname;
```

Sample partial query output:

<table>
<thead>
<tr>
<th>rolname</th>
<th>rsgname</th>
<th>procpid</th>
<th>waiting</th>
<th>current_query</th>
<th>datname</th>
</tr>
</thead>
<tbody>
<tr>
<td>sammy</td>
<td>rg_light</td>
<td>31861</td>
<td>f</td>
<td>&lt;IDLE&gt; in transaction</td>
<td>testdb</td>
</tr>
<tr>
<td>billy</td>
<td>rg_light</td>
<td>31905</td>
<td>t</td>
<td>SELECT * FROM topten;</td>
<td>testdb</td>
</tr>
</tbody>
</table>

Use this output to identify the process id (procpid) of the transaction you want to cancel, and then cancel the process. For example, to cancel the pending query identified in the sample output above:

```sql
=# SELECT pg_cancel_backend(31905);
```

You can provide an optional message in a second argument to `pg_cancel_backend()` to indicate to the user why the process was cancelled.

Note:

Do not use an operating system `KILL` command to cancel any Greenplum Database process.
Investigating a Performance Problem

This section provides guidelines for identifying and troubleshooting performance problems in a Greenplum Database system.

This topic lists steps you can take to help identify the cause of a performance problem. If the problem affects a particular workload or query, you can focus on tuning that particular workload. If the performance problem is system-wide, then hardware problems, system failures, or resource contention may be the cause.

Checking System State

Use the `gpstate` utility to identify failed segments. A Greenplum Database system will incur performance degradation when segment instances are down because other hosts must pick up the processing responsibilities of the down segments.

Failed segments can indicate a hardware failure, such as a failed disk drive or network card. Greenplum Database provides the hardware verification tool `gpcheckperf` to help identify the segment hosts with hardware issues.

Checking Database Activity

- Checking for Active Sessions (Workload)
- Checking for Locks (Contention)
- Checking Query Status and System Utilization

Checking for Active Sessions (Workload)

The `pg_stat_activity` system catalog view shows one row per server process; it shows the database OID, database name, process ID, user OID, user name, current query, time at which the current query began execution, time at which the process was started, client address, and port number. To obtain the most information about the current system workload, query this view as the database superuser. For example:

```
SELECT * FROM pg_stat_activity;
```

Note the information does not update instantaneously.

Checking for Locks (Contention)

The `pg_locks` system catalog view allows you to see information about outstanding locks. If a transaction is holding a lock on an object, any other queries must wait for that lock to be released before they can continue. This may appear to the user as if a query is hanging.

Examine `pg_locks` for ungranted locks to help identify contention between database client sessions. `pg_locks` provides a global view of all locks in the database system, not only those relevant to the current database. You can join its relation column against `pg_class.oid` to identify locked relations (such as tables), but this works correctly only for relations in the current database. You can join the `pid` column to the `pg_stat_activity.procpid` to see more information about the session holding or waiting to hold a lock. For example:

```
SELECT locktype, database, c.relname, l.relation, l.transactionid, l.pid, l.mode, l.granted, a.current_query
FROM pg_locks l, pg_class c, pg_stat_activity a
WHERE l.relation=c.oid AND l.pid=a.procpid
ORDER BY c.relname;
```
If you use resource queues, queries that are waiting in a queue will also show in \textit{pg\_locks}. To see how many queries are waiting to run from a resource queue, use the \textit{gp\_resqueue\_status} system catalog view.

For example:

\begin{verbatim}
SELECT * FROM gp_toolkit.gp_resqueue_status;
\end{verbatim}

\section*{Checking Query Status and System Utilization}

You can use system monitoring utilities such as \texttt{ps}, \texttt{top}, \texttt{iostat}, \texttt{vmstat}, \texttt{netstat} and so on to monitor database activity on the hosts in your Greenplum Database array. These tools can help identify Greenplum Database processes (\texttt{postgres} processes) currently running on the system and the most resource intensive tasks with regards to CPU, memory, disk I/O, or network activity. Look at these system statistics to identify queries that degrade database performance by overloading the system and consuming excessive resources. Greenplum Database's management tool \texttt{gppssh} allows you to run these system monitoring commands on several hosts simultaneously.

You can create and use the Greenplum Database \texttt{session\_level\_memory\_consumption} view that provides information about the current memory utilization and idle time for sessions that are running queries on Greenplum Database. For information about the view, see \texttt{Viewing Session Memory Usage Information}.

You can enable a dedicated database, \texttt{gpperfmon}, in which data collection agents running on each segment host save query and system utilization metrics. Refer to the \texttt{gpermon\_install} management utility reference in the \texttt{Greenplum Database Management Utility Reference Guide} for help creating the \texttt{gpperfmon} database and managing the agents. See documentation for the tables and views in the \texttt{gpperfmon} database in the \texttt{Greenplum Database Reference Guide}.

The optional Greenplum Command Center web-based user interface graphically displays query and system utilization metrics saved in the \texttt{gpperfmon} database. See the \texttt{Greenplum Command Center Documentation} web site for procedures to enable Greenplum Command Center.

\section*{Troubleshooting Problem Queries}

If a query performs poorly, look at its query plan to help identify problems. The \texttt{EXPLAIN} command shows the query plan for a given query. See \texttt{Query Profiling} for more information about reading query plans and identifying problems.

When an out of memory event occurs during query execution, the Greenplum Database memory accounting framework reports detailed memory consumption of every query running at the time of the event. The information is written to the Greenplum Database segment logs.

\section*{Investigating Error Messages}

Greenplum Database log messages are written to files in the \texttt{pg\_log} directory within the master's or segment's data directory. Because the master log file contains the most information, you should always check it first. Log files roll over daily and use the naming convention: \texttt{gpdb-YYYY-MM-DD_hhmss.csv}.

To locate the log files on the master host:

\begin{verbatim}
$ cd $MASTER\_DATA\_DIRECTORY/pg\_log
\end{verbatim}

Log lines have the format of:

\begin{verbatim}
timestamp | user | database | statement\_id | con\#cmd\# |
\end{verbatim}

You may want to focus your search for \texttt{WARNING}, \texttt{ERROR}, \texttt{FATAL} or \texttt{PANIC} log level messages. You can use the Greenplum utility \texttt{gplogfilter} to search through Greenplum Database log files. For example,
when you run the following command on the master host, it checks for problem log messages in the standard logging locations:

```
$ gplogfilter -t
```

To search for related log entries in the segment log files, you can run `gplogfilter` on the segment hosts using `gpssh`. You can identify corresponding log entries by the `statement_id` or `con#` (session identifier). For example, to search for log messages in the segment log files containing the string `con6` and save output to a file:

```
gpssh -f seg_hosts_file -e 'source /usr/local/greenplum-db/greenplum_path.sh ; gplogfilter -f con6 /gpdata/*/pg_log/gpdb*.csv' > seglog.out
```

**Gathering Information for Pivotal Customer Support**

The Greenplum Magic Tool (GPMT) utility can run diagnostics and collect information from a Greenplum Database system. You can then send the information to Pivotal Customer Support to aid the diagnosis of Greenplum Database errors or system failures.

The GPMT utility is available from the [Pivotal Knowledge Base](https://knowledge.pivotal.io) on the [GPMT page](https://knowledge.pivotal.io).
Chapter 9

Greenplum Database Security Configuration Guide

This guide describes how to secure a Greenplum Database system. The guide assumes knowledge of Linux/UNIX system administration and database management systems. Familiarity with structured query language (SQL) is helpful.

Because Greenplum Database is based on PostgreSQL8.3.23, this guide assumes some familiarity with PostgreSQL. References to PostgreSQL documentation are provided throughout this guide for features that are similar to those in Greenplum Database.

This information is intended for system administrators responsible for administering a Greenplum Database system.
Securing the Database

Introduces Greenplum Database security topics.

The intent of security configuration is to configure the Greenplum Database server to eliminate as many security vulnerabilities as possible. This guide provides a baseline for minimum security requirements, and is supplemented by additional security documentation.

The essential security requirements fall into the following categories:

- **Authentication** covers the mechanisms that are supported and that can be used by the Greenplum database server to establish the identity of a client application.
- **Authorization** pertains to the privilege and permission models used by the database to authorize client access.
- **Auditing**, or log settings, covers the logging options available in Greenplum Database to track successful or failed user actions.
- **Data Encryption** addresses the encryption capabilities that are available for protecting data at rest and data in transit. This includes the security certifications that are relevant to the Greenplum Database.

Accessing a Kerberized Hadoop Cluster

Greenplum Database can read or write external tables in a Hadoop file system. If the Hadoop cluster is secured with Kerberos ("Kerberized"), Greenplum Database must be configured to allow external table owners to authenticate with Kerberos. See Enabling gphdfs Authentication with a Kerberos-secured Hadoop Cluster for the steps to perform this setup.

Platform Hardening

Platform hardening involves assessing and minimizing system vulnerability by following best practices and enforcing federal security standards. Hardening the product is based on the US Department of Defense (DoD) guidelines Security Template Implementation Guides (STIG). Hardening removes unnecessary packages, disables services that are not required, sets up restrictive file and directory permissions, removes unowned files and directories, performs authentication for single-user mode, and provides options for end users to configure the package to be compliant to the latest STIGs.
Greenplum Database Ports and Protocols

Lists network ports and protocols used within the Greenplum cluster.

Greenplum Database clients connect with TCP to the Greenplum master instance at the client connection port, 5432 by default. The listen port can be reconfigured in the `postgresql.conf` configuration file. Client connections use the PostgreSQL libpq API. The `psql` command-line interface, several Greenplum utilities, and language-specific programming APIs all either use the libpq library directly or implement the libpq protocol internally.

Each segment instance also has a client connection port, used solely by the master instance to coordinate database operations with the segments. The `gpstate -p` command, executed on the Greenplum master, lists the port assignments for the Greenplum master and the primary segments and mirrors. For example:

```
[gpm] ~$ gpstate -p
Version: 'postgres (Greenplum Database) 4.3.6.0 build 62994'
Version: 'PostgreSQL 8.2.15 (Greenplum Database 4.3.6.0 build 62994) on
x86_64-unknown-linux-gnu, compiled by GCC gcc (GCC) 4.4.2 compiled on Jul
24 2015 11:35:08'
details from master...
instance /data/master/gpseg-1 port = 5432
assignments
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:-   primary/gpseg0  40000
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:-   gpseg0  50000
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:-   primary/gpseg1  40000
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:-   gpseg1  50001
```

Additional Greenplum Database network connections are created for features such as standby replication, segment mirroring, statistics collection, and data exchange between segments. Some persistent connections are established when the database starts up and other transient connections are created during operations such as query execution. Transient connections for query execution processes, data movement, and statistics collection use available ports in the range 1025 to 65535 with both TCP and UDP protocols.
Some add-on products and services that work with Greenplum Database have additional networking requirements. The following table lists ports and protocols used within the Greenplum cluster, and includes services and applications that integrate with Greenplum Database.

**Table 65: Greenplum Database Ports and Protocols**

<table>
<thead>
<tr>
<th>Service</th>
<th>Protocol/Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master SQL client connection</td>
<td>TCP 5432, libpq</td>
<td>SQL client connection port on the Greenplum master host. Supports clients using the PostgreSQL libpq API. Configurable.</td>
</tr>
<tr>
<td>Segment SQL client connection</td>
<td>varies, libpq</td>
<td>The SQL client connection port for a segment instance. Each primary and mirror segment on a host must have a unique port. Ports are assigned when the Greenplum system is initialized or expanded. The gp_segment_configuration system catalog records port numbers for each segment in the port column. Run <code>gpstate -p</code> to view the ports in use.</td>
</tr>
<tr>
<td>Segment mirroring port</td>
<td>varies, libpq</td>
<td>The port where a segment receives mirrored blocks from its primary. The port is assigned when the mirror is set up. The port number is stored in the gp_segment_configuration system catalog in the mirror_port column.</td>
</tr>
<tr>
<td>Greenplum Database Interconnect</td>
<td>UDP 1025-65535, dynamically allocated</td>
<td>The Interconnect transports database tuples between Greenplum segments during query execution.</td>
</tr>
<tr>
<td>Standby master client listener</td>
<td>TCP 5432, libpq</td>
<td>SQL client connection port on the standby master host. Usually the same as the master client connection port. Configure with the gpinitstandby utility -P option.</td>
</tr>
<tr>
<td>Standby master replicator</td>
<td>TCP 1025-65535, gpsyncmaster</td>
<td>The gpsyncmaster process on the master host establishes a connection to the secondary master host to replicate the master's log to the standby master.</td>
</tr>
<tr>
<td>Greenplum Control Center (GPCC)</td>
<td>TCP 28080, HTTP/HTTPS</td>
<td>Default listen port for the GPCC console web server, which is usually installed on the master host. Configured in the lighttpd.conf file in the GPCC instance.</td>
</tr>
<tr>
<td>Greenplum database file load and transfer utilities: gpfdist, gpload, gptransfer</td>
<td>TCP 8080, HTTP TCP 9000, HTTPS</td>
<td>The gpfdist file serving utility can run on Greenplum hosts or external hosts. Specify the connection port with the -p option when starting the server. The gpload and gptransfer utilities run one or more instances of gpfdist with ports or port ranges specified in a configuration file.</td>
</tr>
<tr>
<td>GPCC agents</td>
<td>TCP 8888</td>
<td>Connection port for GPCC agents executing on each Greenplum host. Configure by setting the gpperfmon_port configuration variable in postgresql.conf on master and segment hosts.</td>
</tr>
<tr>
<td>Service</td>
<td>Protocol/Port</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Backup completion notification</td>
<td>TCP 25, TCP 587, SMTP</td>
<td>The gpcrondump backup utility can optionally send email to a list of email addresses at completion of a backup. The SMTP service must be enabled on the Greenplum master host.</td>
</tr>
<tr>
<td>Greenplum Database secure shell (SSH): gpssh, gpscp, gpssh-exkeys, gppkg, gpseginstall</td>
<td>TCP 22, SSH</td>
<td>Many Greenplum utilities use scp and ssh to transfer files between hosts and manage the Greenplum system within the cluster.</td>
</tr>
<tr>
<td>gphdfs</td>
<td>TCP 8020</td>
<td>The gphdfs protocol allows access to data in a Hadoop file system via Greenplum external tables. The URL in the <code>LOCATION</code> clause of the CREATE EXTERNAL TABLE command specifies the host address and port number for the Hadoop namenode service.</td>
</tr>
<tr>
<td>Greenplum Workload Manager (GP-WLM)</td>
<td>TCP 4369, epmd</td>
<td>Erlang port mapper (epmd) allows nodes in the cluster to resolve node names.</td>
</tr>
<tr>
<td></td>
<td>TCP 25672</td>
<td>rabbitmq clustering</td>
</tr>
<tr>
<td></td>
<td>TCP 7777</td>
<td>rabbitmq main port</td>
</tr>
<tr>
<td>EMC Data Domain and DD Boost</td>
<td>TCP/UDP 111, NFS portmapper</td>
<td>Used to assign a random port for the mountd service used by NFS and DD Boost. The mountd service port can be statically assigned on the Data Domain server.</td>
</tr>
<tr>
<td></td>
<td>TCP 2052</td>
<td>Main port used by NFS mountd. This port can be set on the Data Domain system using the <code>nfs set mountd-port</code> command.</td>
</tr>
<tr>
<td></td>
<td>TCP 2049, NFS</td>
<td>Main port used by NFS. This port can be configured using the <code>nfs set server-port</code> command on the Data Domain server.</td>
</tr>
<tr>
<td></td>
<td>TCP 2051, replication</td>
<td>Used when replication is configured on the Data Domain system. This port can be configured using the <code>replication modify</code> command on the Data Domain server.</td>
</tr>
<tr>
<td>Symantec NetBackup</td>
<td>TCP/UDP 1556, veritas-pbx</td>
<td>The Symantec NetBackup client network port.</td>
</tr>
<tr>
<td></td>
<td>TCP 13724, vnetd</td>
<td>Symantec NetBackup vnetd communication port.</td>
</tr>
<tr>
<td>Service</td>
<td>Protocol/Port</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Pgbouncer connection pooler</td>
<td>TCP, libpq</td>
<td>The pgbouncer connection pooler runs between libpq clients and Greenplum (or PostgreSQL) databases. It can be run on the Greenplum master host, but running it on a host outside of the Greenplum cluster is recommended. When it runs on a separate host, pgbouncer can act as a warm standby mechanism for the Greenplum master host, switching to the Greenplum standby host without requiring clients to reconfigure. Set the client connection port and the Greenplum master host address and port in the <code>pgbouncer.ini</code> configuration file.</td>
</tr>
<tr>
<td>stunnel SSL proxy</td>
<td>TCP, ssh, libpq</td>
<td>A stunnel SSL proxy can be used to add SSL support for database clients accessing the database through a pgbouncer connection pool. A secure tunnel can be set up by setting up stunnel on the client and the pgbouncer host. Newer versions of stunnel that support encrypted libpq connections only require stunnel on the pgbouncer host. The stunnel proxy's connection ports and the pgbouncer host and port are specified in the <code>stunnel.conf</code> configuration file.</td>
</tr>
</tbody>
</table>
Configuring Client Authentication

Describes the available methods for authenticating Greenplum Database clients.

When a Greenplum Database system is first initialized, the system contains one predefined superuser role. This role will have the same name as the operating system user who initialized the Greenplum Database system. This role is referred to as gpadmin. By default, the system is configured to only allow local connections to the database from the gpadmin role. If you want to allow any other roles to connect, or if you want to allow connections from remote hosts, you have to configure Greenplum Database to allow such connections. This section explains how to configure client connections and authentication to Greenplum Database.

- Allowing Connections to Greenplum Database
- Editing the pg_hba.conf File
- Authentication Methods
- Limiting Concurrent Connections
- Encrypting Client/Server Connections

Allowing Connections to Greenplum Database

Client access and authentication is controlled by a configuration file named pg_hba.conf (the standard PostgreSQL host-based authentication file). For detailed information about this file, see The pg_hba.conf File in the PostgreSQL documentation.

In Greenplum Database, the pg_hba.conf file of the master instance controls client access and authentication to your Greenplum system. The segments also have pg_hba.conf files, but these are already correctly configured to only allow client connections from the master host. The segments never accept outside client connections, so there is no need to alter the pg_hba.conf file on segments.

The general format of the pg_hba.conf file is a set of records, one per line. Blank lines are ignored, as is any text after a # comment character. A record is made up of a number of fields which are separated by spaces and/or tabs. Fields can contain white space if the field value is quoted. Records cannot be continued across lines. Each remote client access record is in this format:

<table>
<thead>
<tr>
<th>host</th>
<th>database</th>
<th>role</th>
<th>CIDR-address</th>
<th>authentication-method</th>
</tr>
</thead>
</table>

Each UNIX-domain socket access record is in this format:

<table>
<thead>
<tr>
<th>local</th>
<th>database</th>
<th>role</th>
<th>authentication-method</th>
</tr>
</thead>
</table>

The meaning of the pg_hba_conf fields is as follows:

local

Matches connection attempts using UNIX-domain sockets. Without a record of this type, UNIX-domain socket connections are disallowed.

host

Matches connection attempts made using TCP/IP. Remote TCP/IP connections will not be possible unless the server is started with an appropriate value for the listen_addresses server configuration parameter.

hostssl

Matches connection attempts made using TCP/IP, but only when the connection is made with SSL encryption. SSL must be enabled at server start time by setting the ssl configuration parameter. Requires SSL authentication be configured in postgresql.conf. See Configuring postgresql.conf for SSL Authentication.
hostnossl
Matches connection attempts made over TCP/IP that do not use SSL. Requires SSL authentication be configured in `postgresql.conf`. See `Configuring postgresql.conf for SSL Authentication`.

database
Specifies which database names this record matches. The value all specifies that it matches all databases. Multiple database names can be supplied by separating them with commas. A separate file containing database names can be specified by preceding the file name with @.

role
Specifies which database role names this record matches. The value all specifies that it matches all roles. If the specified role is a group and you want all members of that group to be included, precede the role name with a +. Multiple role names can be supplied by separating them with commas. A separate file containing role names can be specified by preceding the file name with @.

CIDR-address
Specifies the client machine IP address range that this record matches. It contains an IP address in standard dotted decimal notation and a CIDR mask length. IP addresses can only be specified numerically, not as domain or host names. The mask length indicates the number of high-order bits of the client IP address that must match. Bits to the right of this must be zero in the given IP address. There must not be any white space between the IP address, the /, and the CIDR mask length.

Typical examples of a CIDR-address are 172.20.143.89/32 for a single host, or 172.20.143.0/24 for a small network, or 10.6.0.0/16 for a larger one. To specify a single host, use a CIDR mask of 32 for IPv4 or 128 for IPv6. In a network address, do not omit trailing zeroes.

IP-address
IP-mask
These fields can be used as an alternative to the CIDR-address notation. Instead of specifying the mask length, the actual mask is specified in a separate column. For example, 255.0.0.0 represents an IPv4 CIDR mask length of 8, and 255.255.255.255 represents a CIDR mask length of 32. These fields only apply to host, hostssl, and hostnossl records.

authentication-method
Specifies the authentication method to use when connecting. See `Authentication Methods` for more details.

Editing the pg_hba.conf File

This example shows how to edit the `pg_hba.conf` file of the master to allow remote client access to all databases from all roles using encrypted password authentication.

Note: For a more secure system, consider removing all connections that use trust authentication from your master `pg_hba.conf`. Trust authentication means the role is granted access without any authentication, therefore bypassing all security. Replace trust entries with ident authentication if your system has an ident service available.

To edit `pg_hba.conf`:

1. Open the file `$MASTER_DATA_DIRECTORY/pg_hba.conf` in a text editor.
2. Add a line to the file for each type of connection you want to allow. Records are read sequentially, so the order of the records is significant. Typically, earlier records will have tight connection match
parameters and weaker authentication methods, while later records will have looser match parameters and stronger authentication methods. For example:

```text
# allow the gpadmin user local access to all databases
# using ident authentication
local   all   gpadmin   ident         sameuser
host   all   gpadmin   127.0.0.1/32 ident
host   all   gpadmin   ::1/128       ident

# allow the 'dba' role access to any database from any
# host with IP address 192.168.x.x and use md5 encrypted
# passwords to authenticate the user
# Note that to use SHA-256 encryption, replace md5 with
# password in the line below
host   all   dba   192.168.0.0/32 md5
```

**Authentication Methods**

- **Basic Authentication**
- **Kerberos Authentication**
- **LDAP Authentication**
- **SSL Client Authentication**
- **PAM Based Authentication**
- **Radius Authentication**

**Basic Authentication**

The following basic authentication methods are supported:

**Password or MD5**

Requires clients to provide a password, one of either:

- **Md5** – password transmitted as an MD5 hash.
- **Password** – A password transmitted in clear text. Always use SSL connections to prevent password sniffing during transit. This is configurable, see "Encrypting Passwords" in the *Greenplum Database Administrator Guide* for more information.

**Reject**

Reject the connections with the matching parameters. You should typically use this to restrict access from specific hosts or insecure connections.

**Ident**

Authenticates based on the client’s operating system user name. You should only use this for local connections.

Following are some sample *pg_hba.conf* basic authentication entries:

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

**Kerberos Authentication**

You can authenticate against a Kerberos server (RFC 2743, 1964).

The format for Kerberos authentication in the *pg_hba.conf* file is:

```
servicename/hostname@realm
```

The following options may be added to the entry:
Map
- Map system and database users.

Include_realm
- Option to specify realm name included in the system-user name in the ident map file.

Krb_realm
- Specify the realm name for matching the principals.

Krb_server_hostname
- The hostname of the service principal.

Following is an example pg_hba.conf entry for Kerberos:

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

The following Kerberos server settings are specified in postgresql.conf:

- krb_server_key file
  - Sets the location of the Kerberos server key file.

- krb_srvname string
  - Kerberos service name.

- krb_caseins_users boolean
  - Case-sensitivity. The default is off.

The following client setting is specified as a connection parameter:

- Krbsrvname
  - The Kerberos service name to use for authentication.

### LDAP Authentication

You can authenticate against an LDAP directory.

- LDAPS and LDAP over TLS options encrypt the connection to the LDAP server.
- The connection from the client to the server is not encrypted unless SSL is enabled. Configure client connections to use SSL to encrypt connections from the client.
- To configure or customize LDAP settings, set the LDAPCONF environment variable with the path to the ldap.conf file and add this to the greenplum_path.sh script.

Following are the recommended steps for configuring your system for LDAP authentication:

1. Set up the LDAP server with the database users/roles to be authenticated via LDAP.
2. On the database:
   a. Verify that the database users to be authenticated via LDAP exist on the database. LDAP is only used for verifying username/password pairs, so the roles should exist in the database.
   b. Update the pg_hba.conf file in the $MASTER_DATA_DIRECTORY to use LDAP as the authentication method for the respective users. Note that the first entry to match the user/role in the pg_hba.conf file will be used as the authentication mechanism, so the position of the entry in the file is important.
   c. Reload the server for the pg_hba.conf configuration settings to take effect (gpstop -u).

Specify the following parameter auth-options.

- ldapserver
  - Names or IP addresses of LDAP servers to connect to. Multiple servers may be specified, separated by spaces.

- ldapprefix
String to prepend to the user name when forming the DN to bind as, when doing simple bind authentication.

**ldapsuffix**

String to append to the user name when forming the DN to bind as, when doing simple bind authentication.

**ldapport**

Port number on LDAP server to connect to. If no port is specified, the LDAP library's default port setting will be used.

**ldaptls**

Set to 1 to make the connection between PostgreSQL and the LDAP server use TLS encryption. Note that this only encrypts the traffic to the LDAP server — the connection to the client will still be unencrypted unless SSL is used.

**ldapbasedn**

Root DN to begin the search for the user in, when doing search+bind authentication.

**ldapbinddn**

DN of user to bind to the directory with to perform the search when doing search+bind authentication.

**ldapbindpasswd**

Password for user to bind to the directory with to perform the search when doing search +bind authentication.

**ldapsearchattribute**

Attribute to match against the user name in the search when doing search+bind authentication.

Example:

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

Following are sample `pg_hba.conf` file entries for LDAP authentication:

```
host all testuser 0.0.0.0/0 ldap ldap
  ldapsuffix="ldapserver.greenplum.com" ldapprefix="cn="
  ldapprefix="", ou=people, dc=greenplum, dc=com"
hostssl all ldapmodel 0.0.0.0/0 ldap
  ldapsuffix="", ou=people, dc=greenplum, dc=com"
```

**SSL Client Authentication**

SSL authentication compares the Common Name (cn) attribute of an SSL certificate provided by the connecting client during the SSL handshake to the requested database user name. The database user should exist in the database. A map file can be used for mapping between system and database user names.

**SSL Authentication Parameters**

Authentication method:

- Cert

  Authentication options:

  - Hostssl
Connection type must be hostssl.

map=\texttt{mapping}

mapping.

This is specified in the \texttt{pg\_ident.conf}, or in the file specified in the \texttt{ident\_file server}
setting.

Following are sample \texttt{pg\_hba.conf} entries for SSL client authentication:

\begin{verbatim}
Hostssl testdb certuser 192.168.0.0/16 cert
Hostssl testdb all 192.168.0.0/16 cert map=gpuser
\end{verbatim}

\section*{OpenSSL Configuration}

Greenplum Database reads the OpenSSL configuration file specified in $\texttt{GP\_HOME/etc/openssl.cnf}$
by default. You can make changes to the default configuration for OpenSSL by modifying or updating this
file and restarting the server.

\section*{Creating a Self-Signed Certificate}

A self-signed certificate can be used for testing, but a certificate signed by a certificate authority (CA)
(either one of the global CAs or a local one) should be used in production so that clients can verify the
server's identity. If all the clients are local to the organization, using a local CA is recommended.

To create a self-signed certificate for the server:

1. Enter the following \texttt{openssl} command:

\begin{verbatim}
openssl req -new -text -out server.req
\end{verbatim}

2. Enter the requested information at the prompts.

Make sure you enter the local host name for the Common Name. The challenge password can be left
blank.

3. The program generates a key that is passphrase-protected; it does not accept a passphrase that is less
than four characters long. To remove the passphrase (and you must if you want automatic start-up of
the server), run the following command:

\begin{verbatim}
openssl rsa -in privkey.pem -out server.key rm privkey.pem
\end{verbatim}

4. Enter the old passphrase to unlock the existing key. Then run the following command:

\begin{verbatim}
openssl req -x509 -in server.req -text -key server.key -out server.crt
\end{verbatim}

This turns the certificate into a self-signed certificate and copies the key and certificate to where the
server will look for them.

5. Finally, run the following command:

\begin{verbatim}
chmod og-rwx server.key
\end{verbatim}

For more details on how to create your server private key and certificate, refer to the OpenSSL
documentation.

\section*{Configuring postgresql.conf for SSL Authentication}

The following Server settings need to be specified in the \texttt{postgresql.conf} configuration file:

- \texttt{ssl boolean}. Enables SSL connections.
- \texttt{ssl\_renegotiation\_limit integer}. Specifies the data limit before key renegotiation.
• `ssl_ciphers string`. Lists SSL ciphers that are allowed.

The following SSL server files can be found in the Master Data Directory:
• `server.crt`. Server certificate.
• `server.key`. Server private key.
• `root.crt`. Trusted certificate authorities.
• `root.crl`. Certificates revoked by certificate authorities.

**Configuring the SSL Client Connection**

SSL options:

- **require**
  Only use SSL connection. If a root CA file is present, verify the certificate in the same way as if `verify-ca` was specified.

- **verify-ca**
  Only use an SSL connection. Verify that the server certificate is issued by a trusted CA.

- **verify-full**
  Only use an SSL connection. Verify that the server certificate is issued by a trusted CA and that the server host name matches that in the certificate.

- **sslcert**
  The file name of the client SSL certificate. The default is `~/.postgresql/.postgresql.crt`.

- **sslkey**
  The secret key used for the client certificate. The default is `~/.postgresql/.postgresql.key`.

- **sslrootcert**
  The name of a file containing SSL Certificate Authority certificate(s). The default is `~/.postgresql/root.crt`.

- **sslcrl**
  The name of the SSL certificate revocation list. The default is `~/.postgresql/root.crl`.

The client connection parameters can be set using the following environment variables:

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

**PAM Based Authentication**

"PAM" (Pluggable Authentication Modules) is used to validate username/password pairs, similar to basic authentication. PAM authentication only works if the users already exist in the database.

**Parameters**

- **pamservice**
  The default PAM service is `postgresql`. Note that if PAM is set up to read `/etc/shadow`, authentication will fail because the PostgreSQL server is started by a non-root user.
Following are sample `pg_hba.conf` entries for PAM client authentication:

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

**Radius Authentication**

RADIUS (Remote Authentication Dial In User Service) authentication works by sending an Access Request message of type 'Authenticate Only' to a configured RADIUS server. It includes parameters for user name, password (encrypted), and the Network Access Server (NAS) Identifier. The request is encrypted using the shared secret specified in the `radiussecret` option. The RADIUS server responds with either Access Accept or Access Reject.

**Note:** RADIUS accounting is not supported.

RADIUS authentication only works if the users already exist in the database.

The RADIUS encryption vector requires SSL to be enabled in order to be cryptographically strong.

**RADIUS Authentication Options**

- `radiusserver`  
  The name of the RADIUS server.

- `radiussecret`  
  The RADIUS shared secret.

- `radiusport`  
  The port to connect to on the RADIUS server.

- `radiusidentifier`  
  NAS identifier in RADIUS requests.

Following are sample `pg_hba.conf` entries for RADIUS client authentication:

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

**Limiting Concurrent Connections**

To limit the number of active concurrent sessions to your Greenplum Database system, you can configure the `max_connections` server configuration parameter. This is a local parameter, meaning that you must set it in the `postgresql.conf` file of the master, the standby master, and each segment instance (primary and mirror). The value of `max_connections` on segments must be 5-10 times the value on the master.

When you set `max_connections`, you must also set the dependent parameter `max_prepared_transactions`. This value must be at least as large as the value of `max_connections` on the master, and segment instances should be set to the same value as the master.

In `$MASTER_DATA_DIRECTORY/postgresql.conf` (including standby master):

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

In `SEGMENT_DATA_DIRECTORY/postgresql.conf` for all segment instances:

```plaintext
max_connections=500
max_prepared_transactions=100
```
Note: Raising the values of these parameters may cause Greenplum Database to request more shared memory. To mitigate this effect, consider decreasing other memory-related parameters such as `gp_cached_segworkers_threshold`.

To change the number of allowed connections:

1. Stop your Greenplum Database system:

   ```
   $ gpstop
   ```

2. On the master host, edit `$MASTER_DATA_DIRECTORY/postgresql.conf` and change the following two parameters:

   - `max_connections` – the number of active user sessions you want to allow plus the number of `superuser_reserved_connections`.
   - `max_prepared_transactions` – must be greater than or equal to `max_connections`.

3. On each segment instance, edit `$SEGMENT_DATA_DIRECTORY/postgresql.conf` and change the following two parameters:

   - `max_connections` – must be 5-10 times the value on the master.
   - `max_prepared_transactions` – must be equal to the value on the master.

4. Restart your Greenplum Database system:

   ```
   $ gpstart
   ```

**Encrypting Client/Server Connections**

Greenplum Database has native support for SSL connections between the client and the master server. SSL connections prevent third parties from snooping on the packets, and also prevent man-in-the-middle attacks. SSL should be used whenever the client connection goes through an insecure link, and must be used whenever client certificate authentication is used.

Note: For information about encrypting data between the `gpfdist` server and Greenplum Database segment hosts, see *Encrypting gpfdist Connections*.

To enable SSL requires that OpenSSL be installed on both the client and the master server systems. Greenplum can be started with SSL enabled by setting the server configuration parameter `ssl=on` in the master `postgresql.conf`. When starting in SSL mode, the server will look for the files `server.key` (server private key) and `server.crt` (server certificate) in the master data directory. These files must be set up correctly before an SSL-enabled Greenplum system can start.

Important: Do not protect the private key with a passphrase. The server does not prompt for a passphrase for the private key, and the database startup fails with an error if one is required.

A self-signed certificate can be used for testing, but a certificate signed by a certificate authority (CA) should be used in production, so the client can verify the identity of the server. Either a global or local CA can be used. If all the clients are local to the organization, a local CA is recommended. See *Creating a Self-Signed Certificate* for steps to create a self-signed certificate.
Configuring Database Authorization

Describes how to restrict authorization access to database data at the user level by using roles and permissions.

Access Permissions and Roles

Greenplum Database manages database access permissions using roles. The concept of roles subsumes the concepts of users and groups. A role can be a database user, a group, or both. Roles can own database objects (for example, tables) and can assign privileges on those objects to other roles to control access to the objects. Roles can be members of other roles, thus a member role can inherit the object privileges of its parent role.

Every Greenplum Database system contains a set of database roles (users and groups). Those roles are separate from the users and groups managed by the operating system on which the server runs. However, for convenience you may want to maintain a relationship between operating system user names and Greenplum Database role names, since many of the client applications use the current operating system user name as the default.

In Greenplum Database, users log in and connect through the master instance, which verifies their role and access privileges. The master then issues out commands to the segment instances behind the scenes using the currently logged in role.

Roles are defined at the system level, so they are valid for all databases in the system.

To bootstrap the Greenplum Database system, a freshly initialized system always contains one predefined superuser role (also referred to as the system user). This role will have the same name as the operating system user that initialized the Greenplum Database system. Customarily, this role is named gpadmin. To create more roles you first must connect as this initial role.

Managing Object Privileges

When an object (table, view, sequence, database, function, language, schema, or tablespace) is created, it is assigned an owner. The owner is normally the role that executed the creation statement. For most kinds of objects, the initial state is that only the owner (or a superuser) can do anything with the object. To allow other roles to use it, privileges must be granted. Greenplum Database supports the following privileges for each object type:

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables, Views, Sequences</td>
<td>SELECT, INSERT, UPDATE, DELETE, RULE, ALL</td>
</tr>
<tr>
<td>External Tables</td>
<td>SELECT, RULE, ALL</td>
</tr>
</tbody>
</table>
Privileges must be granted for each object individually. For example, granting ALL on a database does not grant full access to the objects within that database. It only grants all of the database-level privileges (CONNECT, CREATE, TEMPORARY) to the database itself.

Use the `GRANT` SQL command to give a specified role privileges on an object. For example:

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

To revoke privileges, use the `REVOKE` command. For example:

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

You can also use the `DROP OWNED` and `REASSIGN OWNED` commands for managing objects owned by deprecated roles. (Note: only an object's owner or a superuser can drop an object or reassign ownership.) For example:

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

**Using SSH-256 Encryption**

Greenplum Database access control corresponds roughly to the Orange Book 'C2' level of security, not the 'B1' level. Greenplum Database currently supports access privileges at the object level. Row-level or column-level access is not supported, nor is labeled security.

Row-level and column-level access can be simulated using views to restrict the columns and/or rows that are selected. Row-level labels can be simulated by adding an extra column to the table to store sensitivity information, and then using views to control row-level access based on this column. Roles can then be granted access to the views rather than the base table. While these workarounds do not provide the same as "B1" level security, they may still be a viable alternative for many organizations.

To use SHA-256 encryption, you must set a parameter either at the system or the session level. This section outlines how to use a server parameter to implement SHA-256 encrypted password storage. Note that in order to use SHA-256 encryption for storage, the client authentication method must be set to password rather than the default, MD5. (See Configuring the SSL Client Connection for more details.) This means that the password is transmitted in clear text over the network, so we highly recommend that you set up SSL to encrypt the client server communication channel.

You can set your chosen encryption method system-wide or on a per-session basis. The available encryption methods are SHA-256 and MD5 (for backward compatibility).
Setting Encryption Method System-wide

To set the `password_hash_algorithm` server parameter on a complete Greenplum system (master and its segments):

1. Log in to your Greenplum Database instance as a superuser.
2. Execute `gpconfig` with the `password_hash_algorithm` set to SHA-256:
   
   ```
   $ gpconfig -c password_hash_algorithm -v 'SHA-256'
   ```

3. Verify the setting:
   
   ```
   $ gpconfig -s
   ```

   You will see:
   
   ```
   Master value: SHA-256
   Segment value: SHA-256
   ```

Setting Encryption Method for an Individual Session

To set the `password_hash_algorithm` server parameter for an individual session:

1. Log in to your Greenplum Database instance as a superuser.
2. Set the `password_hash_algorithm` to SHA-256:
   
   ```
   # set password_hash_algorithm = 'SHA-256'
   ```

3. Verify the setting:
   
   ```
   # show password_hash_algorithm;
   ```

   You will see:
   
   ```
   SHA-256
   ```

Following is an example of how the new setting works:

1. Log in as a super user and verify the password hash algorithm setting:
   
   ```
   # show password_hash_algorithm
   password_hash_algorithm
   -----------------------------------
   SHA-256
   ```

2. Create a new role with password that has login privileges.
   
   ```
   create role testdb with password 'testdb12345#' LOGIN;
   ```

3. Change the client authentication method to allow for storage of SHA-256 encrypted passwords:
   
   Open the `pg_hba.conf` file on the master and add the following line:
   
   ```
   host all testdb 0.0.0.0/0 password
   ```

4. Restart the cluster.
5. Log in to the database as the user just created, \texttt{testdb}.

$p$psql -U testdb$

6. Enter the correct password at the prompt.

7. Verify that the password is stored as a SHA-256 hash.

Password hashes are stored in \texttt{pg_authid.rolpassword}.

8. Log in as the super user.

9. Execute the following query:

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

\section*{Restricting Access by Time}

Greenplum Database enables the administrator to restrict access to certain times by role. Use the \texttt{CREATE ROLE} or \texttt{ALTER ROLE} commands to specify time-based constraints.

Access can be restricted by day or by day and time. The constraints are removable without deleting and recreating the role.

Time-based constraints only apply to the role to which they are assigned. If a role is a member of another role that contains a time constraint, the time constraint is not inherited.

Time-based constraints are enforced only during login. The \texttt{SET ROLE} and \texttt{SET SESSION AUTHORIZATION} commands are not affected by any time-based constraints.

Superuser or \texttt{CREATEROLE} privileges are required to set time-based constraints for a role. No one can add time-based constraints to a superuser.

There are two ways to add time-based constraints. Use the keyword \texttt{DENY} in the \texttt{CREATE ROLE} or \texttt{ALTER ROLE} command followed by one of the following.

- A day, and optionally a time, when access is restricted. For example, no access on Wednesdays.
- An interval—that is, a beginning and ending day and optional time—when access is restricted. For example, no access from Wednesday 10 p.m. through Thursday at 8 a.m.

You can specify more than one restriction; for example, no access Wednesdays at any time and no access on Fridays between 3:00 p.m. and 5:00 p.m.

There are two ways to specify a day. Use the word \texttt{DAY} followed by either the English term for the weekday, in single quotation marks, or a number between 0 and 6, as shown in the table below.

<table>
<thead>
<tr>
<th>English Term</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAY 'Sunday'</td>
<td>DAY 0</td>
</tr>
<tr>
<td>DAY 'Monday'</td>
<td>DAY 1</td>
</tr>
<tr>
<td>DAY 'Tuesday'</td>
<td>DAY 2</td>
</tr>
<tr>
<td>DAY 'Wednesday'</td>
<td>DAY 3</td>
</tr>
<tr>
<td>DAY 'Thursday'</td>
<td>DAY 4</td>
</tr>
<tr>
<td>DAY 'Friday'</td>
<td>DAY 5</td>
</tr>
<tr>
<td>DAY 'Saturday'</td>
<td>DAY 6</td>
</tr>
</tbody>
</table>
A time of day is specified in either 12- or 24-hour format. The word `TIME` is followed by the specification in single quotation marks. Only hours and minutes are specified and are separated by a colon (`:`). If using a 12-hour format, add `AM` or `PM` at the end. The following examples show various time specifications.

```
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:** Time-based authentication is enforced with the server time. Timezones are disregarded.

To specify an interval of time during which access is denied, use two day/time specifications with the words `BETWEEN` and `AND`, as shown. `DAY` is always required.

```
BETWEEN DAY 'Monday' AND DAY 'Tuesday'
BETWEEN DAY 'Monday' TIME '00:00' AND DAY 'Monday' TIME '01:00'
BETWEEN DAY 'Monday' TIME '12:00 AM' AND DAY 'Tuesday' TIME '02:00 AM'
BETWEEN DAY 'Monday' TIME '00:00' AND DAY 'Tuesday' TIME '02:00'
BETWEEN DAY 'Monday' TIME '02:00'
```

The last three statements are equivalent.

**Note:** Intervals of days cannot wrap past Saturday.

The following syntax is not correct:

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

The correct specification uses two `DENY` clauses, as follows:

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

The following examples demonstrate creating a role with time-based constraints and modifying a role to add time-based constraints. Only the statements needed for time-based constraints are shown. For more details on creating and altering roles see the descriptions of `CREATE ROLE` and `ALTER ROLE` in the *Greenplum Database Reference Guide*.

**Example 1 – Create a New Role with Time-based Constraints**

No access is allowed on weekends.

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

**Example 2 – Alter a Role to Add Time-based Constraints**

No access is allowed every night between 2:00 a.m. and 4:00 a.m.

```
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'

Example 3 – Alter a Role to Add Time-based Constraints

No access is allowed Wednesdays or Fridays between 3:00 p.m. and 5:00 p.m.

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

Dropping a Time-based Restriction

To remove a time-based restriction, use the ALTER ROLE command. Enter the keywords DROP DENY FOR followed by a day/time specification to drop.

DROP DENY FOR DAY 'Sunday'

Any constraint containing all or part of the conditions in a DROP clause is removed. For example, if an existing constraint denies access on Mondays and Tuesdays, and the DROP clause removes constraints for Mondays, the existing constraint is completely dropped. The DROP clause completely removes all constraints that overlap with the constraint in the drop clause. The overlapping constraints are completely removed even if they contain more restrictions that the restrictions mentioned in the DROP clause.

Example 1 - Remove a Time-based Restriction from a Role

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

This statement would remove all constraints that overlap with a Monday constraint for the role generaluser in Example 2, even if there are additional constraints.
Greenplum Command Center Security

Greenplum Command Center (GPCC) is a web-based application for monitoring and managing Greenplum clusters. GPCC works with data collected by agents running on the segment hosts and saved to the gpperfmon database. The gpperfmon database is created by running the gpperfmon_install utility, which also creates the gpmon database role that GPCC uses to access the gpperfmon database.

The gpmon User

The gpperfmon_install utility creates the gpmon database role and adds the role to the pg_hba.conf file with the following entries:

<table>
<thead>
<tr>
<th></th>
<th>gpperfmon</th>
<th>gpmon</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>gpperfmon</td>
<td>gpmon</td>
<td>md5</td>
<td></td>
</tr>
<tr>
<td>host</td>
<td>all</td>
<td>gpmon</td>
<td>127.0.0.1/28</td>
<td>md5</td>
</tr>
<tr>
<td>host</td>
<td>all</td>
<td>gpmon</td>
<td>::1/128</td>
<td>md5</td>
</tr>
</tbody>
</table>

These entries allow gpmon to establish a local socket connection to the gpperfmon database and a TCP/IP connection to any database.

The gpmon database role is a superuser. In a secure or production environment, it may be desirable to restrict the gpmon user to just the gpperfmon database. Do this by editing the gpmon host entry in the pg_hba.conf file and changing all in the database field to gpperfmon:

<table>
<thead>
<tr>
<th></th>
<th>gpperfmon</th>
<th>gpmon</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>gpperfmon</td>
<td>gpmon</td>
<td>md5</td>
<td></td>
</tr>
<tr>
<td>host</td>
<td>gpperfmon</td>
<td>gpmon</td>
<td>127.0.0.1/28</td>
<td>md5</td>
</tr>
<tr>
<td>host</td>
<td>gpperfmon</td>
<td>gpmon</td>
<td>::1/128</td>
<td>md5</td>
</tr>
</tbody>
</table>

The password used to authenticate the gpmon user is set by the gpperfmon_install utility and is stored in the gpadmin home directory in the ~/.pgpass file. The ~/.pgpass file must be owned by the gpadmin user and be RW-accessible only by the gpadmin user. To change the gpmon password, use the ALTER ROLE command to change the password in the database, change the password in the ~/.pgpass file, and then restart GPCC with the gpcmdr --restart instance_name command.

Note: The GPCC web server can be configured to encrypt connections with SSL. Two-way authentication with public keys can also be enabled for GPCC users. However, the gpmon user always uses md5 authentication with the password saved in the ~/.pgpass file.

GPCC does not allow logins from any role configured with trust authentication, including the gpadmin user.

The gpmon user can log in to the Command Center Console and has access to all of the application’s features. You can allow other database roles access to GPCC so that you can secure the gpmon user and restrict other users’ access to GPCC features. Setting up other GPCC users is described in the next section.

Greenplum Command Center Users

GPCC has the following types of users:

- **Self Only** users can view metrics and view and cancel their own queries. Any Greenplum Database user successfully authenticated through the Greenplum Database authentication system can access Greenplum Command Center with Self Only permission. Higher permission levels are required to view and cancel other’s queries and to access the System and Admin Control Center features.

- **Basic** users can view metrics, view all queries, and cancel their own queries. Users with Basic permission are members of the Greenplum Database gpcc_basic group.
Operator Basic users can view metrics, view their own and others’ queries, cancel their own queries, and view the System and Admin screens. Users with Operator Basic permission are members of the Greenplum Database gpcc_operator_basic group.

Operator users can view their own and others’ queries, cancel their own and other’s queries, and view the System and Admin screens. Users with Operator permission are members of the Greenplum Database gpcc_operator group.

Admin users can access all views and capabilities in the Command Center. Greenplum Database users with the SUPERUSER privilege have Admin permissions in Command Center.

To log in to the GPCC web application, a user must be allowed access to the gpperfmon database in pg_hba.conf. For example, to make user1 a regular GPCC user, edit the pg_hba.conf file and either add or edit a line for the user so that the gpperfmon database is included in the database field. For example:

```
host     gpperfmon,accounts   user1     127.0.0.1/28    md5
```

To designate a user as an operator, grant the gpcc_operator role to the user:

```
=# GRANT gpcc_operator TO user;
```

You can also grant gpcc_operator to a group role to make all members of the group GPCC operators.

See the gpperfmon_install reference in Greenplum Database Utility Guide for more information about managing the gpperfmon database.

Enabling SSL for Greenplum Command Center

The GPCC web server can be configured to support SSL so that client connections are encrypted. A server certificate can be generated when the Command Center instance is created or you can supply an existing certificate.

Two-way authentication with public key encryption can also be enabled for GPCC. See the Greenplum Command Center Administration Guide for instructions.

Enabling Kerberos Authentication for Greenplum Command Center Users

If Kerberos authentication is enabled for Greenplum Database, Command Center users can also authenticate with Kerberos. Command Center supports three Kerberos authentication modes: strict, normal, and gpmon-only.

Strict

Command Center has a Kerberos keytab file containing the Command Center service principal and a principal for every Command Center user. If the principal in the client’s connection request is in the keytab file, the web server grants the client access and the web server connects to Greenplum Database using the client’s principal name. If the principal is not in the keytab file, the connection request fails.

Normal

The Command Center Kerberos keytab file contains the Command Center principal and may contain principals for Command Center users. If the principal in the client’s connection request is in Command Center’s keytab file, it uses the client’s principal for database connections. Otherwise, Command Center uses the gpmon user for database connections.

gpmon-only

The Command Center uses the gpmon database role for all Greenplum Database connections. No client principals are needed in the Command Center’s keytab file.
See the *Greenplum Command Center documentation* for instructions to enable Kerberos authentication with Greenplum Command Center.
Auditing

Describes Greenplum Database events that are logged and should be monitored to detect security threats.

Greenplum Database is capable of auditing a variety of events, including startup and shutdown of the system, segment database failures, SQL statements that result in an error, and all connection attempts and disconnections. Greenplum Database also logs SQL statements and information regarding SQL statements, and can be configured in a variety of ways to record audit information with more or less detail. The `log_error_verbosity` configuration parameter controls the amount of detail written in the server log for each message that is logged. Similarly, the `log_min_error_statement` parameter allows administrators to configure the level of detail recorded specifically for SQL statements, and the `log_statement` parameter determines the kind of SQL statements that are audited. Greenplum Database records the username for all auditable events, when the event is initiated by a subject outside the Greenplum Database.

Greenplum Database prevents unauthorized modification and deletion of audit records by only allowing administrators with an appropriate role to perform any operations on log files. Logs are stored in a proprietary format using comma-separated values (CSV). Each segment and the master stores its own log files, although these can be accessed remotely by an administrator. Greenplum Database also authorizes overwriting of old log files via the `log_truncate_on_rotation` parameter. This is a local parameter and must be set on each segment and master configuration file.

Greenplum provides an administrative schema called `gp_toolkit` that you can use to query log files, as well as system catalogs and operating environment for system status information. For more information, including usage, refer to The `gp_toolkit` Administrative Schema appendix in the Greenplum Database Reference Guide.

Viewing the Database Server Log Files

Every database instance in Greenplum Database (master and segments) is a running PostgreSQL database server with its own server log file. Daily log files are created in the `pg_log` directory of the master and each segment data directory.

The server log files are written in comma-separated values (CSV) format. Not all log entries will have values for all of the log fields. For example, only log entries associated with a query worker process will have the `slice_id` populated. Related log entries of a particular query can be identified by its session identifier (`gp_session_id`) and command identifier (`gp_command_count`).

<table>
<thead>
<tr>
<th>#</th>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>event_time</td>
<td>timestamp with time zone</td>
<td>Time that the log entry was written to the log</td>
</tr>
<tr>
<td>2</td>
<td>user_name</td>
<td>varchar(100)</td>
<td>The database user name</td>
</tr>
<tr>
<td>3</td>
<td>database_name</td>
<td>varchar(100)</td>
<td>The database name</td>
</tr>
<tr>
<td>4</td>
<td>process_id</td>
<td>varchar(10)</td>
<td>The system process id (prefixed with &quot;p&quot;)</td>
</tr>
<tr>
<td>5</td>
<td>thread_id</td>
<td>varchar(50)</td>
<td>The thread count (prefixed with &quot;th&quot;)</td>
</tr>
<tr>
<td>6</td>
<td>remote_host</td>
<td>varchar(100)</td>
<td>On the master, the hostname/address of the client machine. On the segment, the hostname/address of the master.</td>
</tr>
<tr>
<td>#</td>
<td>Field Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----</td>
<td>--------------------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>7</td>
<td>remote_port</td>
<td>varchar(10)</td>
<td>The segment or master port number</td>
</tr>
<tr>
<td>8</td>
<td>session_start_time</td>
<td>timestamp with time zone</td>
<td>Time session connection was opened</td>
</tr>
<tr>
<td>9</td>
<td>transaction_id</td>
<td>int</td>
<td>Top-level transaction ID on the master. This ID is the parent of any subtransactions.</td>
</tr>
<tr>
<td>10</td>
<td>gp_session_id</td>
<td>text</td>
<td>Session identifier number (prefixed with &quot;con&quot;)</td>
</tr>
<tr>
<td>11</td>
<td>gp_command_count</td>
<td>text</td>
<td>The command number within a session (prefixed with &quot;cmd&quot;)</td>
</tr>
<tr>
<td>12</td>
<td>gp_segment</td>
<td>text</td>
<td>The segment content identifier (prefixed with &quot;seg&quot; for primaries or &quot;mir&quot; for mirrors). The master always has a content id of -1.</td>
</tr>
<tr>
<td>13</td>
<td>slice_id</td>
<td>text</td>
<td>The slice id (portion of the query plan being executed)</td>
</tr>
<tr>
<td>14</td>
<td>distr_tranx_id</td>
<td>text</td>
<td>Distributed transaction ID</td>
</tr>
<tr>
<td>15</td>
<td>local_tranx_id</td>
<td>text</td>
<td>Local transaction ID</td>
</tr>
<tr>
<td>16</td>
<td>sub_tranx_id</td>
<td>text</td>
<td>Subtransaction ID</td>
</tr>
<tr>
<td>17</td>
<td>event_severity</td>
<td>varchar(10)</td>
<td>Values include: LOG, ERROR, FATAL, PANIC, DEBUG1, DEBUG2</td>
</tr>
<tr>
<td>18</td>
<td>sql_state_code</td>
<td>varchar(10)</td>
<td>SQL state code associated with the log message</td>
</tr>
<tr>
<td>19</td>
<td>event_message</td>
<td>text</td>
<td>Log or error message text</td>
</tr>
<tr>
<td>20</td>
<td>event_detail</td>
<td>text</td>
<td>Detail message text associated with an error or warning message</td>
</tr>
<tr>
<td>21</td>
<td>event_hint</td>
<td>text</td>
<td>Hint message text associated with an error or warning message</td>
</tr>
<tr>
<td>22</td>
<td>internal_query</td>
<td>text</td>
<td>The internally-generated query text</td>
</tr>
<tr>
<td>23</td>
<td>internal_query_pos</td>
<td>int</td>
<td>The cursor index into the internally-generated query text</td>
</tr>
<tr>
<td>24</td>
<td>event_context</td>
<td>text</td>
<td>The context in which this message gets generated</td>
</tr>
<tr>
<td>#</td>
<td>Field Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----</td>
<td>------------------------</td>
<td>-----------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>25</td>
<td>debug_query_string</td>
<td>text</td>
<td>User-supplied query string with full detail for debugging. This string can be modified for internal use.</td>
</tr>
<tr>
<td>26</td>
<td>error_cursor_pos</td>
<td>int</td>
<td>The cursor index into the query string</td>
</tr>
<tr>
<td>27</td>
<td>func_name</td>
<td>text</td>
<td>The function in which this message is generated</td>
</tr>
<tr>
<td>28</td>
<td>file_name</td>
<td>text</td>
<td>The internal code file where the message originated</td>
</tr>
<tr>
<td>29</td>
<td>file_line</td>
<td>int</td>
<td>The line of the code file where the message originated</td>
</tr>
<tr>
<td>30</td>
<td>stack_trace</td>
<td>text</td>
<td>Stack trace text associated with this message</td>
</tr>
</tbody>
</table>

Greenplum provides a utility called `gplogfilter` that can be used to search through a Greenplum Database log file for entries matching the specified criteria. By default, this utility searches through the Greenplum master log file in the default logging location. For example, to display the last three lines of the master log file:

```
$ gplogfilter -n 3
```

You can also use `gplogfilter` to search through all segment log files at once by running it through the `gpssh` utility. For example, to display the last three lines of each segment log file:

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

The following are the Greenplum security-related audit (or logging) server configuration parameters that are set in the `postgresql.conf` configuration file:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Value Range</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_connections</td>
<td>Boolean</td>
<td>off</td>
<td>This outputs a line to the server log detailing each successful connection. Some client programs, like <code>psql</code>, attempt to connect twice while determining if a password is required, so duplicate &quot;connection received&quot; messages do not always indicate a problem.</td>
</tr>
<tr>
<td>log_disconnections</td>
<td>Boolean</td>
<td>off</td>
<td>This outputs a line in the server log at termination of a client session, and includes the duration of the session.</td>
</tr>
<tr>
<td>Field Name</td>
<td>Value Range</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| log_statement      | NONE  
                     | DDL     
                     | MOD     
                     | ALL    | Controls which SQL statements are logged. DDL logs all data definition commands like CREATE, ALTER, and DROP commands. MOD logs all DDL statements, plus INSERT, UPDATE, DELETE, TRUNCATE, and COPY FROM. PREPARE and EXPLAIN ANALYZE statements are also logged if their contained command is of an appropriate type. |
| log_hostname       | Boolean                      | off     | By default, connection log messages only show the IP address of the connecting host. Turning on this option causes logging of the host name as well. Note that depending on your host name resolution setup this might impose a non-negligible performance penalty. |
| log_duration       | Boolean                      | off     | Causes the duration of every completed statement which satisfies log_statement to be logged.                                                                                                               |
| log_error_verbosity| TERSE  
                     | DEFAULT  
<pre><code>                 | VERBOSE | Controls the amount of detail written in the server log for each message that is logged.                                                                                                                     |
</code></pre>
<p>| log_min_duration_statement | number of milliseconds, 0, -1 | -1      | Logs the statement and its duration on a single log line if its duration is greater than or equal to the specified number of milliseconds. Setting this to 0 will print all statements and their durations. -1 disables the feature. For example, if you set it to 250 then all SQL statements that run 250ms or longer will be logged. Enabling this option can be useful in tracking down unoptimized queries in your applications. |</p>
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Value Range</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_min_messages</td>
<td>DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, INFO, NOTICE, WARNING, ERROR, LOG, FATAL, PANIC</td>
<td>NOTICE</td>
<td>Controls which message levels are written to the server log. Each level includes all the levels that follow it. The later the level, the fewer messages are sent to the log.</td>
</tr>
<tr>
<td>log_rotation_age</td>
<td>Any valid time expression (number and unit)</td>
<td>1d</td>
<td>Determines the maximum lifetime of an individual log file. After this time has elapsed, a new log file will be created. Set to zero to disable time-based creation of new log files.</td>
</tr>
<tr>
<td>log_statement_stats</td>
<td>Boolean</td>
<td>off</td>
<td>For each query, write total performance statistics of the query parser, planner, and executor to the server log. This is a crude profiling instrument.</td>
</tr>
<tr>
<td>log_truncate_on_rotation</td>
<td>Boolean</td>
<td>off</td>
<td>Truncates (overwrites), rather than appends to, any existing log file of the same name. Truncation will occur only when a new file is being opened due to time-based rotation. For example, using this setting in combination with a log_filename such as gpseg##-%H. log would result in generating twenty-four hourly log files and then cyclically overwriting them. When off, pre-existing files will be appended to in all cases.</td>
</tr>
</tbody>
</table>
Encrypting Data and Database Connections

Describes how to encrypt data at rest in the database or in transit over the network, to protect from eavesdroppers or man-in-the-middle attacks.

- Connections between clients and the master database can be encrypted with SSL. This is enabled with the `ssl` server configuration parameter, which is off by default. Setting the `ssl` parameter to on allows client communications with the master to be encrypted. The master database must be set up for SSL. See OpenSSL Configuration for more about encrypting client connections with SSL.
- Greenplum Database allows SSL encryption of data in transit between the Greenplum parallel file distribution server, `gpfdist`, and segment hosts. See Encrypting gpfdist Connections for more information.
- The `pgcrypto` package of encryption/decryptions functions protect data at rest in the database. Encryption at the column level protects sensitive information, such as social security numbers or credit card numbers. See Encrypting Data at Rest with pgcrypto for more information.

Encrypting gpfdist Connections

The `gpfdists` protocol is a secure version of the `gpfdist` protocol that securely identifies the file server and the Greenplum Database and encodes the communications between them. Using `gpfdists` protects against eavesdropping and man-in-the-middle attacks.

The `gpfdists` protocol implements client/server SSL security with the following notable features:

- Client certificates are required.
- Multilingual certificates are not supported.
- A Certificate Revocation List (CRL) is not supported.
- The TLSv1 protocol is used with the `TLS_RSA_WITH_AES_128_CBC_SHA` encryption algorithm. These SSL parameters cannot be changed.
- SSL renegotiation is supported.
- The SSL ignore host mismatch parameter is set to false.
- Private keys containing a passphrase are not supported for the `gpfdist` file server (server.key) or for the Greenplum Database (client.key).
- It is the user's responsibility to issue certificates that are appropriate for the operating system in use. Generally, converting certificates to the required format is supported, for example using the SSL Converter at https://www.sslshopper.com/ssl-converter.html.

A `gpfdist` server started with the `--ssl` option can only communicate with the `gpfdists` protocol. A `gpfdist` server started without the `--ssl` option can only communicate with the `gpfdist` protocol. For more detail about `gpfdist` refer to the Greenplum Database Administrator Guide.

There are two ways to enable the `gpfdists` protocol:

- Run `gpfdist` with the `--ssl` option and then use the `gpfdists` protocol in the `LOCATION` clause of a `CREATE EXTERNAL TABLE` statement.
- Use a YAML control file with the SSL option set to true and run `gpload`. Running `gpload` starts the `gpfdist` server with the `--ssl` option and then uses the `gpfdists` protocol.

When using `gpfdists`, the following client certificates must be located in the `$PGDATA/gpfdists` directory on each segment:

- The client certificate file, `client.crt`
- The client private key file, `client.key`
- The trusted certificate authorities, `root.crt`

**Important:** Do not protect the private key with a passphrase. The server does not prompt for a passphrase for the private key, and loading data fails with an error if one is required.
When using `gpload` with SSL you specify the location of the server certificates in the YAML control file. When using `gpfdist` with SSL, you specify the location of the server certificates with the `--ssl` option.

The following example shows how to securely load data into an external table. The example creates a readable external table named `ext_expenses` from all files with the `.txt` extension, using the `gpfdists` protocol. The files are formatted with a pipe (|) as the column delimiter and an empty space as null.

1. Run `gpfdist` with the `--ssl` option on the segment hosts.
2. Log into the database and execute the following command:

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

### Encrypting Data at Rest with pgcrypto

The `pgcrypto` package for Greenplum Database provides functions for encrypting data at rest in the database. Administrators can encrypt columns with sensitive information, such as social security numbers or credit card numbers, to provide an extra layer of protection. Database data stored in encrypted form cannot be read by users who do not have the encryption key, and the data cannot be read directly from disk.

`pgcrypto` allows PGP encryption using symmetric and asymmetric encryption. Symmetric encryption encrypts and decrypts data using the same key and is faster than asymmetric encryption. It is the preferred method in an environment where exchanging secret keys is not an issue. With asymmetric encryption, a public key is used to encrypt data and a private key is used to decrypt data. This is slower than symmetric encryption and it requires a stronger key.

Using `pgcrypto` always comes at the cost of performance and maintainability. It is important to use encryption only with the data that requires it. Also, keep in mind that you cannot search encrypted data by indexing the data.

Before you implement in-database encryption, consider the following PGP limitations.

- No support for signing. That also means that it is not checked whether the encryption sub-key belongs to the master key.
- No support for encryption key as master key. This practice is generally discouraged, so this limitation should not be a problem.
- No support for several subkeys. This may seem like a problem, as this is common practice. On the other hand, you should not use your regular GPG/PGP keys with `pgcrypto`, but create new ones, as the usage scenario is rather different.

Greenplum Database is compiled with zlib by default; this allows PGP encryption functions to compress data before encrypting. When compiled with OpenSSL, more algorithms will be available.

Because `pgcrypto` functions run inside the database server, the data and passwords move between `pgcrypto` and the client application in clear-text. For optimal security, you should connect locally or use SSL connections and you should trust both the system and database administrators.

`pgcrypto` configures itself according to the findings of the main PostgreSQL configure script.

When compiled with `zlib`, `pgcrypto` encryption functions are able to compress data before encrypting.

`Pgcrypto` has various levels of encryption ranging from basic to advanced built-in functions. The following table shows the supported encryption algorithms.
### Table 66: Pgcrypto Supported Encryption Functions

<table>
<thead>
<tr>
<th>Value</th>
<th>Built-in</th>
<th>With OpenSSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>SHA1</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>SHA224/256/384/512</td>
<td>yes</td>
<td>yes³</td>
</tr>
<tr>
<td>Other digest algorithms</td>
<td>no</td>
<td>yes⁴</td>
</tr>
<tr>
<td>Blowfish</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>AES</td>
<td>yes</td>
<td>yes⁵</td>
</tr>
<tr>
<td>DES/3DES/CAST5</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Raw Encryption</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>PGP Symmetric-Key</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>PGP Public Key</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

### Creating PGP Keys
To use PGP asymmetric encryption in Greenplum Database, you must first create public and private keys and install them.

This section assumes you are installing Greenplum Database on a Linux machine with the Gnu Privacy Guard (gpg) command line tool. Use the latest version of GPG to create keys. Download and install Gnu Privacy Guard (GPG) for your operating system from [https://www.gnupg.org/download/](https://www.gnupg.org/download/). On the GnuPG website you will find installers for popular Linux distributions and links for Windows and Mac OS X installers.

1. As root, execute the following command and choose option 1 from the menu:

   ```
   # 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
   ```

---

³ SHA2 algorithms were added to OpenSSL in version 0.9.8. For older versions, pgcrypto will use built-in code.

⁴ Any digest algorithm OpenSSL supports is automatically picked up. This is not possible with ciphers, which need to be supported explicitly.

⁵ AES is included in OpenSSL since version 0.9.7. For older versions, pgcrypto will use built-in code.
2. Respond to the prompts and follow the instructions, as shown in this example:

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

3. List the PGP keys by entering the following command:

```bash
gpg --list-secret-keys
/root/.gnupg/secring.gpg
```

```
sec 2048R/2027CC30 2015-01-13 [expires: 2016-01-13]
uid John Doe <jdoe@email.com>
sub 2048R/4FD2EFBB 2015-01-13
```
2027CC30 is the public key and will be used to encrypt data in the database. 4FD2EFBB is the private (secret) key and will be used to decrypt data.

4. Export the keys using the following commands:

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

See the `pgcrypto` documentation for more information about PGP encryption functions.

### Encrypting Data in Tables using PGP

This section shows how to encrypt data inserted into a column using the PGP keys you generated.

1. Dump the contents of the public.key file and then copy it to the clipboard:

```bash
# cat public.key
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.14 (GNU/Linux)
mQENBFS1zF0BCADNw8Qvk1V1C6Kfcwd3Kpm/dijPfRyyEwB6pQyA05jtWIxZTh2HislojSP6L0cSklqMU9LAlncec3hRbHuVgK1GS9d9tex2nnSL9Admq1k/yXr5sy4VgK+qcdWuU2g9oOomOmyJh3cn+kkbRTEmuM3f1bMs8shQwzvMstCUVmuHU/VvG5rJae8PuYS6SJC74fI6W7S0H3RrIc7IfuL6xxYddV4213cd44b18/171hq2uyN2/Hbjsji2ymq77tw3jswAwz2gPnsnsDgyQyDy/o9nNgC8EGI1g96ZFnE6Pwabh+iEc8MD0I5/GA1R6hcOZi1H6KEcavruQ1ikjnaABEBAAGOHHRlC3gqa2V5ID0XZNOa2V5QGtvYw1sLmnVnbo7JAT4EewECACFAlg12f0CgwMFCQhM4AGCwk1BwMCbCh1bYAgkBkCqwMAgMBAhAh4BbAheAAAAoJEA9c914gJ8wbwfh/3VvVsPKq11owRNJxvXGt1bY7BFvrU52ykp+PP2Yoes90pdL3CRK8gAm9bx55k08q2UXSLc6fFOEP4W4WgmGyF8JRoc3oeezTkmCBW8I1bU0qG7xFxodXLufGCE7hVWQe9HcSntITLxGov1mJAw07TAcoccXLBxYu2h9RflvLoqKzccyOhH51qXaQ0T10TeFeEp9Tlwcntg3WCSUSP0DG0UAnajDZ3KE8q7Pv74fhG1E2VzHb8PajR62CXSHFkpBgiXnxTok45XbXAdN4etUXPSnwPi46goAp9UqQosfGyB1XD0TB2UoghutAMECa7M7tePv79i02/NfIpF5AQ0E0VL1QEiANabFdQ+8QMCQAdOipAbJfF/RJQz3Uoc4BTqICaxdyAzfz0tUSF7Zro2us9G1AqR1WdE8Eqcl/xmfcJiYjUam6zFxcgNh5Y1sdTMT2dLp5WoejwgCGW/ZLu4wzzIFFDkPv99Dw65MNLtJRSp4hS5o2apKdBo4Ex8304Mynav/rEiDC Wu4T0hv3hskCpkeLcw/s1loqz+aNmP0wpf14hR3UUMP70+9l1beFpwz2JbVLzl31LloHRgpc2la+PzzbEks16j7q7V9kqziTCizXwoALlui1fjFj0v3q9h0/v8/yAnkAmowz2rIb2KBzhuYmN2y2VukAEEAAYkBjQQA0QADAwUCVL1VQ1bDAUJlAeBZgAAKAARCAHfxJEiCfMwMOHycACFhInzA9uAMAc3f41+MrgMU3r9wI1r048WrdTSxR8m55nx1JxwNyYylyFb/hc9k6h5uww2SSDKj/0/FrRl16PFHP0NPSVzv62WHN2iLas0ua0jb2KqGhONFbJuevkyBy1Rz+hI/+8rJkc2OjQxmnK88kk8gb5X/HMUC55H0q2qQAY0BnJh9gOQ45q6p3K3GZ/7Dm1w5j6KrUfrYl5i1NIGWf8dqyEx/UU2B+dYgCwtx0V0nnbU1KNcmk2AKcEcFK3YolCIxomdOxhFOv9Akjjo1DyC65KJciFy2mIKF2sKQoAh1R3LpMa8zD0Et4w3vckPQNnYUtj6zoCxcv =X28J
-----END PGP PUBLIC KEY BLOCK-----
```

2. Create a table called \texttt{userssn} and insert some sensitive data, social security numbers for Bob and Alice, in this example. Paste the public.key contents after “dearmor(“.

```sql
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'))
CROSS JOIN
(SELECT  dearmor('-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.14 (GNU/Linux)
mQENBFS1zF0BCADNw8Qvk1V1C6Kfcwd3Kpm/dijPfRyyEwB6pQyA05jtWIxZTh2HislojSP6L0cSklqMU9LAlncec3hRbHuVgK1GS9d9tex2nnSL9Admq1k/yXr5sy4VgK+qcdWuU2g9oOomOmyJh3cn+kkbRTEmuM3f1bMs8shQwzvMstCUVmuHU/VvG5rJae8PuYS6SJC74fI6W7S0H3RrIc7IfuL6xxYddV4213cd44b18/171hq2uyN2/Hbjsji2ymq77tw3jswAwz2gPnsnsDgyQyDy/o9nNgC8EGI1g96ZFnE6Pwabh+iEc8MD0I5/GA1R6hcOZi1H6KEcavruQ1ikjnaABEBAAGOHHRlC3gqa2V5ID0XZNOa2V5QGtvYw1sLmnVnbo7JAT4EewECACFAlg12f0CgwMFCQhM4AGCwk1BwMCbCh1bYAgkBkCqwMAgMBAhAh4BbAheAAAAoJEA9c914gJ8wbwfh/3VvVsPKq11owRNJxvXGt1bY7BFvrU52ykp+PP2Yoes90pdL3CRK8gAm9bx55k08q2UXSLc6fFOEP4W4WgmGyF8JRoc3oeezTkmCBW8I1bU0qG7xFxodXLufGCE7hVWQe9HcSntITLxGov1mJAw07TAcoccXLBxYu2h9RflvLoqKzccyOhH51qXaQ0T10TeFeEp9Tlwcntg3WCSUSP0DG0UAnajDZ3KE8q7Pv74fhG1E2VzHb8PajR62CXSHFkpBgiXnxTok45XbXAdN4etUXPSnwPi46goAp9UqQosfGyB1XD0TB2UoghutAMECa7M7tePv79i02/NfIpF5AQ0E0VL1QEiANabFdQ+8QMCQAdOipAbJfF/RJQz3Uoc4BTqICaxdyAzfz0tUSF7Zro2us9G1AqR1WdE8Eqcl/xmfcJiYjUam6zFxcgNh5Y1sdTMT2dLp5WoejwgCGW/ZLu4wzzIFFDkPv99Dw65MNLtJRSp4hS5o2apKdBo4Ex8304Mynav/rEiDC Wu4T0hv3hskCpkeLcw/s1loqz+aNmP0wpf14hR3UUMP70+9l1beFpwz2JbVLzl31LloHRgpc2la+PzzbEks16j7q7V9kqziTCizXwoALlui1fjFj0v3q9h0/v8/yAnkAmowz2rIb2KBzhuYmN2y2VukAEEAAYkBjQQA0QADAwUCVL1VQ1bDAUJlAeBZgAAKAARCAHfxJEiCfMwMOHycACFhInzA9uAMAc3f41+MrgMU3r9wI1r048WrdTSxR8m55nx1JxwNyYylyFb/hc9k6h5uww2SSDKj/0/FrRl16PFHP0NPSVzv62WHN2iLas0ua0jb2KqGhONFbJuevkyBy1Rz+hI/+8rJkc2OjQxmnK88kk8gb5X/HMUC55H0q2qQAY0BnJh9gOQ45q6p3K3GZ/7Dm1w5j6KrUfrYl5i1NIGWf8dqyEx/UU2B+dYgCwtx0V0nnbU1KNcmk2AKcEcFK3YolCIxomdOxhFOv9Akjjo1DyC65KJciFy2mIKF2sKQoAh1R3LpMa8zD0Et4w3vckPQNnYUtj6zoCxcv =X28J
-----END PGP PUBLIC KEY BLOCK-----

''')
```

Version: GnuPG v2.0.14 (GNU/Linux)
3. Verify that the \textit{ssn} column is encrypted.

test_db=# select * from userssn;

<table>
<thead>
<tr>
<th>ssn_id</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Alice</td>
</tr>
<tr>
<td>ssn</td>
<td>\301\300L\003\235M%_O \322\357\273\001\010\000\272\227\101\010\341</td>
</tr>
<tr>
<td>ssn_id</td>
<td>2</td>
</tr>
<tr>
<td>username</td>
<td>Bob</td>
</tr>
</tbody>
</table>

4. Extract the public.key ID from the database:

```sql
SELECT pgp_key_id(dearmor('-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.14 (GNU/Linux)
mQENBFS1Zf0BCADNw8QvK1V1C6Kfwcd3Kpm/4iDjPFryyEw6PqKyA05jtWixZTh2HIslojSP6L10cSklqM9ULAlnceczh2RINhuVgK1G9d9texe2nnSL9Admq3kYR5syVKGqcdDuwy29o0Omyew33h+kkBxTEMuM3flbm85oahMwStMVStCUMwHUCvG5rjAe8FuYDSJC7416w7SOH3RiRic7fl6xYdxV4213ctd44b18/171hq2YuN2/7Bsji2ymq7ttw3j5WAXx2gPnnPsogQyQDy/o9NqCE8Eg1q9G27FnS6E6Pwbhn+iC8M0D1K/GAI16HcO2ZHF8Ecuavru91lkjEBEBAAAG0HHRc3Qga2V5lDX0ZNX0a2V5GVtYWlsLmvNv76JAt4EEwECaCGFA1S1Zf0CGwMFCQHHnM4AGCwkIBwMBCHUIAgkKChwQAgMBAh4BHaeeAAAAeJAd9c14J88Wbfwh/3VYVsPkJ11owRJVJtXG7t1y7BfrtU52y+kPZPyeZoes90qpdL3CMR82gAM9bxS008qUXZSL6FePwEW4UWgwyGFy8JRozCooeeTkmCBWIIbUbUqG70GetzVxopdXLufGCE7hWHQe9HcSntiTLxGovlMjAwOT7AaccXlbYu2h9Rf5LvQoDkzCcy0H51qkAqQT100teFeEpb9TIiwtcg3WCSUSP5DG0uOAanJZDj6Xe0P77fVhIE2ZkstHsnpfBpqiNxnT0K45nBAXDn4etUXPnSwiP46qap9U0qQgof9gBY1DXDTE2U0ghuAMCMA7MVScPe7v97100z/FNnBeA0Q0EVL1V/92E1NAdFDq+8QMCADoIpm1bf/JrQt3zUoc4BTqICaxdyzaFz0OFS/7Zro2us99G1ARqLw8EDq1j/xmfcJizLuyAm62ZxSxcngH51ydGTMJZlDp5WeOjgwC2L2U4w2z0dLF0kX2DWe6s5MNltJrSp4hSsOPa2Kd0B4E8304fMjNanva/rEiDDCWW4T0lh3v3Kcpke61Cwxs+W7x+0z+aMn0P0yfw14hR3U0MP70+V1beFgw2JbVLz3111L0uHRgpcZlq++PzzBEKs16j77V9gkq2TCIzXoWaiLjuzt1fjK0q3v9h0/8y8AnKcAmowzRl4lFg2KKBzhunYM2YKUAEQAAYkBjQQUQIAQIMAUCWVL1Q/1QbDAUJAEzqAAKCRAhFgXeJICf8M0OHYCAFhIn2A9uAM3TC441+MrgMU3rJ3w9izr048WdrThsR8XW5kN1OxJ0WYxYulFy/Pb/shc9k65huw2SSDkj/0Ffr161FPH0NP5Sv62W7+H2N1asUoaoJzb2KQGbhL0nFbJuEvvjyBy1Rz+H+8+8JkC2OjQkkmm8K8Kk8qbb5x/HMUC55h9q2Q9OBFnJrHgQQ45q6pX3G27DbeKSW6K1uUrFy5sLSnWEG8wqEx/UU2B+6YQctwVAOnbnULNCm1sAKAcFk3YoliCxcmd0xHrov9AAkjojDy65JKc1vP2miKFSzF0AgI3LpMaBzdET41w3vCkPNQnNuUtAf76zoCvX =X28J
-----END PGP PUBLIC KEY BLOCK-----'));
```

This shows that the PGP key ID used to encrypt the ssn column is 9D4D255F4FD2EFBB. It is recommended to perform this step whenever a new key is created and then store the ID for tracking.
You can use this key to see which key pair was used to encrypt the data:

```
SELECT username, pgp_key_id(ssn) As key_used
FROM userssn;
```

<table>
<thead>
<tr>
<th>username</th>
<th>Bob</th>
<th>key_used</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Alice</td>
<td>9D4D255F4FD2EFBB</td>
</tr>
</tbody>
</table>

**Note:** Different keys may have the same ID. This is rare, but is a normal event. The client application should try to decrypt with each one to see which fits — like handling ANYKEY. See `pgp_key_id()` in the `pgcrypto` documentation.

5. Decrypt the data using the private key.

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

1Q0YBFS12fOBACDNw8qyK1v1LC36Kfcwd3KpdmjDfPryyEB6pQyA05jtWiXZTh
2HislojSP6lL0cSKqMu9LA1ncec2hRInHuBvgKLgS9d9txeg2nS9L9Adm9qk/yX
R5syVKG+qcdWUyrv92oO0meyjhc3n+kbbRTEmuM3fb1Ms8shOzwVmvstCUMHuVU/V
vg5srj ils8ePyYuSg3C714I6w7S0H3rIc7I7fL6xYddV421tcd44b18/17hqZUYN2
/hbsjii2ymqvt3wJz5Wq9psdn9qQyQ0/o/nMcBEG1lgi96ZFnE6Pvbnh
ic8CM0DI5d/KAlfRm6oZ1HF8KEcauvrlkljknABEAEEAB/wNsfjvb1rfjIjM/j
XwUNM+sI4vz7c2947VtPgf671vqcyY7UJucXvrvz2B16vm165xEUIU7y7bN8A
fe08paM4Wy+x9r4c2zbpBwqarmNN33QAQy4r1BTvrgqWy+mpbd871Tjzwid2nNmx
02szir4q1RT0x21JhrkpF67ftmzOH01rS0bW0VHueMY7TcwmdPe9bQde1Pr
9c11UqBn4/acTtC1CIWJRnE2nXASnixtTIFCV1+90n9yMecMKVnWnfIPkIhYAM
OPFnu/Z/d1r1CRHjNHb516zYUm5qDxQVnnezktxqROESXm0fGmpcPHQogmZmbc
6uyBBADSCXHPFo/VPVTeM5p1yGrNOR2jR2rUj9+po2zd2gjktS/xIKRLbku4Q0l
emuz2wr9dVEX7msOnVdQ5ibtuQ4doJ1dCMeSRQ21eEr1b875VJ3htimb1pn
4JX6SWRXPUJPGXG18u0U0bHOLwJ7M2PFw711ao+mLEA9aJaqWQ+s89BKpL4Ya2
r5en72gsCClswick0rVdf1RgtcwbYDmpm2hOarKjKtOMG6rCXJr6lq1N8w/L
/lgnZtMhch35MCG/zMA2e/bN4VMYpe11wzxVzkJLs7yyqX/Ya7ac7b7DOKxcIfxI
MOSAOhJmHmk1Wl1QRhn3cnyi8w3q7X4OEA/w54FVvvPqg3+scD6sAApM4UU0R3
tisUnVembWmBxmlk8Ajsz7VrveU5y2B8hBuCqj1c7gea/NLXi7f8rscsJScox
Bf+iIc/VNEv4k4ubD41g0m1wc/cen/EnCZQYcINmla4M6Vqg/Rf1tFh4
nwkt/Z/8LsIarZ20HHR1c3q9a2Dn02X0n2a5QGyvWTLSmLnmVgT6J7A4EwEwEC
AcgF1s12f0cGmWFMQhH4MACGkWIBwBCHuiAgkKwCwAGmBAh4RlhaAAAsJEa9d
cl4j38wbbbfw/3VyVsPQk1l0wrJNvXgxtbY7bfrvZU5you+PPYyoes9u0pdL3CMrK
8gAM9bzx580q2UXS2LC6F0pEW4uWqmgyf87JRc3oeezTcmkCBW81lB0uqGetzVx
opdxLvuPFCGE7wVQoe9hCnStniYtLxGovmlJAw07TaocclxLybu29h5r95fLoQdKrzCcyOH
h51qXaqQ0t10TeFeEpb931Tiwctng3WCSUS50PDGoUAonjdJ23KEq8p774Ve6I1EZV
zhhB8fajR62C6SHFrkgBqinXnt0k4SbXAd4eUXP8nwP46q0a99Q0qsgfgyb1X
D0TB2Uq0htuAMECaMTvpeNV47Z/02n/6BEd65gEVLV1/QLIAaFbpdQ+8QCMDAOI
pm1BfR/Jrqt3sU0x1C4T1cxdzaYf0Us5f/7zro2u9991GLaqRwd8Eqcj1/xMcJ
i2yUam6ZaZPZCGHnY51ydsteMTJd2p5weOj2gCwCG/2Lzw14wzOFFZdkIPv9RdW6e5
MNNLtrsp4h5s2apKb04Ex8304mJynav/rElIDCvUW401hv3SkCpke6LcsxW+7
liozp+aNmoP0Ywfi4hR3UUMP7+VibeWJ2WBvL31OLLuHrPqCzla+PzbzBkia16
jg7q7V9g7KZIzCXX0aLiui3tRlfJKO3v9q0h/0v8yAnrkAcwQr21yFgK2bzhunY
mN2YVyuKAEQEAQAQ/A7r4hDrmmxz3QX69FazaPIB7n1Jte2EIEhA8uFF05Q2ZkU/
ic1S7W7tqMGAIyAsdKOhfhxanc9veGufvYygyKt3eSdm1RFKpRJX22m0zKdwy67B
Mu95V20k1l160Cm8d06+2fkmGxGqCwq2sK+yJqxtx3K399y5MCDv2a2c5SN4Tw3
Uc7ztc7/hk61bmaLd7F1d7b7xt7HnQsma81rtpg8ErDtomluRouoDeojeqkYI
ORwasp8n89w2CSpFXsry6hFyGSZa2e1hAqJYLFO2dR0MrzQZC7/7n+IhjupnDC
Nd4a8DVx3y3Qs3A7mVhzFasCdjiX13iw7j0moF0keAOcz1e63SNKnniQ111RlJMj
jagqov6U/xzGMLmwt8TnuOi+Kr8/dboG15sy123OuAb4tfFQAXapo9A4A2KqojqO
xCUvAenfDzaxzrh5LId72nTDKXgyGTWFDF7H2avHJ4981mCWL711z1HBB9XMi6p
dhFvboqy0JS1LaFMF/JoBAd0f3z3RhP7e6X112zdBGthjC5S/10kWbBwg6i2yx
If you created a key with passphrase, you may have to enter it here. However for the purpose of this example, the passphrase is blank.

**Key Management**

Whether you are using symmetric (single private key) or asymmetric (public and private key) cryptography, it is important to store the master or private key securely. There are many options for storing encryption keys, for example, on a file system, key vault, encrypted USB, trusted platform module (TPM), or hardware security module (HSM).

Consider the following questions when planning for key management:

- **Where will the keys be stored?**
- **When should keys expire?**
- **How are keys protected?**
- **How are keys accessed?**
- **How can keys be recovered and revoked?**

The Open Web Application Security Project (OWASP) provides a very comprehensive guide to securing encryption keys.
Enabling gphdfs Authentication with a Kerberos-secured Hadoop Cluster

Provides steps for configuring Greenplum Database to access external tables in a Hadoop cluster secured with Kerberos.

Using external tables and the gphdfs protocol, Greenplum Database can read files from and write files to a Hadoop File System (HDFS). Greenplum segments read and write files in parallel from HDFS for fast performance.

When a Hadoop cluster is secured with Kerberos ("Kerberized"), Greenplum Database must be configured to allow the Greenplum Database gpadmin role, which owns external tables in HDFS, to authenticate through Kerberos. This topic provides the steps for configuring Greenplum Database to work with a Kerberized HDFS, including verifying and troubleshooting the configuration.

- Prerequisites
- Configuring the Greenplum Cluster
- Creating and Installing Keytab Files
- Configuring gphdfs for Kerberos
- Testing Greenplum Database Access to HDFS
- Troubleshooting HDFS with Kerberos

Prerequisites

Make sure the following components are functioning and accessible on the network:

- Greenplum Database cluster
- Kerberos-secured Hadoop cluster. See the Greenplum Database Release Notes for supported Hadoop versions.
- Kerberos Key Distribution Center (KDC) server.

Configuring the Greenplum Cluster

The hosts in the Greenplum Cluster must have a Java JRE, Hadoop client files, and Kerberos clients installed.

Follow these steps to prepare the Greenplum Cluster.

1. Install a Java 1.6 or later JRE on all Greenplum cluster hosts.
   
   Match the JRE version the Hadoop cluster is running. You can find the JRE version by running `java --version` on a Hadoop node.

2. (Optional) Confirm that Java Cryptography Extension (JCE) is present.
   
   The default location of the JCE libraries is $JAVA_HOME/lib/security. If a JDK is installed, the directory is $JAVA_HOME/jre/lib/security. The files local_policy.jar and US_export_policy.jar should be present in the JCE directory.

   The Greenplum cluster and the Kerberos server should, preferably, use the same version of the JCE libraries. You can copy the JCE files from the Kerberos server to the Greenplum cluster, if needed.

3. Set the JAVA_HOME environment variable to the location of the JRE in the .bashrc or .bash_profile file for the gpadmin account. For example:

   ```
   export JAVA_HOME=/usr/java/default
   ```
4. Source the `.bashrc` or `.bash_profile` file to apply the change to your environment. For example:

   $ source ~/.bashrc

5. Install the Kerberos client utilities on all cluster hosts. Ensure the libraries match the version on the KDC server before you install them.

   For example, the following command installs the Kerberos client files on Red Hat or CentOS Linux:

   $ sudo yum install krb5-libs krb5-workstation

   Use the `kinit` command to confirm the Kerberos client is installed and correctly configured.

6. Install Hadoop client files on all hosts in the Greenplum Cluster. Refer to the documentation for your Hadoop distribution for instructions.

7. Set the Greenplum Database server configuration parameters for Hadoop. The `gp_hadoop_target_version` parameter specifies the version of the Hadoop cluster. See the `Greenplum Database Release Notes` for the target version value that corresponds to your Hadoop distribution. The `gp_hadoop_home` parameter specifies the Hadoop installation directory.

   $ gpconfig -c gp_hadoop_target_version -v "hdp2"
   $ gpconfig -c gp_hadoop_home -v "/usr/lib/hadoop"

   See the `Greenplum Database Reference Guide` for more information.

8. Reload the updated `postgresql.conf` files for master and segments:

   `gpstop -u`

   You can confirm the changes with the following commands:

   $ gpconfig -s gp_hadoop_target_version
   $ gpconfig -s gp_hadoop_home

9. Grant Greenplum Database `gphdfs` protocol privileges to roles that own external tables in HDFS, including `gpadmin` and other superuser roles. Grant `SELECT` privileges to enable creating readable external tables in HDFS. Grant `INSERT` privileges to enable creating writable exeternal tables on HDFS.

   `=# GRANT SELECT ON PROTOCOL gphdfs TO gpadmin;`
   `=# GRANT INSERT ON PROTOCOL gphdfs TO gpadmin;`

10. Grant Greenplum Database external table privileges to external table owner roles:

    `ALTER ROLE HDFS_USER CREATEEXTTABLE (type='readable');`
    `ALTER ROLE HDFS_USER CREATEEXTTABLE (type='writable');`

    **Note:** It is best practice to review database privileges, including `gphdfs` external table privileges, at least annually.

### Creating and Installing Keytab Files

1. Log in to the KDC server as root.
2. Use the `kadmin.local` command to create a new principal for the `gpadmin` user:

   `# kadmin.local -q "addprinc -randkey gpadmin@LOCAL.DOMAIN"

3. Use `kadmin.local` to generate a Kerberos service principal for each host in the Greenplum Database cluster. The service principal should be of the form `name/role@REALM`, where:

   - `name` is the `gphdfs` service user name. This example uses `gphdfs`.
• **role** is the DNS-resolvable host name of a Greenplum cluster host (the output of the `hostname -f` command).
• **REALM** is the Kerberos realm, for example `LOCAL.DOMAIN`.

For example, the following commands add service principals for four Greenplum Database hosts, `mdw.example.com`, `smdw.example.com`, `sdw1.example.com`, and `sdw2.example.com`:

```
# kadmin.local -q "addprinc -randkey gphdfs/mdw.example.com@LOCAL.DOMAIN"
# kadmin.local -q "addprinc -randkey gphdfs/smdw.example.com@LOCAL.DOMAIN"
# kadmin.local -q "addprinc -randkey gphdfs/sdw1.example.com@LOCAL.DOMAIN"
# kadmin.local -q "addprinc -randkey gphdfs/sdw2.example.com@LOCAL.DOMAIN"
```

Create a principal for each Greenplum cluster host. Use the same principal name and realm, substituting the fully-qualified domain name for each host.

4. Generate a keytab file for each principal that you created (gpadmin and each gphdfs service principal). You can store the keytab files in any convenient location (this example uses the directory `/etc/security/keytabs`). You will deploy the service principal keytab files to their respective Greenplum host machines in a later step:

```
# kadmin.local -q "xst -k /etc/security/keytabs/gphdfs.service.keytab gpadmin@LOCAL.DOMAIN"
# kadmin.local -q "xst -k /etc/security/keytabs/mdw.service.keytab gpadmin/mdw gphdfs/mdw.example.com@LOCAL.DOMAIN"
# kadmin.local -q "xst -k /etc/security/keytabs/smdw.service.keytab gpadmin/smdw gphdfs/smdw.example.com@LOCAL.DOMAIN"
# kadmin.local -q "xst -k /etc/security/keytabs/sdw1.service.keytab gpadmin/sdw1 gphdfs/sdw1.example.com@LOCAL.DOMAIN"
# kadmin.local -q "xst -k /etc/security/keytabs/sdw2.service.keytab gpadmin/sdw2 gphdfs/sdw2.example.com@LOCAL.DOMAIN"
# kadmin.local -q "listprincs"
```

5. Change the ownership and permissions on `gphdfs.service.keytab` as follows:

```
# chown gpadmin:gpadmin /etc/security/keytabs/gphdfs.service.keytab
# chmod 440 /etc/security/keytabs/gphdfs.service.keytab
```

6. Copy the keytab file for `gpadmin@LOCAL.DOMAIN` to the Greenplum master host:

```
# scp /etc/security/keytabs/gphdfs.service.keytab mdw_fqdn:/home/gpadmin/gphdfs.service.keytab
```

7. Copy the keytab file for each service principal to its respective Greenplum host:

```
# scp /etc/security/keytabs/mdw.service.keytab mdw_fqdn:/home/gpadmin/mdw.service.keytab
# scp /etc/security/keytabs/smdw.service.keytab smdw_fqdn:/home/gpadmin/smdw.service.keytab
# scp /etc/security/keytabs/sdw1.service.keytab sdw1_fqdn:/home/gpadmin/sdw1.service.keytab
# scp /etc/security/keytabs/sdw2.service.keytab sdw2_fqdn:/home/gpadmin/sdw2.service.keytab
```

**Configuring gphdfs for Kerberos**

1. Edit the Hadoop `core-site.xml` client configuration file on all Greenplum cluster hosts. Enable service-level authorization for Hadoop by setting the `hadoop.security.authorization` property to `true`. For example:

```
<property>
  <name>hadoop.security.authorization</name>
```
2. Edit the yarn-site.xml client configuration file on all cluster hosts. Set the resource manager address and yarn Kerberos service principle. For example:

```xml
<property>
  <name>yarn.resourcemanager.address</name>
  <value>hostname:8032</value>
</property>
<pROPERTY>
  <name>yarn.resourcemanager.principal</name>
  <value>yarn/hostname@DOMAIN</value>
</property>
```

3. Edit the hdfs-site.xml client configuration file on all cluster hosts. Set properties to identify the NameNode Kerberos principals, the location of the Kerberos keytab file, and the principal it is for:

- `dfs.namenode.kerberos.principal` - the Kerberos principal name the gphdfs protocol will use for the NameNode, for example `gpadmin@LOCAL.DOMAIN`.
- `dfs.namenode.https.principal` - the Kerberos principal name the gphdfs protocol will use for the NameNode's secure HTTP server, for example `gpadmin@LOCAL.DOMAIN`.
- `com.emc.greenplum.gpdb.hdfsconnector.security.user.keytab.file` - the path to the keytab file for the Kerberos HDFS service, for example `/home/gpadmin/mdw.service.keytab`.
- `com.emc.greenplum.gpdb.hdfsconnector.security.user.name` - the gphdfs service principal for the host, for example `gphdfs/mdw.example.com@LOCAL.DOMAIN`.

For example:

```xml
<property>
  <name>dfs.namenode.kerberos.principal</name>
  <value>gphdfs/gpadmin@LOCAL.DOMAIN</value>
</property>
<property>
  <name>dfs.namenode.https.principal</name>
  <value>gphdfs/gpadmin@LOCAL.DOMAIN</value>
</property>
<property>
  <name>com.emc.greenplum.gpdb.hdfsconnector.security.user.keytab.file</name>
  <value>/home/gpadmin/gpadmin.hdfs.keytab</value>
</property>
<property>
  <name>com.emc.greenplum.gpdb.hdfsconnector.security.user.name</name>
  <value>gpadmin@LOCAL.DOMAIN</value>
</property>
```

**Testing Greenplum Database Access to HDFS**

Confirm that HDFS is accessible via Kerberos authentication on all hosts in the Greenplum cluster. For example, enter the following command to list an HDFS directory:

```
hdfs dfs -ls hdfs://namenode:8020
```

**Create a Readable External Table in HDFS**

Follow these steps to verify that you can create a readable external table in a Kerberized Hadoop cluster.
1. Create a comma-delimited text file, `test1.txt`, with contents such as the following:

```
25, Bill
19, Anne
32, Greg
27, Gloria
```

2. Persist the sample text file in HDFS:

```
hdfs dfs -put test1.txt hdfs://namenode:8020/tmp
```

3. Log in to Greenplum Database and create a readable external table that points to the `test1.txt` file in Hadoop:

```
CREATE EXTERNAL TABLE test_hdfs (age int, name text)
LOCATION('gphdfs://namenode:8020/tmp/test1.txt')
FORMAT 'text' (delimiter ',');
```

4. Read data from the external table:

```
SELECT * FROM test_hdfs;
```

**Create a Writable External Table in HDFS**

Follow these steps to verify that you can create a writable external table in a Kerberized Hadoop cluster. The steps use the `test_hdfs` readable external table created previously.

1. Log in to Greenplum Database and create a writable external table pointing to a text file in HDFS:

```
CREATE WRITABLE EXTERNAL TABLE test_hdfs2 (LIKE test_hdfs)
LOCATION ('gphdfs://namenode:8020/tmp/test2.txt')
FORMAT 'text' (DELIMITER ',');
```

2. Load data into the writable external table:

```
INSERT INTO test_hdfs2
SELECT * FROM test_hdfs;
```

3. Check that the file exists in HDFS:

```
hdfs dfs -ls hdfs://namenode:8020/tmp/test2.txt
```

4. Verify the contents of the external file:

```
hdfs dfs -cat hdfs://namenode:8020/tmp/test2.txt
```

**Troubleshooting HDFS with Kerberos**

**Forcing Classpaths**

If you encounter "class not found" errors when executing `SELECT` statements from `gphdfs` external tables, edit the `$GPHOME/lib/hadoop-env.sh` file and add the following lines towards the end of the file, before the `JAVA_LIBRARY_PATH` is set. Update the script on all of the cluster hosts.

```
if [ -d "/usr/hdp/current" ]; then
  for f in /usr/hdp/current/**/*.jar; do
    CLASSPATH=${CLASSPATH}:${f};
  done
fi
```
Enabling Kerberos Client Debug Messages
To see debug messages from the Kerberos client, edit the $GPHOME/lib/hadoop-env.sh client shell script on all cluster hosts and set the HADOOP_OPTS variable as follows:

```bash
export HADOOP_OPTS="-Djava.net.preffIPv4Stack=true -Dsun.security.krb5.debug=true $\{HADOOP_OPTS\}"
```

Adjusting JVM Process Memory on Segment Hosts
Each segment launches a JVM process when reading or writing an external table in HDFS. To change the amount of memory allocated to each JVM process, configure the GP_JAVA_OPT environment variable.

Edit the $GPHOME/lib/hadoop-env.sh client shell script on all cluster hosts.

For example:

```bash
export GP_JAVA_OPT=-Xmx1000m
```

Verify Kerberos Security Settings
Review the /etc/krb5.conf file:

- If AES256 encryption is not disabled, ensure that all cluster hosts have the JCE Unlimited Strength Jurisdiction Policy Files installed.
- Ensure all encryption types in the Kerberos keytab file match definitions in the krb5.conf file.

```bash
cat /etc/krb5.conf | egrep supported_enctypes
```

Test Connectivity on an Individual Segment Host
Follow these steps to test that a single Greenplum Database host can read HDFS data. This test method executes the Greenplum HDFSReader Java class at the command-line, and can help to troubleshoot connectivity problems outside of the database.

1. Save a sample data file in HDFS.

```bash
hdfs dfs -put test1.txt hdfs://namenode:8020/tmp
```

2. On the segment host to be tested, create an environment script, env.sh, like the following:

```bash
export JAVA_HOME=/usr/java/default
export HADOOP_HOME=/usr/lib/hadoop
export GP_HADOOP_CON_VERSION=hdp2
export GP_HADOOP_CON_JARDIR=/usr/lib/hadoop
```

3. Source all environment scripts:

```bash
source /usr/local/greenplum-db/greenplum_path.sh
source env.sh
source $GPHOME/lib/hadoop-env.sh
```

4. Test the Greenplum Database HDFS reader:

```bash
java com.emc.greenplum.gpdb.hdfsconnector.HDFSReader 0 32 TEXT hdp2
gphdfs://namenode:8020/tmp/test1.txt
```
Security Best Practices

Describes basic security best practices that you should follow to ensure the highest level of system security.

In the default Greenplum Database security configuration:

- Only local connections are allowed.
- Basic authentication is configured for the superuser (gpadmin).
- The superuser is authorized to do anything.
- Only database role passwords are encrypted.

System User (gpadmin)

Secure and limit access to the gpadmin system user.

Greenplum requires a UNIX user id to install and initialize the Greenplum Database system. This system user is referred to as gpadmin in the Greenplum documentation. The gpadmin user is the default database superuser in Greenplum Database, as well as the file system owner of the Greenplum installation and its underlying data files. The default administrator account is fundamental to the design of Greenplum Database. The system cannot run without it, and there is no way to limit the access of the gpadmin user id.

The gpadmin user can bypass all security features of Greenplum Database. Anyone who logs on to a Greenplum host with this user id can read, alter, or delete any data, including system catalog data and database access rights. Therefore, it is very important to secure the gpadmin user id and only allow essential system administrators access to it.

Administrators should only log in to Greenplum as gpadmin when performing certain system maintenance tasks (such as upgrade or expansion).

Database users should never log on as gpadmin, and ETL or production workloads should never run as gpadmin.

Superusers

Roles granted the SUPERUSER attribute are superusers. Superusers bypass all access privilege checks and resource queues. Only system administrators should be given superuser rights.

See "Altering Role Attributes" in the Greenplum Database Administrator Guide.

Login Users

Assign a distinct role to each user who logs in and set the LOGIN attribute.

For logging and auditing purposes, each user who is allowed to log in to Greenplum Database should be given their own database role. For applications or web services, consider creating a distinct role for each application or service. See "Creating New Roles (Users)" in the Greenplum Database Administrator Guide.

Each login role should be assigned to a single, non-default resource queue.

Groups

Use groups to manage access privileges.

Create a group for each logical grouping of object/access permissions.

Every login user should belong to one or more roles. Use the GRANT statement to add group access to a role. Use the REVOKE statement to remove group access from a role.
The LOGIN attribute should not be set for group roles.
See "Creating Groups (Role Membership)" in the Greenplum Database Administrator Guide.

Object Privileges

Only the owner and superusers have full permissions to new objects. Permission must be granted to allow other rules (users or groups) to access objects. Each type of database object has different privileges that may be granted. Use the GRANT statement to add a permission to a role and the REVOKE statement to remove the permission.

You can change the owner of an object using the REASSIGN OWNED BY statement. For example, to prepare to drop a role, change the owner of the objects that belong to the role. Use the DROP OWNED BY to drop objects, including dependent objects, that are owned by a role.

Schemas can be used to enforce an additional layer of object permissions checking, but schema permissions do not override object privileges set on objects contained within the schema.

Operating System Users and File System

To protect the network from intrusion, system administrators should verify the passwords used within an organization are sufficiently strong. The following recommendations can strengthen a password:

- Minimum password length recommendation: At least 9 characters. MD5 passwords should be 15 characters or longer.
- Mix upper and lower case letters.
- Mix letters and numbers.
- Include non-alphanumeric characters.
- Pick a password you can remember.

The following are recommendations for password cracker software that you can use to determine the strength of a password:

- John The Ripper. A fast and flexible password cracking program. It allows the use of multiple word lists and is capable of brute-force password cracking. It is available online at http://www.openwall.com/john/.
- Crack. Perhaps the most well-known password cracking software, Crack is also very fast, though not as easy to use as John The Ripper. It can be found online at https://dropsafe.crypticide.com/alecm/software/crack/c50-faq.html.

The security of the entire system depends on the strength of the root password. This password should be at least 12 characters long and include a mix of capitalized letters, lowercase letters, special characters, and numbers. It should not be based on any dictionary word.

Password expiration parameters should be configured.

Ensure the following line exists within the file /etc/libuser.conf under the [import] section.

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

Ensure no lines in the [userdefaults] section begin with the following text, as these words override settings from /etc/login.defs:

- LU_SHADOWMAX
- LU_SHADOWMIN
- LU_SHADOWWARNING

Ensure the following command produces no output. Any accounts listed by running this command should be locked.

```
grep "^+:" /etc/passwd /etc/shadow /etc/group
```
Note: We strongly recommend that customers change their passwords after initial setup.

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

Find all the files that are world-writable and that do not have their sticky bits set.

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

Set the sticky bit (# chmod +t {dir}) for all the directories that result from running the previous command.

Find all the files that are world-writable and fix each file listed.

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

Set the right permissions (# chmod o-w {file}) for all the files generated by running the aforementioned command.

Find all the files that do not belong to a valid user or group and either assign an owner or remove the file, as appropriate.

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

Find all the directories that are world-writable and ensure they are owned by either root or a system account (assuming only system accounts have a User ID lower than 500). If the command generates any output, verify the assignment is correct or reassign it to root.

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

Authentication settings such as password quality, password expiration policy, password reuse, password retry attempts, and more can be configured using the Pluggable Authentication Modules (PAM) framework. PAM looks in the directory `/etc/pam.d` for application-specific configuration information. Running `authconfig` or `system-config-authentication` will re-write the PAM configuration files, destroying any manually made changes and replacing them with system defaults.

The default `pam_cracklib` PAM module provides strength checking for passwords. To configure `pam_cracklib` to require at least one uppercase character, lowercase character, digit, and special character, as recommended by the U.S. Department of Defense guidelines, edit the file `/etc/pam.d/system-auth` to include the following parameters in the line corresponding to password requisite `pam_cracklib.so try_first_pass`.

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

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

These parameters can be set to reflect your security policy requirements. Note that the password restrictions are not applicable to the root password.

The pam_tally2 PAM module provides the capability to lock out user accounts after a specified number of failed login attempts. To enforce password lockout, edit the file /etc/pam.d/system-auth to include the following lines:

- The first of the auth lines should include:

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

- The first of the account lines should include:

  account required pam_tally2.so

Here, the deny parameter is set to limit the number of retries to 5 and the unlock_time has been set to 900 seconds to keep the account locked for 900 seconds before it is unlocked. These parameters may be configured appropriately to reflect your security policy requirements. A locked account can be manually unlocked using the pam_tally2 utility:

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

You can use PAM to limit the reuse of recent passwords. The remember option for the pam_unix module can be set to remember the recent passwords and prevent their reuse. To accomplish this, edit the appropriate line in /etc/pam.d/system-auth to include the remember option.

For example:

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

You can set the number of previous passwords to remember to appropriately reflect your security policy requirements.

  cd /etc
  chown root:root passwd shadow group gshadow
  chmod 644 passwd group
  chmod 400 shadow gshadow
Chapter 11

Greenplum Database Best Practices

A best practice is a method or technique that has consistently shown results superior to those achieved with other means. Best practices are found through experience and are proven to reliably lead to a desired result. Best practices are a commitment to use any product correctly and optimally, by leveraging all the knowledge and expertise available to ensure success.

This document does not teach you how to use Greenplum Database features. Links are provided to other relevant parts of the Greenplum Database documentation for information on how to use and implement specific Greenplum Database features. This document addresses the most important best practices to follow when designing, implementing, and using Greenplum Database.

It is not the intent of this document to cover the entire product or compendium of features, but rather to provide a summary of what matters most in Greenplum Database. This document does not address edge use cases. While edge use cases can further leverage and benefit from Greenplum Database features, they require a proficient knowledge and expertise with these features, as well as a deep understanding of your environment, including SQL access, query execution, concurrency, workload, and other factors.

By mastering these best practices, you will increase the success of your Greenplum Database clusters in the areas of maintenance, support, performance, and scalability.
Best Practices Summary

A summary of best practices for Greenplum Database.

Data Model
Greenplum Database is an analytical MPP shared-nothing database. This model is significantly different from a highly normalized/transactional SMP database. Because of this, the following best practices are recommended.

- Greenplum Database performs best with a denormalized schema design suited for MPP analytical processing for example, Star or Snowflake schema, with large fact tables and smaller dimension tables.
- Use the same data types for columns used in joins between tables.

Heap vs. Append-Optimized Storage
- Use heap storage for tables and partitions that will receive iterative batch and singleton UPDATE, DELETE, and INSERT operations.
- Use heap storage for tables and partitions that will receive concurrent UPDATE, DELETE, and INSERT operations.
- Use append-optimized storage for tables and partitions that are updated infrequently after the initial load and have subsequent inserts only performed in large batch operations.
- Avoid performing singleton INSERT, UPDATE, or DELETE operations on append-optimized tables.
- Avoid performing concurrent batch UPDATE or DELETE operations on append-optimized tables. Concurrent batch INSERT operations are acceptable.

Row vs. Column Oriented Storage
- Use row-oriented storage for workloads with iterative transactions where updates are required and frequent inserts are performed.
- Use row-oriented storage when selects against the table are wide.
- Use row-oriented storage for general purpose or mixed workloads.
- Use column-oriented storage where selects are narrow and aggregations of data are computed over a small number of columns.
- Use column-oriented storage for tables that have single columns that are regularly updated without modifying other columns in the row.

Compression
- Use compression on large append-optimized and partitioned tables to improve I/O across the system.
- Set the column compression settings at the level where the data resides.
- Balance higher levels of compression with the time and CPU cycles needed to compress and uncompress data.

Distributions
- Explicitly define a column or random distribution for all tables. Do not use the default.
- Use a single column that will distribute data across all segments evenly.
- Do not distribute on columns that will be used in the `WHERE` clause of a query.
- Do not distribute on dates or timestamps.
- Never distribute and partition tables on the same column.
- Achieve local joins to significantly improve performance by distributing on the same column for large tables commonly joined together.
- To ensure there is no data skew, validate that data is evenly distributed after the initial load and after incremental loads.

See *Distributions*.

**Memory Management**

- Set `vm.overcommit_memory` to 2.
- Do not configure the OS to use huge pages.
- Use `gp_vmem_protect_limit` to set the maximum memory that the instance can allocate for *all* work being done in each segment database.
- You can use `gp_vmem_protect_limit` by calculating:
  
  \[ gp\_vmem = \frac{((SWAP + RAM) - (7.5GB + 0.05 \times RAM))}{1.7} \]

  *where* `SWAP` *is the host's swap space in GB, and* `RAM` *is the host's RAM in GB*

- Use `gp_vmem_protect_limit` to set the maximum number of primary segments that could be running on a host when mirror segments are activated due to a host or segment failure
- \[ gp\_vmem\_protect\_limit = \frac{gp\_vmem}{acting\_primary\_segments} \]

  Convert to MB to set the value of the configuration parameter.

- In a scenario where a large number of workfiles are generated, calculate the `gp_vmem` factor with this formulat to account for the workfiles:
  
  \[ gp\_vmem = \frac{((SWAP + RAM) - (7.5GB + 0.05 \times RAM - (300KB \times total\_#\_workfiles)))}{1.7} \]

- Never set `gp_vmem_protect_limit` too high or larger than the physical RAM on the system.
- Use the calculated `gp_vmem` value to calculate the setting for the `vm.overcommit_ratio` operating system parameter:
  
  \[ vm\_overcommit\_ratio = \frac{(RAM - 0.026 \times gp\_vmem)}{RAM} \]

- Use `statement_mem` to allocate memory used for a query per segment db.
- Use resource queues to set both the numbers of active queries (ACTIVE_STATEMENTS) and the amount of memory (MEMORY_LIMIT) that can be utilized by queries in the queue.
- Associate all users with a resource queue. Do not use the default queue.
- Set PRIORITY to match the real needs of the queue for the workload and time of day. Avoid using MAX priority.
- Ensure that resource queue memory allocations do not exceed the setting for `gp_vmem_protect_limit`.
- Dynamically update resource queue settings to match daily operations flow.

See *Memory and Resource Management*.

**Partitioning**

- Partition large tables only. Do not partition small tables.
• Use partitioning only if partition elimination (partition pruning) can be achieved based on the query criteria.
• Choose range partitioning over list partitioning.
• Partition the table based on a commonly-used column, such as a date column.
• Never partition and distribute tables on the same column.
• Do not use default partitions.
• Do not use multi-level partitioning; create fewer partitions with more data in each partition.
• Validate that queries are selectively scanning partitioned tables (partitions are being eliminated) by examining the query `EXPLAIN` plan.
• Do not create too many partitions with column-oriented storage because of the total number of physical files on every segment: \[ \text{physical files} = \text{segments} \times \text{columns} \times \text{partitions} \]

See *Partitioning*.

**Indexes**

• In general indexes are not needed in Greenplum Database.
• Create an index on a single column of a columnar table for drill-through purposes for high cardinality tables that require queries with high selectivity.
• Do not index columns that are frequently updated.
• Consider dropping indexes before loading data into a table. After the load, re-create the indexes for the table.
• Create selective B-tree indexes.
• Do not create bitmap indexes on columns that are updated.
• Avoid using bitmap indexes for unique columns, very high or very low cardinality data. Bitmap indexes perform best when the column has a low cardinality—100 to 100,000 distinct values.
• Do not use bitmap indexes for transactional workloads.
• In general do not index partitioned tables. If indexes are needed, the index columns must be different than the partition columns.

See *Indexes*.

**Resource Queues**

• Use resource queues to manage the workload on the cluster.
• Associate all roles with a user-defined resource queue.
• Use the `ACTIVE_STATEMENTS` parameter to limit the number of active queries that members of the particular queue can run concurrently.
• Use the `MEMORY_LIMIT` parameter to control the total amount of memory that queries running through the queue can utilize.
• Alter resource queues dynamically to match the workload and time of day.

See *Configuring Resource Queues*.

**Monitoring and Maintenance**

• Implement the “Recommended Monitoring and Maintenance Tasks” in the *Greenplum Database Administrator Guide*.
• Run `gpcheckperf` at install time and periodically thereafter, saving the output to compare system performance over time.
• Use all the tools at your disposal to understand how your system behaves under different loads.
• Examine any unusual event to determine the cause.
• Monitor query activity on the system by running explain plans periodically to ensure the queries are running optimally.
• Review plans to determine whether indexes are being used and partition elimination is occurring as expected.
• Know the location and content of system log files and monitor them on a regular basis, not just when problems arise.

See System Monitoring and Maintenance, Query Profiling and Monitoring Greenplum Database Log Files.

**ANALYZE**

• Determine if analyzing the database is actually needed. Analyzing is not needed if `gp_autostats_mode` is set to `on_no_stats` (the default) and the table is not partitioned.
• Use `analyzedb` in preference to `ANALYZE` when dealing with large sets of tables, as it does not require analyzing the entire database. The `analyzedb` utility updates statistics data for the specified tables incrementally and concurrently. For append-optimized tables, `analyzedb` updates statistics incrementally only if the statistics are not current. For heap tables, statistics are always updated. `ANALYZE` does not update the table metadata that the `analyzedb` utility uses to determine whether table statistics are up to date.
• Selectively run `ANALYZE` at the table level when needed.
• Always run `ANALYZE` after `INSERT`, `UPDATE`, and `DELETE` operations that significantly change the underlying data.
• Always run `ANALYZE` after `CREATE_INDEX` operations.
• If `ANALYZE` on very large tables takes too long, run `ANALYZE` only on the columns used in a join condition, WHERE clause, SORT, GROUP BY, or HAVING clause.
• When dealing with large sets of tables, use `analyzedb` instead of `ANALYZE`.

See Updating Statistics with ANALYZE.

**Vacuum**

• Run `VACUUM` after large `UPDATE` and `DELETE` operations.
• Do not run `VACUUM FULL`. Instead run a `CREATE TABLE...AS` operation, then rename and drop the original table.
• Frequently run `VACUUM` on the system catalogs to avoid catalog bloat and the need to run `VACUUM FULL` on catalog tables.
• Never kill `VACUUM` on catalog tables.

See Managing Bloat in the Database.

**Loading**

• Maximize the parallelism as the number of segments increase.
• Spread the data evenly across as many ETL nodes as possible.
  • Split very large data files into equal parts and spread the data across as many file systems as possible.
  • Run two `gpfdist` instances per file system.
  • Run `gpfdist` on as many interfaces as possible.
  • Use `gp_external_max_segs` to control the number of segments that will request data from the `gpfdist` process.
  • Always keep `gp_external_max_segs` and the number of `gpfdist` processes an even factor.
• Always drop indexes before loading into existing tables and re-create the index after loading.
• Run `VACUUM` after load errors to recover space.

See Loading Data.
gptransfer

- Avoid using the `--full` or `--schema-only` options. Instead, copy schemas to the destination database using a different method, and then transfer the table data.
- Drop indexes before transferring tables and recreate them when the transfer is complete.
- Transfer smaller tables to the destination database using the SQL `COPY` command.
- Transfer larger tables in batches using `gptransfer`.
- Test running `gptransfer` before performing a production migration. Experiment with the `--batch-size` and `--sub-batch-size` options to obtain maximum parallelism. Determine proper batching of tables for iterative `gptransfer` runs.
- Use only fully qualified table names. Periods (.), whitespace, quotes (') and double quotes (") in table names may cause problems.
- If you use the `--validation` option to validate the data after transfer, be sure to also use the `-x` option to place an exclusive lock on the source table.
- Ensure any roles, functions, and resource queues are created in the destination database. These objects are not transferred when you use the `gptransfer -t` option.
- Copy the `postgres.conf` and `pg_hba.conf` configuration files from the source to the destination cluster.
- Install needed extensions in the destination database with `gppkg`.

See [Migrating Data with Gptransfer](#).

**Security**

- Secure the `gpadmin` user id and only allow essential system administrators access to it.
- Administrators should only log in to Greenplum as `gpadmin` when performing certain system maintenance tasks (such as upgrade or expansion).
- Limit users who have the `SUPERUSER` role attribute. Roles that are superusers bypass all access privilege checks in Greenplum Database, as well as resource queuing. Only system administrators should be given superuser rights. See "Altering Role Attributes" in the *Greenplum Database Administrator Guide*.
- Database users should never log on as `gpadmin`, and ETL or production workloads should never run as `gpadmin`.
- Assign a distinct Greenplum Database role to each user, application, or service that logs in.
- For applications or web services, consider creating a distinct role for each application or service.
- Use groups to manage access privileges.
- Protect the root password.
- Enforce a strong password password policy for operating system passwords.
- Ensure that important operating system files are protected.

See [Security](#).

**Encryption**

- Encrypting and decrypting data has a performance cost; only encrypt data that requires encryption.
- Do performance testing before implementing any encryption solution in a production system.
- Server certificates in a production Greenplum Database system should be signed by a certificate authority (CA) so that clients can authenticate the server. The CA may be local if all clients are local to the organization.
- Client connections to Greenplum Database should use SSL encryption whenever the connection goes through an insecure link.
- A symmetric encryption scheme, where the same key is used to both encrypt and decrypt, has better performance than an asymmetric scheme and should be used when the key can be shared safely.
• Use cryptographic functions to encrypt data on disk. The data is encrypted and decrypted in the database process, so it is important to secure the client connection with SSL to avoid transmitting unencrypted data.
• Use the gpfdists protocol to secure ETL data as it is loaded into or unloaded from the database.

See Encrypting Data and Database Connections

High Availability

Note: The following guidelines apply to actual hardware deployments, but not to public cloud-based infrastructure, where high availability solutions may already exist.

• Use a hardware RAID storage solution with 8 to 24 disks.
• Use RAID 1, 5, or 6 so that the disk array can tolerate a failed disk.
• Configure a hot spare in the disk array to allow rebuild to begin automatically when disk failure is detected.
• Protect against failure of the entire disk array and degradation during rebuilds by mirroring the RAID volume.
• Monitor disk utilization regularly and add additional space when needed.
• Monitor segment skew to ensure that data is distributed evenly and storage is consumed evenly at all segments.
• Set up a standby master instance to take over if the primary master fails.
• Plan how to switch clients to the new master instance when a failure occurs, for example, by updating the master address in DNS.
• Set up monitoring to send notifications in a system monitoring application or by email when the primary fails.
• Set up mirrors for all segments.
• Locate primary segments and their mirrors on different hosts to protect against host failure.
• Recover failed segments promptly, using the gprecoverseg utility, to restore redundancy and return the system to optimal balance.
• Configure Greenplum Database to send SNMP notifications to your network monitor.
• Set up email notification in the $MASTER_DATA_DIRECTORY/postgresql.conf configuration file so that the Greenplum system can email administrators when a critical issue is detected.
• Consider a Dual Cluster configuration to provide an additional level of redundancy and additional query processing throughput.
• Backup Greenplum databases regularly unless the data is easily restored from sources.
• Use incremental backups if heap tables are relatively small and few append-optimized or column-oriented partitions are modified between backups.
• If backups are saved to local cluster storage, move the files to a safe, off-cluster location when the backup is complete.
• If backups are saved to NFS mounts, use a scale-out NFS solution such as Dell EMC Isilon to prevent IO bottlenecks.
• Consider using Greenplum integration to stream backups to the Dell EMC Data Domain or Veritas NetBackup enterprise backup platforms.

See High Availability.
System Configuration

Requirements and best practices for system administrators who are configuring Greenplum Database cluster hosts.

Configuration of the Greenplum Database cluster is usually performed as root.

File System

XFS is the file system used for Greenplum Database data directories. XFS should be mounted with the following mount options:

```
rw,noatime,nobARRIER,nodev,inode64,allocsize=16m
```

Port Configuration

Set up `ip_local_port_range` so it does not conflict with the Greenplum Database port ranges. For example:

```
net.ipv4.ip_local_port_range = 3000         65535
PORT_BASE=2000
MIRROR_PORT_BASE=2100
REPLICATION_PORT_BASE=2200
MIRROR_REPLICATION_PORT_BASE=2300
```

I/O Configuration

Set the blockdev read-ahead size to 16384 on the devices that contain data directories.

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

The deadline IO scheduler should be set for all data directory devices.

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

The maximum number of OS files and processes should be increased in the `/etc/security/limits.conf` file.

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

Enable core files output to a known location and make sure `limits.conf` allows core files.

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

OS Memory Configuration

The Linux `sysct1 vm.overcommit_memory` and `vm.overcommit_ratio` variables affect how the operating system manages memory allocation. These variables should be set as follows:
vm.overcommit_memory determines the method the OS uses for determining how much memory can be allocated to processes. This should be always set to 2, which is the only safe setting for the database.

**Note:** For information on configuration of overcommit memory, refer to:
- https://en.wikipedia.org/wiki/Memory_overcommitment

vm.overcommit_ratio is the percent of RAM that is used for application processes. The default is 50 on Red Hat Enterprise Linux. See *Segment Memory Configuration* for a formula to calculate an optimal value.

Do not enable huge pages in the operating system.

See also *Memory and Resource Management*.

## Shared Memory Settings

Greenplum Database uses shared memory to communicate between *postgres* processes that are part of the same *postgres* instance. The following shared memory settings should be set in *sysctl* and are rarely modified.

```plaintext
kernel.shmmax = 500000000
kernel.shmmni = 4096
kernel.shmall = 4000000000
```

## Validate the Operating System

Run `gpcheck` to validate the operating system configuration. See `gpcheck` in the *Greenplum Database Utility Guide*.

## Number of Segments per Host

Determining the number of segments to execute on each segment host has immense impact on overall system performance. The segments share the host's CPU cores, memory, and NICs with each other and with other processes running on the host. Over-estimating the number of segments a server can accommodate is a common cause of suboptimal performance.

The factors that must be considered when choosing how many segments to run per host include the following:

- Number of cores
- Amount of physical RAM installed in the server
- Number of NICs
- Amount of storage attached to server
- Mixture of primary and mirror segments
- ETL processes that will run on the hosts
- Non-Greenplum processes running on the hosts

## Segment Memory Configuration

The `gp_vmem_protect_limit` server configuration parameter specifies the amount of memory that all active *postgres* processes for a single segment can consume at any given time. Queries that exceed this amount will fail. Use the following calculations to estimate a safe value for `gp_vmem_protect_limit`.

1. Calculate `gp_vmem`, the host memory available to Greenplum Database, using this formula:

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

   where `SWAP` is the host's swap space in GB and `RAM` is the RAM installed on the host in GB.
2. Calculate `max_acting_primary_segments`. This is the maximum number of primary segments that can be running on a host when mirror segments are activated due to a segment or host failure on another host in the cluster. With mirrors arranged in a 4-host block with 8 primary segments per host, for example, a single segment host failure would activate two or three mirror segments on each remaining host in the failed host's block. The `max_acting_primary_segments` value for this configuration is 11 (8 primary segments plus 3 mirrors activated on failure).

3. Calculate `gp_vmem_protect_limit` by dividing the total Greenplum Database memory by the maximum number of acting primaries:

   \[
   gp\_vmem\_protect\_limit = \frac{gp\_vmem}{max\_acting\_primary\_segments}
   \]

   Convert to megabytes to find the value to set for the `gp_vmem_protect_limit` system configuration parameter.

   For scenarios where a large number of workfiles are generated, adjust the calculation for `gp_vmem` to account for the workfiles:

   \[
   gp\_vmem = \frac{((SWAP + RAM) - (7.5GB + 0.05 \times RAM - (300KB \times total\_#\_workfiles)))}{1.7}
   \]

   For information about monitoring and managing workfile usage, see the *Greenplum Database Administrator Guide*.

   You can calculate the value of the `vm.overcommit_ratio` operating system parameter from the value of `gp_vmem`:

   \[
   vm\_overcommit\_ratio = \frac{(RAM - 0.026 \times gp\_vmem)}{RAM}
   \]

   See *OS Memory Configuration* for more about `vm.overcommit_ratio`.

   See also *Memory and Resource Management*.

**Statement Memory Configuration**

The `statement_mem` server configuration parameter is the amount of memory to be allocated to any single query in a segment database. If a statement requires additional memory it will spill to disk. Calculate the value for `statement_mem` with the following formula:

\[
(statement\_mem = \frac{(gp\_vmem\_protect\_limit \times .9)}{max\_expected\_concurrent\_queries})
\]

For example, for 40 concurrent queries with `gp_vmem_protect_limit` set to 8GB (8192MB), the calculation for `statement_mem` would be:

\[
(8192MB \times .9) / 40 = 184MB
\]

Each query would be allowed 184MB of memory before it must spill to disk.

To increase `statement_mem` safely you must either increase `gp_vmem_protect_limit` or reduce the number of concurrent queries. To increase `gp_vmem_protect_limit`, you must add physical RAM and/or swap space, or reduce the number of segments per host.

Note that adding segment hosts to the cluster cannot help out-of-memory errors unless you use the additional hosts to decrease the number of segments per host.

Spill files are created when there is not enough memory to fit all the mapper output, usually when 80% of the buffer space is occupied.

Also, see *Resource Management* for best practices for managing query memory using resource queues.
Spill File Configuration

Greenplum Database creates spill files (also called workfiles) on disk if a query is allocated insufficient memory to execute in memory. A single query can create no more than 100,000 spill files, by default, which is sufficient for the majority of queries.

You can control the maximum number of spill files created per query and per segment with the configuration parameter `gp_workfile_limit_files_per_query`. Set the parameter to 0 to allow queries to create an unlimited number of spill files. Limiting the number of spill files permitted prevents runaway queries from disrupting the system.

A query could generate a large number of spill files if not enough memory is allocated to it or if data skew is present in the queried data. If a query creates more than the specified number of spill files, Greenplum Database returns this error:

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

Before raising the `gp_workfile_limit_files_per_query`, try reducing the number of spill files by changing the query, changing the data distribution, or changing the memory configuration.

The `gp_toolkit` schema includes views that allow you to see information about all the queries that are currently using spill files. This information can be used for troubleshooting and for tuning queries:

- **The `gp_workfile_entries` view** contains one row for each operator using disk space for workfiles on a segment at the current time. See *How to Read Explain Plans* for information about operators.
- **The `gp_workfile_usage_per_query` view** contains one row for each query using disk space for workfiles on a segment at the current time.
- **The `gp_workfile_usage_per_segment` view** contains one row for each segment. Each row displays the total amount of disk space used for workfiles on the segment at the current time.

See the *Greenplum Database Reference Guide* for descriptions of the columns in these views.

The `gp_workfile_compress_algorithm` configuration parameter specifies a compression algorithm to apply to spill files. It can have the value `none` (the default) or `zlib`. Setting this parameter to `zlib` can improve performance when spill files are used.

A query could generate a large number of spill files if not enough memory is allocated to it or if data skew is present in the queried data. If a query creates more than the specified number of spill files, Greenplum Database returns this error:

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

Before raising the `gp_workfile_limit_files_per_query`, try reducing the number of spill files by changing the query, changing the data distribution, or changing the memory configuration.
Schema Design

Best practices for designing Greenplum Database schemas.

Greenplum Database is an analytical, shared-nothing database, which is much different than a highly normalized, transactional SMP database. Greenplum Database performs best with a denormalized schema design suited for MPP analytical processing, a star or snowflake schema, with large centralized fact tables connected to multiple smaller dimension tables.

Data Types

Use Types Consistently

Use the same data types for columns used in joins between tables. If the data types differ, Greenplum Database must dynamically convert the data type of one of the columns so the data values can be compared correctly. With this in mind, you may need to increase the data type size to facilitate joins to other common objects.

Choose Data Types that Use the Least Space

You can increase database capacity and improve query execution by choosing the most efficient data types to store your data.

Use TEXT or VARCHAR rather than CHAR. There are no performance differences among the character data types, but using TEXT or VARCHAR can decrease the storage space used.

Use the smallest numeric data type that will accommodate your data. Using BIGINT for data that fits in INT or SMALLINT wastes storage space.

Storage Model

Greenplum Database provides an array of storage options when creating tables. It is very important to know when to use heap storage versus append-optimized (AO) storage, and when to use row-oriented storage versus column-oriented storage. The correct selection of heap versus AO and row versus column is extremely important for large fact tables, but less important for small dimension tables.

The best practices for determining the storage model are:

1. Design and build an insert-only model, truncating a daily partition before load.
2. For large partitioned fact tables, evaluate and use optimal storage options for different partitions. One storage option is not always right for the entire partitioned table. For example, some partitions can be row-oriented while others are column-oriented.
3. When using column-oriented storage, every column is a separate file on every Greenplum Database segment. For tables with a large number of columns consider columnar storage for data often accessed (hot) and row-oriented storage for data not often accessed (cold).
4. Storage options should be set at the partition level.
5. Compress large tables to improve I/O performance and to make space in the cluster.

Heap Storage or Append-Optimized Storage

Heap storage is the default model, and is the model PostgreSQL uses for all database tables. Use heap storage for tables and partitions that will receive iterative UPDATE, DELETE, and singleton INSERT operations. Use heap storage for tables and partitions that will receive concurrent UPDATE, DELETE, and INSERT operations.
Use append-optimized storage for tables and partitions that are updated infrequently after the initial load and have subsequent insert operations performed only in batch operations. Avoid performing singleton INSERT, UPDATE, or DELETE operations on append-optimized tables. Concurrent batch INSERT operations are acceptable, but never perform concurrent batch UPDATE or DELETE operations.

The append-optimized storage model is inappropriate for frequently updated tables, because space occupied by rows that are updated and deleted in append-optimized tables is not recovered and reused as efficiently as with heap tables. Append-optimized storage is intended for large tables that are loaded once, updated infrequently, and queried frequently for analytical query processing.

### Row or Column Orientation

Row orientation is the traditional way to store database tuples. The columns that comprise a row are stored on disk contiguously, so that an entire row can be read from disk in a single I/O.

Column orientation stores column values together on disk. A separate file is created for each column. If the table is partitioned, a separate file is created for each column and partition. When a query accesses only a small number of columns in a column-oriented table with many columns, the cost of I/O is substantially reduced compared to a row-oriented table; any columns not referenced do not have to be retrieved from disk.

Row-oriented storage is recommended for transactional type workloads with iterative transactions where updates are required and frequent inserts are performed. Use row-oriented storage when selects against the table are wide, where many columns of a single row are needed in a query. If the majority of columns in the SELECT list or WHERE clause is selected in queries, use row-oriented storage. Use row-oriented storage for general purpose or mixed workloads, as it offers the best combination of flexibility and performance.

Column-oriented storage is optimized for read operations but it is not optimized for write operations; column values for a row must be written to different places on disk. Column-oriented tables can offer optimal query performance on large tables with many columns where only a small subset of columns are accessed by the queries.

Another benefit of column orientation is that a collection of values of the same data type can be stored together in less space than a collection of mixed type values, so column-oriented tables use less disk space (and consequently less disk I/O) than row-oriented tables. Column-oriented tables also compress better than row-oriented tables.

Use column-oriented storage for data warehouse analytic workloads where selects are narrow or aggregations of data are computed over a small number of columns. Use column-oriented storage for tables that have single columns that are regularly updated without modifying other columns in the row. Reading a complete row in a wide columnar table requires more time than reading the same row from a row-oriented table. It is important to understand that each column is a separate physical file on every segment in Greenplum Database.

### Compression

Greenplum Database offers a variety of options to compress append-optimized tables and partitions. Use compression to improve I/O across the system by allowing more data to be read with each disk read operation. The best practice is to set the column compression settings at the partition level.

Note that new partitions added to a partitioned table do not automatically inherit compression defined at the table level; you must specifically define compression when you add new partitions.

Run-length encoding (RLE) compression provides the best levels of compression. Higher levels of compression usually result in more compact storage on disk, but require additional time and CPU cycles when compressing data on writes and uncompressing on reads. Sorting data, in combination with the various compression options, can achieve the highest level of compression.

Data compression should never be used for data that is stored on a compressed file system.
Test different compression types and ordering methods to determine the best compression for your specific data. For example, you might start zlib compression at level 4 or 5 and adjust for best results. RLE compression works best with files that contain repetitive data.

**Distributions**

An optimal distribution that results in evenly distributed data is the most important factor in Greenplum Database. In an MPP shared nothing environment overall response time for a query is measured by the completion time for all segments. The system is only as fast as the slowest segment. If the data is skewed, segments with more data will take more time to complete, so every segment must have an approximately equal number of rows and perform approximately the same amount of processing. Poor performance and out of memory conditions may result if one segment has significantly more data to process than other segments.

Consider the following best practices when deciding on a distribution strategy:

- Explicitly define a column or random distribution for all tables. Do not use the default.
- Ideally, use a single column that will distribute data across all segments evenly.
- Do not distribute on columns that will be used in the *WHERE* clause of a query.
- Do not distribute on dates or timestamps.
- The distribution key column data should contain unique values or very high cardinality.
- If a single column cannot achieve an even distribution, use a multi-column distribution key with a maximum of two columns. Additional column values do not typically yield a more even distribution and they require additional time in the hashing process.
- If a two-column distribution key cannot achieve an even distribution of data, use a random distribution. Multi-column distribution keys in most cases require motion operations to join tables, so they offer no advantages over a random distribution.

Greenplum Database random distribution is not round-robin, so there is no guarantee of an equal number of records on each segment. Random distributions typically fall within a target range of less than ten percent variation.

Optimal distributions are critical when joining large tables together. To perform a join, matching rows must be located together on the same segment. If data is not distributed on the same join column, the rows needed from one of the tables are dynamically redistributed to the other segments. In some cases a broadcast motion, in which each segment sends its individual rows to all other segments, is performed rather than a redistribution motion, where each segment rehashes the data and sends the rows to the appropriate segments according to the hash key.

**Local (Co-located) Joins**

Using a hash distribution that evenly distributes table rows across all segments and results in local joins can provide substantial performance gains. When joined rows are on the same segment, much of the processing can be accomplished within the segment instance. These are called *local* or *co-located* joins. Local joins minimize data movement; each segment operates independently of the other segments, without network traffic or communications between segments.

To achieve local joins for large tables commonly joined together, distribute the tables on the same column. Local joins require that both sides of a join be distributed on the same columns (and in the same order) and that all columns in the distribution clause are used when joining tables. The distribution columns must also be the same data type—although some values with different data types may appear to have the same representation, they are stored differently and hash to different values, so they are stored on different segments.
**Data Skew**

Data skew is often the root cause of poor query performance and out of memory conditions. Skewed data affects scan (read) performance, but it also affects all other query execution operations, for instance, joins and group by operations.

It is very important to **validate** distributions to **ensure** that data is evenly distributed after the initial load. It is equally important to **continue** to validate distributions after incremental loads.

The following query shows the number of rows per segment as well as the variance from the minimum and maximum numbers of rows:

```sql
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;
```

The **gp_toolkit** schema has two views that you can use to check for skew.

- The **gp_toolkit.gp_skew_coefficients** view shows data distribution skew by calculating the coefficient of variation (CV) for the data stored on each segment. The **skccoeff** column shows the coefficient of variation (CV), which is calculated as the standard deviation divided by the average. It takes into account both the average and variability around the average of a data series. The lower the value, the better. Higher values indicate greater data skew.

- The **gp_toolkit.gp_skew_idle_fractions** view shows data distribution skew by calculating the percentage of the system that is idle during a table scan, which is an indicator of computational skew. The **sifraction** column shows the percentage of the system that is idle during a table scan. This is an indicator of uneven data distribution or query processing skew. For example, a value of 0.1 indicates 10% skew, a value of 0.5 indicates 50% skew, and so on. Tables that have more than 10% skew should have their distribution policies evaluated.

**Processing Skew**

Processing skew results when a disproportionate amount of data flows to, and is processed by, one or a few segments. It is often the culprit behind Greenplum Database performance and stability issues. It can happen with operations such join, sort, aggregation, and various OLAP operations. Processing skew happens in flight while a query is executing and is not as easy to detect as data skew, which is caused by uneven data distribution due to the wrong choice of distribution keys. Data skew is present at the table level, so it can be easily detected and avoided by choosing optimal distribution keys.

If single segments are failing, that is, not all segments on a host, it may be a processing skew issue. Identifying processing skew is currently a manual process. First look for spill files. If there is skew, but not enough to cause spill, it will not become a performance issue. If you determine skew exists, then find the query responsible for the skew. Following are the steps and commands to use. (Change names like the host file name passed to `gpssh` accordingly):

1. Find the OID for the database that is to be monitored for skew processing:

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

   Example output:

<table>
<thead>
<tr>
<th>oid</th>
<th>datname</th>
</tr>
</thead>
<tbody>
<tr>
<td>17088</td>
<td>gpadmin</td>
</tr>
<tr>
<td>10899</td>
<td>postgres</td>
</tr>
<tr>
<td>1</td>
<td>template1</td>
</tr>
<tr>
<td>10898</td>
<td>template0</td>
</tr>
<tr>
<td>38817</td>
<td>pws</td>
</tr>
<tr>
<td>39682</td>
<td>gpperfmon</td>
</tr>
</tbody>
</table>

   The following query shows the number of rows per segment as well as the variance from the minimum and maximum numbers of rows:
2. Run a `gpssh` command to check file sizes across all of the segment nodes in the system. Replace `<OID>` with the OID of the database from the prior command:

```
[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 \n    { for ( i in arr ) print "Segment node" i, arr[i], "bytes (" arr[i]/ (1024**3)" GB)"; \n      print "Total", tot, "bytes (" tot/(1024**3)" GB)" }' -
```

Example output:

```
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)
```

If there is a **significant and sustained** difference in disk usage, then the queries being executed should be investigated for possible skew (the example output above does not reveal significant skew). In monitoring systems, there will always be some skew, but often it is **transient** and will be **short in duration**.

3. If significant and sustained skew appears, the next task is to identify the offending query. The command in the previous step sums up the entire node. This time, find the actual segment directory. You can do this from the master or by logging into the specific node identified in the previous step. Following is an example run from the master.

This example looks specifically for sort files. Not all spill files or skew situations are caused by sort files, so you will need to customize the command:

```
$ 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
```

Here is output from this command:

```
[sdw1] 288718848
    /data1/primary/gpseg2/.../pgsql_tmp_slice0_sort_17758_0001.0
291176448
    /data2/primary/gpseg5/.../pgsql_tmp_slice0_sort_17764_0001.0
924581888
    /data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_0010.9
980582400
    /data1/primary/gpseg18/.../pgsql_tmp_slice10_sort_29425_0001.0
986447872
    /data2/primary/gpseg35/.../pgsql_tmp_slice10_sort_29602_0001.0...
[sdw5] 999620608
    /data1/primary/gpseg26/.../pgsql_tmp_slice10_sort_28637_0001.0
999751680
    /data2/primary/gpseg9/.../pgsql_tmp_slice10_sort_3969_0001.0
1000112128
    /data1/primary/gpseg13/.../pgsql_tmp_slice10_sort_24723_0001.0
1000898560
```

Scanning this output reveals that segment gpseg45 on host sdw8 is the culprit, as its sort files are larger than the others in the output.

4. Log in to the offending node with ssh and become root. Use the lsof command to find the PID for the process that owns one of the sort files:

```
[root@sdw8 ~]# lsof /data2/primary/gpseg45/base/19979/pgsql_tmp/
pgsql_tmp_slice10_sort_15673_0002.1
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_slice10_sort_15673_0002.1
```

The PID, 15673, is also part of the file name, but this may not always be the case.

5. Use the ps command with the PID to identify the database and connection information:

```
[root@sdw8 ~]# ps -eaf | grep 15673
pgadmin 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. On the master, check the pg_log log file for the user in the previous command (sbaskin), connection (con699238, and command (cmd32). The line in the log file with these three values should be the line that contains the query, but occasionally, the command number may differ slightly. For example, the ps output may show cmd32, but in the log file it is cmd34. If the query is still running, the last query for the user and connection is the offending query.

The remedy for processing skew in almost all cases is to rewrite the query. Creating temporary tables can eliminate skew. Temporary tables can be randomly distributed to force a two-stage aggregation.

**Partitioning**

A good partitioning strategy reduces the amount of data to be scanned by reading only the partitions needed to satisfy a query.
Each partition is a separate physical file or set of tiles (in the case of column-oriented tables) on every segment. Just as reading a complete row in a wide columnar table requires more time than reading the same row from a heap table, reading all partitions in a partitioned table requires more time than reading the same data from a non-partitioned table.

Following are partitioning best practices:

- Partition large tables only, do not partition small tables.
- Use partitioning on large tables only when partition elimination (partition pruning) can be achieved based on query criteria and is accomplished by partitioning the table based on the query predicate. Whenever possible, use range partitioning instead of list partitioning.
- The query planner can selectively scan partitioned tables only when the query contains a direct and simple restriction of the table using immutable operators, such as =, <, <=, >, >=, and <>. Selective scanning recognizes STABLE and IMMUTABLE functions, but does not recognize VOLATILE functions within a query. For example, WHERE clauses such as

  ```sql
date > CURRENT_DATE
  ```

  cause the query planner to selectively scan partitioned tables, but a WHERE clause such as

  ```sql
time > TIMEOFDAY
  ```

  does not. It is important to validate that queries are selectively scanning partitioned tables (partitions are being eliminated) by examining the query EXPLAIN plan.
- Do not use default partitions. The default partition is always scanned but, more importantly, in many environments they tend to overfill resulting in poor performance.
- Never partition and distribute tables on the same column.
- Do not use multi-level partitioning. While sub-partitioning is supported, it is not recommended because typically subpartitions contain little or no data. It is a myth that performance increases as the number of partitions or subpartitions increases; the administrative overhead of maintaining many partitions and subpartitions will outweigh any performance benefits. For performance, scalability and manageability, balance partition scan performance with the number of overall partitions.
- Beware of using too many partitions with column-oriented storage.
- Consider workload concurrency and the average number of partitions opened and scanned for all concurrent queries.

**Number of Partition and Columnar Storage Files**

The only hard limit for the number of files Greenplum Database supports is the operating system's open file limit. It is important, however, to consider the total number of files in the cluster, the number of files on every segment, and the total number of files on a host. In an MPP shared nothing environment, every node operates independently of other nodes. Each node is constrained by its disk, CPU, and memory. CPU and I/O constraints are not common with Greenplum Database, but memory is often a limiting factor because the query execution model optimizes query performance in memory.

The optimal number of files per segment also varies based on the number of segments on the node, the size of the cluster, SQL access, concurrency, workload, and skew. There are generally six to eight segments per host, but large clusters should have fewer segments per host. When using partitioning and columnar storage it is important to balance the total number of files in the cluster, but it is more important to consider the number of files per segment and the total number of files on a node.

**Example DCA V2 64GB Memory per Node**

- Number of nodes: 16
- Number of segments per node: 8
- Average number of files per segment: 10,000
The total number of files per node is \(8 \times 10,000 = 80,000\) and the total number of files for the cluster is \(8 \times 16 \times 10,000 = 1,280,000\). The number of files increases quickly as the number of partitions and the number of columns increase.

As a general best practice, limit the total number of files per node to under 100,000. As the previous example shows, the optimal number of files per segment and total number of files per node depends on the hardware configuration for the nodes (primarily memory), size of the cluster, SQL access, concurrency, workload and skew.

**Indexes**

Indexes are not generally needed in Greenplum Database. Most analytical queries operate on large volumes of data, while indexes are intended for locating single rows or small numbers of rows of data. In Greenplum Database, a sequential scan is an efficient method to read data as each segment contains an equal portion of the data and all segments work in parallel to read the data.

If adding an index does not produce performance gains, drop it. Verify that every index you create is used by the optimizer.

For queries with high selectivity, indexes may improve query performance. Create an index on a single column of a columnar table for drill through purposes for high cardinality columns that are required for highly selective queries.

Do not index columns that are frequently updated. Creating an index on a column that is frequently updated increases the number of writes required on updates.

Indexes on expressions should be used only if the expression is used frequently in queries.

An index with a predicate creates a partial index that can be used to select a small number of rows from large tables.

Avoid overlapping indexes. Indexes that have the same leading column are redundant.

Indexes can improve performance on compressed append-optimized tables for queries that return a targeted set of rows. For compressed data, an index access method means only the necessary pages are uncompressed.

Create selective B-tree indexes. Index selectivity is a ratio of the number of distinct values a column has divided by the number of rows in a table. For example, if a table has 1000 rows and a column has 800 distinct values, the selectivity of the index is 0.8, which is considered good.

As a general rule, drop indexes before loading data into a table. The load will run an order of magnitude faster than loading data into a table with indexes. After the load, re-create the indexes.

Bitmap indexes are suited for querying and not updating. Bitmap indexes perform best when the column has a low cardinality—100 to 100,000 distinct values. Do not use bitmap indexes for unique columns, very high, or very low cardinality data. Do not use bitmap indexes for transactional workloads.

If indexes are needed on partitioned tables, the index columns must be different than the partition columns. A benefit of indexing partitioned tables is that because the b-tree performance degrades exponentially as the size of the b-tree grows, creating indexes on partitioned tables creates smaller b-trees that perform better than with non-partitioned tables.

**Column Sequence and Byte Alignment**

For optimum performance lay out the columns of a table to achieve data type byte alignment. Lay out the columns in heap tables in the following order:

1. Distribution and partition columns
2. Fixed numeric types
3. Variable data types
Lay out the data types from largest to smallest, so that `BIGINT` and `TIMESTAMP` come before `INT` and `DATE`, and all of these types come before `TEXT`, `VARCHAR`, or `NUMERIC(x, y)`. For example, 8-byte types first (`BIGINT`, `TIMESTAMP`), 4-byte types next (`INT`, `DATE`), 2-byte types next (`SMALLINT`), and variable data type last (`VARCHAR`).

Instead of defining columns in this sequence:

```
Int, Bigint, Timestamp, Bigint, Timestamp, Int (distribution key), Date (partition key), Bigint, Smallint
```

define the columns in this sequence:

```
Int (distribution key), Date (partition key), Bigint, Bigint, Timestamp, Bigint, Timestamp, Int, Smallint
```
Memory and Resource Management

Avoid memory errors and manage Greenplum Database resources.

Memory management has a significant impact on performance in a Greenplum Database cluster. The default settings are suitable for most environments. Do not change the default settings until you understand the memory characteristics and usage on your system.

- Resolving Out of Memory Errors
- Configuring Memory for Greenplum Database
- Example Memory Configuration Calculations
- Configuring Resource Queues

Resolving Out of Memory Errors

An out of memory error message identifies the Greenplum segment, host, and process that experienced the out of memory error. For example:

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

Some common causes of out-of-memory conditions in Greenplum Database are:

- Insufficient system memory (RAM) available on the cluster
- Improperly configured memory parameters
- Data skew at the segment level
- Operational skew at the query level

Following are possible solutions to out of memory conditions:

- Tune the query to require less memory
- Reduce query concurrency using a resource queue
- Validate the `gp_vmem_protect_limit` configuration parameter at the database level. See calculations for the maximum safe setting in Configuring Memory for Greenplum Database.
- Set the memory quota on a resource queue to limit the memory used by queries executed within the resource queue
- Use a session setting to reduce the `statement_mem` used by specific queries
- Decrease `statement_mem` at the database level
- Decrease the number of segments per host in the Greenplum Database cluster. This solution requires a re-initializing Greenplum Database and reloading your data.
- Increase memory on the host, if possible. (Additional hardware may be required.)

Adding segment hosts to the cluster will not in itself alleviate out of memory problems. The memory used by each query is determined by the `statement_mem` parameter and it is set when the query is invoked. However, if adding more hosts allows decreasing the number of segments per host, then the amount of memory allocated in `gp_vmem_protect_limit` can be raised.

Configuring Memory for Greenplum Database

Most out of memory conditions can be avoided if memory is thoughtfully managed.

It is not always possible to increase system memory, but you can prevent out-of-memory conditions by configuring memory use correctly and setting up resource queues to manage expected workloads.

It is important to include memory requirements for mirror segments that become primary segments during a failure to ensure that the cluster can continue when primary segments or segment hosts fail.
The following are recommended operating system and Greenplum Database memory settings:

- **Do not configure the OS to use huge pages.**
- **vm.overcommit_memory**
  
  This is a Linux kernel parameter, set in `/etc/sysctl.conf`. It should always be set to 2. It determines the method the OS uses for determining how much memory can be allocated to processes and 2 is the only safe setting for Greenplum Database.
- **vm.overcommit_ratio**
  
  This is a Linux kernel parameter, set in `/etc/sysctl.conf`. It is the percentage of RAM that is used for application processes. The remainder is reserved for the operating system. The default on Red Hat is 50.

  Setting `vm.overcommit_ratio` too high may result in not enough memory being reserved for the operating system, which can result in segment host failure or database failure. Setting the value too low reduces the amount of concurrency and query complexity that can be run by reducing the amount of memory available to Greenplum Database. When increasing the setting it is important to remember to always reserve some memory for operating system activities.

  See [Segment Memory Configuration](#) for instructions to calculate a value for `vm.overcommit_ratio`.
- **gp_vmem_protect_limit**

  Use `gp_vmem_protect_limit` to set the maximum memory that the instance can allocate for *all* work being done in each segment database. Never set this value larger than the physical RAM on the system. If `gp_vmem_protect_limit` is too high, it is possible for memory to become exhausted on the system and normal operations may fail, causing segment failures. If `gp_vmem_protect_limit` is set to a safe lower value, true memory exhaustion on the system is prevented; queries may fail for hitting the limit, but system disruption and segment failures are avoided, which is the desired behavior.

  See [Segment Memory Configuration](#) for instructions to calculate a safe value for `gp_vmem_protect_limit`.
- **runaway_detector_activation_percent**

  Runaway Query Termination, introduced in Greenplum Database 4.3.4, prevents out of memory conditions. The `runaway_detector_activation_percent` system parameter controls the percentage of `gp_vmem_protect_limit` memory utilized that triggers termination of queries. It is set on by default at 90%. If the percentage of `gp_vmem_protect_limit` memory that is utilized for a segment exceeds the specified value, Greenplum Database terminates queries based on memory usage, beginning with the query consuming the largest amount of memory. Queries are terminated until the utilized percentage of `gp_vmem_protect_limit` is below the specified percentage.
- **statement_mem**

  Use `statement_mem` to allocate memory used for a query per segment database. If additional memory is required it will spill to disk. Set the optimal value for `statement_mem` as follows:

  \[(vmprotect * .9) / maxExpectedConcurrentQueries\]

  The default value of `statement_mem` is 125MB. For example, a query running on a Dell EMC DCA V2 system using the default `statement_mem` value will use 1GB of memory on each segment server (8 segments # 125MB). Set `statement_mem` at the session level for specific queries that require additional memory to complete. This setting works well to manage query memory on clusters with low concurrency. For clusters with high concurrency also use resource queues to provide additional control on what and how much is running on the system.
- **gp_workfile_limit_files_per_query**

  Set `gp_workfile_limit_files_per_query` to limit the maximum number of temporary spill files (workfiles) allowed per query. Spill files are created when a query requires more memory than it is allocated. When the limit is exceeded the query is terminated. The default is zero, which allows an unlimited number of spill files and may fill up the file system.
• **gp_workfile_compress_algorithm**
  
  If there are numerous spill files then set `gp_workfile_compress_algorithm` to compress the spill files. Compressing spill files may help to avoid overloading the disk subsystem with IO operations.

**Example Memory Configuration Calculations**

• Total RAM = 256GB  
• SWAP = 64GB  
• 8 primary segments and 8 mirror segments per host, in blocks of 4 hosts  
• Maximum number of primaries per host during failure is 11

**vm.overcommit_ratio calculation**

\[
gp\_vmem = \left(\frac{SWAP + RAM - (7.5GB + 0.05 \times RAM)}{1.7}\right)
\]

\[
= \left(\frac{(64 + 256) - (7.5 + 0.05 \times 256)}{1.7}\right)
\]

\[
= 176
\]

\[
vm\_overcommit\_ratio = \left(\frac{RAM - (0.026 \times gp\_vmem)}{RAM}\right)
\]

\[
= \left(\frac{256 - (0.026 \times 176)}{256}\right)
\]

\[
= .982
\]

Set `vm.overcommit_ratio` to 98.

**gp_vmem_protect_limit calculation**

\[
gp\_vmem\_protect\_limit = \frac{gp\_vmem}{maximum\_acting\_primary\_segments}
\]

\[
= \frac{176}{11}
\]

\[
= 16GB = 16384MB
\]

**Configuring Resource Queues**

Greenplum Database resource queues provide a powerful mechanism for managing the workload of the cluster. Queues can be used to limit both the numbers of active queries and the amount of memory that can be used by queries in the queue. When a query is submitted to Greenplum Database, it is added to a resource queue, which determines if the query should be accepted and when the resources are available to execute it.

• Associate all roles with an administrator-defined resource queue.

  Each login user (role) is associated with a single resource queue; any query the user submits is handled by the associated resource queue. If a queue is not explicitly assigned the user’s queries are handed by the default queue, `pg_default`.

• Do not run queries with the gpadmin role or other superuser roles.

  Superusers are exempt from resource queue limits, therefore superuser queries always run regardless of the limits set on their assigned queue.

• Use the `ACTIVE_STATEMENTS` resource queue parameter to limit the number of active queries that members of a particular queue can run concurrently.

• Use the `MEMORY_LIMIT` parameter to control the total amount of memory that queries running through the queue can utilize. By combining the `ACTIVE_STATEMENTS` and `MEMORY_LIMIT` attributes an administrator can fully control the activity emitted from a given resource queue.

  The allocation works as follows: Suppose a resource queue, `sample_queue`, has `ACTIVE_STATEMENTS` set to 10 and `MEMORY_LIMIT` set to 2000MB. This limits the queue to approximately 2 gigabytes of memory per segment. For a cluster with 8 segments per server, the total usage per server is 16 GB for `sample_queue` (2GB * 8 segments/server). If a segment server has
64GB of RAM, there could be no more than four of this type of resource queue on the system before there is a chance of running out of memory (4 queues * 16GB per queue).

Note that by using `STATEMENT_MEM`, individual queries running in the queue can allocate more than their "share" of memory, thus reducing the memory available for other queries in the queue.

- Resource queue priorities can be used to align workloads with desired outcomes. Queues with `MAX` priority throttle activity in all other queues until the `MAX` queue completes running all queries.
- Alter resource queues dynamically to match the real requirements of the queue for the workload and time of day. You can script an operational flow that changes based on the time of day and type of usage of the system and add `crontab` entries to execute the scripts.
- Use `gptoolkit` to view resource queue usage and to understand how the queues are working.
System Monitoring and Maintenance

Best practices for regular maintenance that will ensure Greenplum Database high availability and optimal performance.

Monitoring

Greenplum Database includes utilities that are useful for monitoring the system.

The gp_toolkit schema contains several views that can be accessed using SQL commands to query system catalogs, log files, and operating environment for system status information.

The gp_stats_missing view shows tables that do not have statistics and require ANALYZE to be run.

For additional information on gpstate and gpcheckperf refer to the Greenplum Database Utility Guide. For information about the gp_toolkit schema, see the Greenplum Database Reference Guide.

gpstate

The gpstate utility program displays the status of the Greenplum system, including which segments are down, master and segment configuration information (hosts, data directories, etc.), the ports used by the system, and mapping of primary segments to their corresponding mirror segments.

Run gpstate -Q to get a list of segments that are marked "down" in the master system catalog.

To get detailed status information for the Greenplum system, run gpstate -s.

gpcheckperf

The gpcheckperf utility tests baseline hardware performance for a list of hosts. The results can help identify hardware issues. It performs the following checks:

- disk I/O test – measures I/O performance by writing and reading a large file using the dd operating system command. It reports read and write rates in megabytes per second.
- memory bandwidth test – measures sustainable memory bandwidth in megabytes per second using the STREAM benchmark.
- network performance test – runs the gpnetbench network benchmark program (optionally netperf) to test network performance. The test is run in one of three modes: parallel pair test (-r N), serial pair test (-r n), or full-matrix test (-r M). The minimum, maximum, average, and median transfer rates are reported in megabytes per second.

To obtain valid numbers from gpcheckperf, the database system must be stopped. The numbers from gpcheckperf can be inaccurate even if the system is up and running with no query activity.

gpcheckperf requires a trusted host setup between the hosts involved in the performance test. It calls gpssh and gpscp, so these utilities must also be in your PATH. Specify the hosts to check individually (-h host1 -h host2 ...) or with -f hosts_file, where hosts_file is a text file containing a list of the hosts to check. If you have more than one subnet, create a separate host file for each subnet so that you can test the subnets separately.

By default, gpcheckperf runs the disk I/O test, the memory test, and a serial pair network performance test. With the disk I/O test, you must use the -d option to specify the file systems you want to test. The following command tests disk I/O and memory bandwidth on hosts listed in the subnet_1_hosts file:

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

The -r option selects the tests to run: disk I/O (d), memory bandwidth (s), network parallel pair (N), network serial pair test (n), network full-matrix test (m). Only one network mode can be selected per execution. See the Greenplum Database Reference Guide for the detailed gpcheckperf reference.
Monitoring with Operating System Utilities

The following Linux/UNIX utilities can be used to assess host performance:

- `iostat` allows you to monitor disk activity on segment hosts.
- `top` displays a dynamic view of operating system processes.
- `vmstat` displays memory usage statistics.

You can use `gpssh` to run utilities on multiple hosts.

Best Practices

- Implement the “Recommended Monitoring and Maintenance Tasks” in the *Greenplum Database Administrator Guide*.
- Run `gpcheckperf` at install time and periodically thereafter, saving the output to compare system performance over time.
- Use all the tools at your disposal to understand how your system behaves under different loads.
- Examine any unusual event to determine the cause.
- Monitor query activity on the system by running explain plans periodically to ensure the queries are running optimally.
- Review plans to determine whether index are being used and partition elimination is occurring as expected.

Additional Information

- `gpcheckperf` reference in the *Greenplum Database Utility Guide*.
- "Recommended Monitoring and Maintenance Tasks" in the *Greenplum Database Administrator Guide*.
- [www.netperf.org](http://www.netperf.org) to use `netperf`, `netperf` must be installed on each host you test. See `gpcheckperf` reference for more information.

Updating Statistics with **ANALYZE**

The most important prerequisite for good query performance is to begin with accurate statistics for the tables. Updating statistics with the **ANALYZE** statement enables the query planner to generate optimal query plans. When a table is analyzed, information about the data is stored in the system catalog tables. If the stored information is out of date, the planner can generate inefficient plans.

Generating Statistics Selectively

Running **ANALYZE** with no arguments updates statistics for all tables in the database. This can be a very long-running process and it is not recommended. You should **ANALYZE** tables selectively when data has changed or use the `analyzedb` utility.

Running **ANALYZE** on a large table can take a long time. If it is not feasible to run **ANALYZE** on all columns of a very large table, you can generate statistics for selected columns only using **ANALYZE** `table(column, ...)`. Be sure to include columns used in joins, WHERE clauses, SORT clauses, GROUP BY clauses, or HAVING clauses.

For a partitioned table, you can run **ANALYZE** on just partitions that have changed, for example, if you add a new partition. Note that for partitioned tables, you can run **ANALYZE** on the parent (main) table, or on the leaf nodes—the partition files where data and statistics are actually stored. The intermediate files for sub-partitioned tables store no data or statistics, so running **ANALYZE** on them does not work. You can find the names of the partition tables in the `pg_partitions` system catalog:

```sql
SELECT partitiontablename from pg_partitions WHERE tablename='parent_table';
```
**Improving Statistics Quality**

There is a trade-off between the amount of time it takes to generate statistics and the quality, or accuracy, of the statistics.

To allow large tables to be analyzed in a reasonable amount of time, `ANALYZE` takes a random sample of the table contents, rather than examining every row. To increase sampling for all table columns adjust the `default_statistics_target` configuration parameter. The target value ranges from 1 to 1000; the default target value is 25. The `default_statistics_target` variable applies to all columns by default. A larger target value increases the time needed to perform the `ANALYZE`, but may improve the quality of the query planner’s estimates. This is especially true for columns with irregular data patterns. `default_statistics_target` can be set at the master/session level and requires a reload.

The `gp_analyze_relative_error` configuration parameter affects the sampling rate during statistics collection to determine cardinality in a column. For example, a value of .5 is equivalent to an acceptable error of 50%. The default is .25. Use the `gp_analyze_relative_error` parameter to set the acceptable estimated relative error in the cardinality of a table. If statistics do not produce good estimates of cardinality for a particular table attribute, decreasing the relative error fraction (accepting less errors) tells the system to sample more rows. However, it is not recommended to reduce this below 0.1 as it will increase `ANALYZE` time substantially.

**When to Run ANALYZE**

Run `ANALYZE`:
- after loading data,
- after `CREATE INDEX` operations,
- and after `INSERT`, `UPDATE`, and `DELETE` operations that significantly change the underlying data.

`ANALYZE` requires only a read lock on the table, so it may be run in parallel with other database activity, but do not run `ANALYZE` while performing loads, `INSERT`, `UPDATE`, `DELETE`, and `CREATE INDEX` operations.

**Configuring Automatic Statistics Collection**

The `gp_autostats_mode` configuration parameter, together with the `gp_autostats_on_change_threshold` parameter, determines when an automatic analyze operation is triggered. When automatic statistics collection is triggered, the planner adds an `ANALYZE` step to the query.

By default, `gp_autostats_mode` is `on_no_stats`, which triggers statistics collection for `CREATE TABLE AS SELECT`, `INSERT`, or `COPY` operations on any table that has no existing statistics.

Setting `gp_autostats_mode` to `on_change` triggers statistics collection only when the number of rows affected exceeds the threshold defined by `gp_autostats_on_change_threshold`, which has a default value of 2147483647. Operations that can trigger automatic statistics collection with `on_change` are: `CREATE TABLE AS SELECT`, `UPDATE`, `DELETE`, `INSERT`, and `COPY`.

Setting `gp_autostats_mode` to `none` disables automatic statistics collection.

For partitioned tables, automatic statistics collection is not triggered if data is inserted from the top-level parent table of a partitioned table. But automatic statistics collection is triggered if data is inserted directly in a leaf table (where the data is stored) of the partitioned table.

**Managing Bloat in the Database**

Greenplum Database heap tables use the PostgreSQL Multiversion Concurrency Control (MVCC) storage implementation. A deleted or updated row is logically deleted from the database, but a non-visible image of the row remains in the table. These deleted rows, also called expired rows, are tracked in a free space map. Running `VACUUM` marks the expired rows as free space that is available for reuse by subsequent inserts.
If the free space map is not large enough to accommodate all of the expired rows, the VACUUM command is unable to reclaim space for expired rows that overflowed the free space map. The disk space may only be recovered by running VACUUM FULL, which locks the table, copies rows one-by-one to the beginning of the file, and truncates the file. This is an expensive operation that can take an exceptional amount of time to complete with a large table. It should be used only on smaller tables. If you attempt to kill a VACUUM FULL operation, the system can be disrupted.

**Important:**

It is very important to run VACUUM after large UPDATE and DELETE operations to avoid the necessity of ever running VACUUM FULL.

If the free space map overflows and it is necessary to recover the space it is recommended to use the CREATE TABLE...AS SELECT command to copy the table to a new table, which will create a new compact table. Then drop the original table and rename the copied table.

It is normal for tables that have frequent updates to have a small or moderate amount of expired rows and free space that will be reused as new data is added. But when the table is allowed to grow so large that active data occupies just a small fraction of the space, the table has become significantly "b hollowed." Bloated tables require more disk storage and additional I/O that can slow down query execution.

Bloat affects heap tables, system catalogs, and indexes.

Running the VACUUM statement on tables regularly prevents them from growing too large. If the table does become significantly bloated, the VACUUM FULL statement (or an alternative procedure) must be used to compact the file. If a large table becomes significantly bloated, it is better to use one of the alternative methods described in Removing Bloat from Database Tables to remove the bloat.

**Caution:** Never run VACUUM FULL <database_name> and do not run VACUUM FULL on large tables in a Greenplum Database.

**Sizing the Free Space Map**

Expired rows in heap tables are added to a shared free space map when you run VACUUM. The free space map must be adequately sized to accommodate these rows. If the free space map is not large enough, any space occupied by rows that overflow the free space map cannot be reclaimed by a regular VACUUM statement. You will have to use VACUUM FULL or an alternative method to recover the space.

You can avoid overflowing the free space map by running the VACUUM statement regularly. The more bloated a table becomes, the more rows that must be tracked in the free space map. For very large databases with many objects, you may need to increase the size of the free space map to prevent overflow.

The max_fsm_pages configuration parameter sets the maximum number of disk pages for which free space will be tracked in the shared free-space map. Each page slot consumes six bytes of shared memory. The default value for max_fsm_pages is 200,000.

The max_fsm_relations configuration parameter sets the maximum number of relations for which free space will be tracked in the shared memory free-space map. It should be set to a value larger than the total number of tables, indexes, and system tables in the database. It costs about 60 bytes of memory for each relation per segment instance. The default value is 1000.

See the Greenplum Database Reference Guide for detailed information about these configuration parameters.

**Detecting Bloat**

The statistics collected by the ANALYZE statement can be used to calculate the expected number of disk pages required to store a table. The difference between the expected number of pages and the actual number of pages is a measure of bloat. The gp_toolkit schema provides a gp_bloat_diag view that
identifies table bloat by comparing the ratio of expected to actual pages. To use it, make sure statistics are up to date for all of the tables in the database, then run the following SQL:

```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)
```

The results include only tables with moderate or significant bloat. Moderate bloat is reported when the ratio of actual to expected pages is greater than four and less than ten. Significant bloat is reported when the ratio is greater than ten.

The `gp_toolkit.gp_bloat_expected_pages` view lists the actual number of used pages and expected number of used pages for each database object.

```sql
gpadmin=# SELECT * FROM gp_toolkit.gp_bloat_expected_pages LIMIT 5;
btdrelid | btdrelpages | btdexppages
----------+-------------+-------------
10789 | 1 | 1
10794 | 1 | 1
10799 | 1 | 1
5004 | 1 | 1
7175 | 1 | 1
(5 rows)
```

The `btdrelid` is the object ID of the table. The `btdrelpages` column reports the number of pages the table uses; the `btdexppages` column is the number of pages expected. Again, the numbers reported are based on the table statistics, so be sure to run `ANALYZE` on tables that have changed.

### Removing Bloat from Database Tables

The `VACUUM` command adds expired rows to the shared free space map so that the space can be reused. When `VACUUM` is run regularly on a table that is frequently updated, the space occupied by the expired rows can be promptly reused, preventing the table file from growing larger. It is also important to run `VACUUM` before the free space map is filled. For heavily updated tables, you may need to run `VACUUM` at least once a day to prevent the table from becoming bloated.

**Warning:** When a table is significantly bloated, it is better to run `ANALYZE` before running `VACUUM`. Because `ANALYZE` uses block-level sampling, a table with a high ratio of blocks containing no valid rows can cause `ANALYZE` to set the `reltuples` column of the `pg_class` system catalog to an inaccurate value or 0, which can lead to poorly optimized queries. The `VACUUM` command produces a more accurate count and when run after `ANALYZE` will correct an inaccurate row count estimate.

When a table accumulates significant bloat, running the `VACUUM` command is insufficient. For small tables, running `VACUUM FULL <table_name>` can reclaim space used by rows that overflowed the free space map and reduce the size of the table file. However, a `VACUUM FULL` statement is an expensive operation that requires an `ACCESS EXCLUSIVE` lock and may take an exceptionally long and unpredictable amount of time to finish. Rather than run `VACUUM FULL` on a large table, an alternative method is required to remove bloat from a large file. Note that every method for removing bloat from large tables is resource intensive and should be done only under extreme circumstances.

The first method to remove bloat from a large table is to create a copy of the table excluding the expired rows, drop the original table, and rename the copy. This method uses the `CREATE TABLE <table_name> AS SELECT` statement to create the new table, for example:

```sql
gpadmin=# CREATE TABLE mytable_tmp AS SELECT * FROM mytable;
gpadmin=# DROP TABLE mytable;
```
A second way to remove bloat from a table is to redistribute the table, which rebuilds the table without the expired rows. Follow these steps:

1. Make a note of the table's distribution columns.
2. Change the table's distribution policy to random:

   ```sql
   ALTER TABLE mytable SET WITH (REORGANIZE=false)
   DISTRIBUTED randomly;
   ```

   This changes the distribution policy for the table, but does not move any data. The command should complete instantly.
3. Change the distribution policy back to its initial setting:

   ```sql
   ALTER TABLE mytable SET WITH (REORGANIZE=true)
   DISTRIBUTED BY (<original distribution columns>);
   ```

   This step redistributes the data. Since the table was previously distributed with the same distribution key, the rows are simply rewritten on the same segment, excluding expired rows.

### Removing Bloat from Indexes

The `VACUUM` command only recovers space from tables. To recover the space from indexes, recreate them using the `REINDEX` command.

To rebuild all indexes on a table run `REINDEX table_name;`. To rebuild a particular index, run `REINDEX index_name;`. `REINDEX` does not update the `reltuples` and `relpages` statistics for the index, so it is important to `ANALYZE` the table to update these statistics after reindexing.

### Removing Bloat from System Catalogs

Greenplum Database system catalogs are also heap tables and can become bloated over time. As database objects are created, altered, or dropped, expired rows are left in the system catalogs. Using `gpload` to load data contributes to the bloat since `gpload` creates and drops external tables. (Rather than use `gpload`, it is recommended to use `gpfdist` to load data.)

Bloat in the system catalogs increases the time require to scan the tables, for example, when creating explain plans. System catalogs are scanned frequently and if they become bloated, overall system performance is degraded.

It is recommended to run `VACUUM` on the system catalog nightly and at least weekly. At the same time, running `REINDEX SYSTEM` removes bloat from the indexes. Alternatively, you can reindex system tables using the `reindexdb` utility with the `-s` (`--system`) option. After reindexing, it is also important to run `ANALYZE`, because the `REINDEX` command rebuilds indexes with empty statistics.

The following script runs `VACUUM`, `REINDEX`, and `ANALYZE` on the system catalogs.

```bash
#!/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'"
psql -tc "SELECT 'VACUUM' || $SYSTABLES" $DBNAME
reindexdb -s -d $DBNAME
analyzedb -s -d $DBNAME
```

If the system catalogs become significantly bloated, you must perform an intensive system catalog maintenance procedure. The `CREATE TABLE AS SELECT` and redistribution key methods for removing bloat cannot be used with system catalogs. You must instead run `VACUUM FULL` during a scheduled downtime period. During this period, stop all catalog activity on the system; `VACUUM FULL` takes exclusive
locks against the system catalog. Running VACUUM regularly can prevent the need for this more costly procedure.

Removing Bloat from Append-Optimized Tables

Append-optimized tables are handled much differently than heap tables. Although append-optimized tables allow updates, inserts, and deletes, they are not optimized for these operations and it is recommended to not use them with append-optimized tables. If you heed this advice and use append-optimized for load-once/read-many workloads, VACUUM on an append-optimized table runs almost instantaneously.

If you do run UPDATE or DELETE commands on an append-optimized table, expired rows are tracked in an auxiliary bitmap instead of the free space map. VACUUM is the only way to recover the space. Running VACUUM on an append-optimized table with expired rows compacts a table by rewriting the entire table without the expired rows. However, no action is performed if the percentage of expired rows in the table exceeds the value of the gp_appendonly_compaction_threshold configuration parameter, which is 10 (10%) by default. The threshold is checked on each segment, so it is possible that a VACUUM statement will compact an append-only table on some segments and not others. Compacting append-only tables can be disabled by setting the gp_appendonly_compaction parameter to no.

Monitoring Greenplum Database Log Files

Know the location and content of system log files and monitor them on a regular basis and not just when problems arise.

The following table shows the locations of the various Greenplum Database log files. In file paths, date is a date in the format YYYYMMDD, instance is the current instance name, and n is the segment number.

<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/var/gpadmin/gpadminlogs/*</td>
<td>Many different types of log files, directory on each server</td>
</tr>
<tr>
<td>/var/gpadmin/gpadminlogs/gpstart_date.log</td>
<td>start log</td>
</tr>
<tr>
<td>/var/gpadmin/gpadminlogs/gpstop_date.log</td>
<td>stop log</td>
</tr>
<tr>
<td>/var/gpadmin/gpadminlogs/gpsegstart.py_idb*gpadmin_date.log</td>
<td>segment start log</td>
</tr>
<tr>
<td>/var/gpadmin/gpadminlogs/gpsegstop.py_idb*gpadmin_date.log</td>
<td>segment stop log</td>
</tr>
<tr>
<td>/var/gpdb/instance/datamaster/gpseg-1/pg_log/startup.log</td>
<td>instance start log</td>
</tr>
<tr>
<td>/var/gpdb/instance/datamaster/gpseg-1/gpperfmon/logs/gpmon.*.log</td>
<td>gpperfmon logs</td>
</tr>
<tr>
<td>/var/gpdb/instance/datamirror/gpseg-1/pg_log/*.csv</td>
<td>mirror segment logs</td>
</tr>
<tr>
<td>/var/gpdb/instance/dataprimary/gpseg-1/pg_log/*.csv</td>
<td>primary segment logs</td>
</tr>
<tr>
<td>/var/log/messages</td>
<td>Global Linux system messages</td>
</tr>
</tbody>
</table>

Use gplogfilter -t (--trouble) first to search the master log for messages beginning with ERROR:, FATAL:, or PANIC:. Messages beginning with WARNING may also provide useful information.
To search log files on the segment hosts, use the Greenplum `gplogfilter` utility with `gpssh` to connect to segment hosts from the master host. You can identify corresponding log entries in segment logs by the `statement_id`.

The `log_rotation_age` configuration parameter specifies when a new log file is automatically created while a database instance is running. By default, a new log file is created every day.
Loading Data
Description of the different ways to add data to Greenplum Database.

**INSERT Statement with Column Values**
A singleton `INSERT` statement with values adds a single row to a table. The row flows through the master and is distributed to a segment. This is the slowest method and is not suitable for loading large amounts of data.

**COPY Statement**
The PostgreSQL `COPY` statement copies data from an external file into a database table. It can insert multiple rows more efficiently than an `INSERT` statement, but the rows are still passed through the master. All of the data is copied in one command; it is not a parallel process.

Data input to the `COPY` command is from a file or the standard input. For example:

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

Use `COPY` to add relatively small sets of data, for example dimension tables with up to ten thousand rows, or one-time data loads.

Use `COPY` when scripting a process that loads small amounts of data, less than 10 thousand rows.

Since `COPY` is a single command, there is no need to disable autocommit when you use this method to populate a table.

You can run multiple concurrent `COPY` commands to improve performance.

**External Tables**
External tables provide access to data in sources outside of Greenplum Database. They can be accessed with `SELECT` statements and are commonly used with the Extract, Load, Transform (ELT) pattern, a variant of the Extract, Transform, Load (ETL) pattern that takes advantage of Greenplum Database's fast parallel data loading capability.

With ETL, data is extracted from its source, transformed outside of the database using external transformation tools, such as Informatica or Datastage, and then loaded into the database.

With ELT, Greenplum external tables provide access to data in external sources, which could be read-only files (for example, text, CSV, or XML files), Web servers, Hadoop file systems, executable OS programs, or the Greenplum `gpfdist` file server, described in the next section. External tables support SQL operations such as select, sort, and join so the data can be loaded and transformed simultaneously, or loaded into a `load table` and transformed in the database into target tables.

The external table is defined with a `CREATE EXTERNAL TABLE` statement, which has a `LOCATION` clause to define the location of the data and a `FORMAT` clause to define the formatting of the source data so that the system can parse the input data. Files use the `file://` protocol, and must reside on a segment host in a location accessible by the Greenplum superuser. The data can be spread out among the segment hosts with no more than one file per primary segment on each host. The number of files listed in the `LOCATION` clause is the number of segments that will read the external table in parallel.

**External Tables with Gpfdist**
The fastest way to load large fact tables is to use external tables with `gpdist`. `gpfdist` is a file server program using an HTTP protocol that serves external data files to Greenplum Database segments.
in parallel. A gpfdist instance can serve 200 MB/second and many gpfdist processes can run simultaneously, each serving up a portion of the data to be loaded. When you begin the load using a statement such as `INSERT INTO <table> SELECT * FROM <external_table>`, the INSERT statement is parsed by the master and distributed to the primary segments. The segments connect to the gpfdist servers and retrieve the data in parallel, parse and validate the data, calculate a hash from the distribution key data and, based on the hash key, send the row to its destination segment. By default, each gpfdist instance will accept up to 64 connections from segments. With many segments and gpfdist servers participating in the load, data can be loaded at very high rates.

Primary segments access external files in parallel when using gpfdist up to the value of `gp_external_max_segments`. When optimizing gpfdist performance, maximize the parallelism as the number of segments increase. Spread the data evenly across as many ETL nodes as possible. Split very large data files into equal parts and spread the data across as many file systems as possible.

Run two gpfdist instances per file system. gpfdist tends to be CPU bound on the segment nodes when loading. But if, for example, there are eight racks of segment nodes, there is lot of available CPU on the segments to drive more gpfdist processes. Run gpfdist on as many interfaces as possible. Be aware of bonded NICs and be sure to start enough gpfdist instances to work them.

It is important to keep the work even across all these resources. The load is as fast as the slowest node. Skew in the load file layout will cause the overall load to bottleneck on that resource.

The `gp_external_max_segs` configuration parameter controls the number of segments each gpfdist process serves. The default is 64. You can set a different value in the `postgresql.conf` configuration file on the master. Always keep `gp_external_max_segs` and the number of gpfdist processes an even factor; that is, the `gp_external_max_segs` value should be a multiple of the number of gpfdist processes. For example, if there are 12 segments and 4 gpfdist processes, the planner round robin the segment connections as follows:

<table>
<thead>
<tr>
<th>Segment 1</th>
<th>gpfdist 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment 2</td>
<td>gpfdist 2</td>
</tr>
<tr>
<td>Segment 3</td>
<td>gpfdist 3</td>
</tr>
<tr>
<td>Segment 4</td>
<td>gpfdist 4</td>
</tr>
<tr>
<td>Segment 5</td>
<td>gpfdist 1</td>
</tr>
<tr>
<td>Segment 6</td>
<td>gpfdist 2</td>
</tr>
<tr>
<td>Segment 7</td>
<td>gpfdist 3</td>
</tr>
<tr>
<td>Segment 8</td>
<td>gpfdist 4</td>
</tr>
<tr>
<td>Segment 9</td>
<td>gpfdist 1</td>
</tr>
<tr>
<td>Segment 10</td>
<td>gpfdist 2</td>
</tr>
<tr>
<td>Segment 11</td>
<td>gpfdist 3</td>
</tr>
<tr>
<td>Segment 12</td>
<td>gpfdist 4</td>
</tr>
</tbody>
</table>

Drop indexes before loading into existing tables and re-create the index after loading. Creating an index on pre-existing data is faster than updating it incrementally as each row is loaded.

Run `ANALYZE` on the table after loading. Disable automatic statistics collection during loading by setting `gp_autostats_mode` to `NONE`. Run `VACUUM` after load errors to recover space.

Performing small, high frequency data loads into heavily partitioned column-oriented tables can have a high impact on the system because of the number of physical files accessed per time interval.

**Gpload**

`gpload` is a data loading utility that acts as an interface to the Greenplum external table parallel loading feature.

Beware of using `gpload` as it can cause catalog bloat by creating and dropping external tables. Use `gpfdist` instead, since it provides the best performance.

`gpload` executes a load using a specification defined in a YAML-formatted control file. It performs the following operations:
- Invokes gpfdist processes
- Creates a temporary external table definition based on the source data defined
- Executes an INSERT, UPDATE, or MERGE operation to load the source data into the target table in the database
- Drops the temporary external table
- Cleans up gpfdist processes

The load is accomplished in a single transaction.

**Best Practices**

- Drop any indexes on an existing table before loading data and recreate the indexes after loading. Newly creating an index is faster than updating an index incrementally as each row is loaded.
- Disable automatic statistics collection during loading by setting the `gp_autostats_mode` configuration parameter to NONE.
- External tables are not intended for frequent or ad hoc access.
- External tables have no statistics to inform the optimizer. You can set rough estimates for the number of rows and disk pages for the external table in the `pg_class` system catalog with a statement like the following:

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

- When using gpfdist, maximize network bandwidth by running one gpfdist instance for each NIC on the ETL server. Divide the source data evenly between the gpfdist instances.
- When using gpload, run as many simultaneous gpload instances as resources allow. Take advantage of the CPU, memory, and networking resources available to increase the amount of data that can be transferred from ETL servers to the Greenplum Database.
- Use the `SEGMENT REJECT LIMIT` clause of the COPY statement to set a limit for the number or percentage of rows that can have errors before the COPY FROM command is aborted. The reject limit is per segment; when any one segment exceeds the limit, the command is aborted and no rows are added. Use the LOG ERRORS clause to save error rows. If a row has errors in the formatting—for example missing or extra values, or incorrect data types—Greenplum Database stores the error information and row internally. Use the `gp_read_error_log()` built-in SQL function to access this stored information.
- If the load has errors, run VACUUM on the table to recover space.
- After you load data into a table, run VACUUM on heap tables, including system catalogs, and ANALYZE on all tables. It is not necessary to run VACUUM on append-optimized tables. If the table is partitioned, you can vacuum and analyze just the partitions affected by the data load. These steps clean up any rows from aborted loads, deletes, or updates and update statistics for the table.
- Recheck for segment skew in the table after loading a large amount of data. You can use a query like the following to check for skew:

  ```sql
  SELECT ...;
  ```

- By default, gpfdist assumes a maximum record size of 32K. To load data records larger than 32K, you must increase the maximum row size parameter by specifying the `-m <bytes>` option on the gpfdist command line. If you use gpload, set the `MAX_LINE_LENGTH` parameter in the gpload control file.

  **Note:** Integrations with Informatica Power Exchange are currently limited to the default 32K record length.

**Additional Information**

See the *Greenplum Database Reference Guide* for detailed instructions for loading data using gpfdist and gpload.
Migrating Data with Gptransfer

Information about the gptransfer migration utility, which transfers Greenplum Database metadata and data from one Greenplum database to another.

gptransfer enables you to migrate the entire contents of a database, or just selected tables, to another database. The source and destination databases may be in the same or a different cluster. gptransfer moves data in parallel across all the segments, using the gpfdist data loading utility to attain the highest transfer rates.

gptransfer handles the setup and execution of the data transfer. Participating clusters must already exist, have network access between all hosts in both clusters, and have certificate-authenticated ssh access between all hosts in both clusters.

The gptransfer interface includes options to transfer one or more full databases, or one or more database tables. A full database transfer includes the database schema, table data, indexes, views, roles, user-defined functions, and resource queues. Configuration files, including postgres.conf and pg_hba.conf, must be transferred manually by an administrator. Extensions installed in the database with gppkg, such as MADlib and programming language extensions, must be installed in the destination database by an administrator.

What gptransfer Does

gptransfer uses writable and readable external tables, the Greenplum gpfdist parallel data-loading utility, and named pipes to transfer data from the source database to the destination database. Segments on the source cluster select from the source database table and insert into a writable external table. Segments in the destination cluster select from a readable external table and insert into the destination database table. The writable and readable external tables are backed by named pipes on the source cluster's segment hosts, and each named pipe has a gpfdist process serving the pipe's output to the readable external table on the destination segments.

gptransfer orchestrates the process by processing the database objects to be transferred in batches. For each table to be transferred, it performs the following tasks:

• creates a writable external table in the source database
• creates a readable external table in the destination database
• creates named pipes and gpfdist processes on segment hosts in the source cluster
• executes a SELECT INTO statement in the source database to insert the source data into the writable external table
• executes a SELECT INTO statement in the destination database to insert the data from the readable external table into the destination table
• optionally validates the data by comparing row counts or MD5 hashes of the rows in the source and destination
• cleans up the external tables, named pipes, and gpfdist processes

Prerequisites

• The gptransfer utility can only be used with Greenplum Database, including the Dell EMC DCA appliance. Pivotal HAWQ is not supported as a source or destination.
• The source and destination Greenplum clusters must both be version 4.2 or higher.
• At least one Greenplum instance must include the gptransfer utility in its distribution. The utility is included with Greenplum Database version 4.2.8.1 and higher and 4.3.2.0 and higher. If neither the source or destination includes gptransfer, you must upgrade one of the clusters to use gptransfer.
• The gptransfer utility can be run from the cluster with the source or destination database.
• The number of segments in the destination cluster must be greater than or equal to the number of hosts in the source cluster. The number of segments in the destination may be smaller than the number of segments in the source, but the data will transfer at a slower rate.
• The segment hosts in both clusters must have network connectivity with each other.
• Every host in both clusters must be able to connect to every other host with certificate-authenticated SSH. You can use the `gpssh_exkeys` utility to exchange public keys between the hosts of both clusters.

**Fast Mode and Slow Mode**

`gptransfer` sets up data transfer using the `gpfdist` parallel file serving utility, which serves the data evenly to the destination segments. Running more `gpfdist` processes increases the parallelism and the data transfer rate. When the destination cluster has the same or a greater number of segments than the source cluster, `gptransfer` sets up one named pipe and one `gpfdist` process for each source segment. This is the configuration for optimal data transfer rates and is called *fast mode*.

The configuration of the input end of the named pipes differs when there are fewer segments in the destination cluster than in the source cluster. `gptransfer` handles this alternative setup automatically. The difference in configuration means that transferring data into a destination cluster with fewer segments than the source cluster is not as fast as transferring into a destination cluster of the same or greater size. It is called *slow mode* because there are fewer `gpfdist` processes serving the data to the destination cluster, although the transfer is still quite fast with one `gpfdist` per segment host.

When the destination cluster is smaller than the source cluster, there is one named pipe per segment host and all segments on the host send their data through it. The segments on the source host write their data to a writable external web table connected to a `gpfdist` process on the input end of the named pipe. This consolidates the table data into a single named pipe. A `gpfdist` process on the output of the named pipe serves the consolidated data to the destination cluster.

On the destination side, `gptransfer` defines a readable external table with the `gpfdist` server on the source host as input and selects from the readable external table into the destination table. The data is distributed evenly to all the segments in the destination cluster.

**Batch Size and Sub-batch Size**

The degree of parallelism of a `gptransfer` execution is determined by two command-line options: `--batch-size` and `--sub-batch-size`. The `--batch-size` option specifies the number of tables to transfer in a batch. The default batch size is 2, which means that two table transfers are in process at any time. The minimum batch size is 1 and the maximum is 10. The `--sub-batch-size` parameter specifies the maximum number of parallel sub-processes to start to do the work of transferring a table. The default is 25 and the maximum is 50. The product of the batch size and sub-batch size is the amount of parallelism. If set to the defaults, for example, `gptransfer` can perform 50 concurrent tasks. Each thread is a Python process and consumes memory, so setting these values too high can cause a Python Out of Memory error. For this reason, the batch sizes should be tuned for your environment.

**Preparing Hosts for gptransfer**

When you install a Greenplum Database cluster, you set up all the master and segment hosts so that the Greenplum Database administrative user (`gpadmin`) can connect with ssh from every host in the cluster to any other host in the cluster without providing a password. The `gptransf` utility requires this capability between every host in the source and destination clusters. First, ensure that the clusters have network connectivity with each other. Then, prepare a hosts file containing a list of all the hosts in both clusters, and use the `gpssh-exkeys` utility to exchange keys. See the reference for `gpssh-exkeys` in the *Greenplum Database Utility Guide*.

The host map file is a text file that lists the segment hosts in the source cluster. It is used to enable communication between the hosts in Greenplum clusters. The file is specified on the `gptransfer` command line with the `--source-map-file=host_map_file` command option. It is a required option when using `gptransfer` to copy data between two separate Greenplum clusters.
The file contains a list in the following format:

```
host1_name,host1_ip_addr
host2_name,host2_ipaddr
...
```

The file uses IP addresses instead of host names to avoid any problems with name resolution between the clusters.

**Limitations**

gptransfer transfers data from user databases only; the postgres, template0, and template1 databases cannot be transferred. Administrators must transfer configuration files manually and install extensions into the destination database with gppkg.

The destination cluster must have at least as many segments as the source cluster has segment hosts. Transferring data to a smaller cluster is not as fast as transferring data to a larger cluster.

Transferring small or empty tables can be unexpectedly slow. There is significant fixed overhead in setting up external tables and communications processes for parallel data loading between segments that occurs whether or not there is actual data to transfer.

**Full Mode and Table Mode**

When run with the `--full` option, gptransfer copies all tables, views, indexes, roles, user-defined functions, and resource queues in the source database to the destination database. Databases to be transferred must not already exist on the destination cluster. If gptransfer finds the database on the destination it fails with a message like the following:

```
[ERROR]:- gptransfer: error: --full option specified but tables exist on destination system
```

To copy tables individually, specify the tables using either the `-t` command-line option (one option per table) or by using the `-f` command-line option to specify a file containing a list of tables to transfer. Tables are specified in the fully-qualified format `database.schema.table`. The table definition, indexes, and table data are copied. The database must already exist on the destination cluster.

By default, gptransfer fails if you attempt to transfer a table that already exists in the destination database:

```
[INFO]:-Validating transfer table set...
[CRITICAL]:- gptransfer failed. (Reason='Table database.schema.table exists in database database .') exiting...
```

Override this behavior with the `--skip-existing`, `--truncate`, or `--drop` options.

The following table shows the objects that are copied in full mode and table mode.

<table>
<thead>
<tr>
<th>Object</th>
<th>Full Mode</th>
<th>Table Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Indexes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Roles</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Functions</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Resource Queues</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>postgres.conf</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
The \texttt{--full} option and the \texttt{--schema-only} option can be used together if you want to copy a database in phases, for example, during scheduled periods of downtime or low activity. Run \texttt{gptransfer --full --schema-only -d <database_name> ...} to create the full database on the destination cluster, but with no data. Then you can transfer the tables in stages during scheduled down times or periods of low activity. Be sure to include the \texttt{--truncate} or \texttt{--drop} option when you later transfer tables to prevent the transfer from failing because the table already exists at the destination.

### Locking

The \texttt{-x} option enables table locking. An exclusive lock is placed on the source table until the copy and validation, if requested, are complete.

### Validation

By default, \texttt{gptransfer} does not validate the data transferred. You can request validation using the \texttt{--validate=type} option. The validation \texttt{type} can be one of the following:

- \texttt{count} – Compares the row counts for the tables in the source and destination databases.
- \texttt{md5} – Sorts tables on both source and destination, and then performs a row-by-row comparison of the MD5 hashes of the sorted rows.

If the database is accessible during the transfer, be sure to add the \texttt{-x} option to lock the table. Otherwise, the table could be modified during the transfer, causing validation to fail.

### Failed Transfers

A failure on a table does not end the \texttt{gptransfer} job. When a transfer fails, \texttt{gptransfer} displays an error message and adds the table name to a failed transfers file. At the end of the \texttt{gptransfer} session, \texttt{gptransfer} writes a message telling you there were failures, and providing the name of the failed transfer file. For example:

```
[WARNING]:-Some tables failed to transfer. A list of these tables
[WARNING]:-has been written to the file
   failed_transfer_tables_20140808_101813.txt
[WARNING]:-This file can be used with the \texttt{-f} option to continue
```

The failed transfers file is in the format required by the \texttt{-f} option, so you can use it to start a new \texttt{gptransfer} session to retry the failed transfers.

### Best Practices

\texttt{gptransfer} creates a configuration that allows transferring large amounts of data at very high rates. For small or empty tables, however, the \texttt{gptransfer} set up and clean up are too expensive. The best practice is to use \texttt{gptransfer} for large tables and to use other methods for copying smaller tables.

1. Before you begin to transfer data, replicate the schema or schemas from the source cluster to the destination cluster. Do not use \texttt{gptransfer} with the \texttt{--full --schema-only} options. Here are some options for copying the schema:

   - Use the \texttt{gpsd} (Greenplum Statistics Dump) support utility. This method includes statistics, so be sure to run \texttt{ANALYZE} after creating the schema on the destination cluster.
   - Use the PostgreSQL \texttt{pg_dump} or \texttt{pg_dumpall} utility with the \texttt{--schema-only} option.
   - DDL scripts, or any other method for recreating schema in the destination database.
2. Divide the non-empty tables to be transferred into large and small categories, using criteria of your own choice. For example, you could decide large tables have more than one million rows or a raw data size greater than 1GB.

3. Transfer data for small tables using the SQL `COPY` command. This eliminates the warm-up/cool-down time each table incurs when using the `gptransfer` utility.
   - Optionally, write or reuse existing shell scripts to loop through a list of table names to copy with the `COPY` command.

4. Use `gptransfer` to transfer the large table data in batches of tables.
   - It is best to transfer to the same size cluster or to a larger cluster so that `gptransfer` runs in fast mode.
   - If any indexes exist, drop them before starting the transfer process.
   - Use the `gptransfer` table (`-t`) or file (`-f`) options to execute the migration in batches of tables. Do not run `gptransfer` using the full mode; the schema and smaller tables have already been transferred.
   - Perform test runs of the `gptransfer` process before the production migration. This ensures tables can be transferred successfully. You can experiment with the `--batch-size` and `--sub-batch-size` options to obtain maximum parallelism. Determine proper batching of tables for iterative `gptransfer` runs.
   - Include the `--skip-existing` option because the schema already exists on the destination cluster.
   - Use only fully qualified table names. Be aware that periods (.), whitespace, quotes (') and double quotes (") in table names may cause problems.
   - If you decide to use the `--validation` option to validate the data after transfer, be sure to also use the `--x` option to place an exclusive lock on the source table.

5. After all tables are transferred, perform the following tasks:
   - Check for and correct any failed transfers.
   - Recreate the indexes that were dropped before the transfer.
   - Ensure any roles, functions, and resource queues are created in the destination database. These objects are not transferred when you use the `gptransfer -t` option.
   - Copy the `postgres.conf` and `pg_hba.conf` configuration files from the source to the destination cluster.
   - Install needed extensions in the destination database with `gppkg`. 
Security

Best practices to ensure the highest level of system security.

Basic Security Best Practices

- Secure the `gpadmin` system user. Greenplum requires a Unix user id to install and initialize the Greenplum Database system. This system user is referred to as `gpadmin` in the Greenplum documentation. The `gpadmin` user is the default database superuser in Greenplum Database, as well as the file system owner of the Greenplum installation and its underlying data files. The default administrator account is fundamental to the design of Greenplum Database. The system cannot run without it, and there is no way to limit the access of the `gpadmin` user id. This `gpadmin` user can bypass all security features of Greenplum Database. Anyone who logs on to a Greenplum host with this user id can read, alter, or delete any data, including system catalog data and database access rights. Therefore, it is very important to secure the `gpadmin` user id and only allow essential system administrators access to it. Administrators should only log in to Greenplum as `gpadmin` when performing certain system maintenance tasks (such as upgrade or expansion). Database users should never log on as `gpadmin`, and ETL or production workloads should never run as `gpadmin`.
- Assign a distinct role to each user who logs in. For logging and auditing purposes, each user who is allowed to log in to Greenplum Database should be given their own database role. For applications or web services, consider creating a distinct role for each application or service. See "Creating New Roles (Users)" in the Greenplum Database Administrator Guide.
- Use groups to manage access privileges. See "Creating Groups (Role Membership)" in the Greenplum Database Administrator Guide.
- Limit users who have the `SUPERUSER` role attribute. Roles that are superusers bypass all access privilege checks in Greenplum Database, as well as resource queuing. Only system administrators should be given superuser rights. See "Altering Role Attributes" in the Greenplum Database Administrator Guide.

Password Strength Guidelines

To protect the network from intrusion, system administrators should verify the passwords used within an organization are strong ones. The following recommendations can strengthen a password:

- Minimum password length recommendation: At least 9 characters. MD5 passwords should be 15 characters or longer.
- Mix upper and lower case letters.
- Mix letters and numbers.
- Include non-alphanumeric characters.
- Pick a password you can remember.

The following are recommendations for password cracker software that you can use to determine the strength of a password.

- John The Ripper. A fast and flexible password cracking program. It allows the use of multiple word lists and is capable of brute-force password cracking. It is available online at [http://www.openwall.com/john/](http://www.openwall.com/john/).
- Crack. Perhaps the most well-known password cracking software, Crack is also very fast, though not as easy to use as John The Ripper. It can be found online at [http://www.crypticide.com/alecm/security/crack/c50-faq.html](http://www.crypticide.com/alecm/security/crack/c50-faq.html).

The security of the entire system depends on the strength of the root password. This password should be at least 12 characters long and include a mix of capitalized letters, lowercase letters, special characters, and numbers. It should not be based on any dictionary word.

Password expiration parameters should be configured.
Ensure the following line exists within the file /etc/libuser.conf under the [import] section.

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

Ensure no lines in the [userdefaults] section begin with the following text, as these words override settings from /etc/login.defs:

- LU_SHADOWMAX
- LU_SHADOWMIN
- LU_SHADOWWARNING

Ensure the following command produces no output. Any accounts listed by running this command should be locked.

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

Note: We strongly recommend that customers change their passwords after initial setup.

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

Find all the files that are world-writable and that do not have their sticky bits set.

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

Set the sticky bit (# chmod +t {dir}) for all the directories that result from running the previous command.

Find all the files that are world-writable and fix each file listed.

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

Set the right permissions (# chmod o-w {file}) for all the files generated by running the aforementioned command.

Find all the files that do not belong to a valid user or group and either assign an owner or remove the file, as appropriate.

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

Find all the directories that are world-writable and ensure they are owned by either root or a system account (assuming only system accounts have a User ID lower than 500). If the command generates any output, verify the assignment is correct or reassign it to root.

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

Authentication settings such as password quality, password expiration policy, password reuse, password retry attempts, and more can be configured using the Pluggable Authentication Modules (PAM) framework. PAM looks in the directory /etc/pam.d for application-specific configuration information. Running authconfig or system-config-authentication will re-write the PAM configuration files, destroying any manually made changes and replacing them with system defaults.
The default `pam_cracklib` PAM module provides strength checking for passwords. To configure `pam_cracklib` to require at least one uppercase character, lowercase character, digit, and special character, as recommended by the U.S. Department of Defense guidelines, edit the file `/etc/pam.d/system-auth` to include the following parameters in the line corresponding to password requisite `pam_cracklib.so try_first_pass`:

- `retry=3`: 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.

For example:

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

These parameters can be set to reflect your security policy requirements. Note that the password restrictions are not applicable to the root password.

The `pam_tally2` PAM module provides the capability to lock out user accounts after a specified number of failed login attempts. To enforce password lockout, edit the file `/etc/pam.d/system-auth` to include the following lines:

- The first of the auth lines should include:
  ```bash
  auth required pam_tally2.so deny=5 onerr=fail unlock_time=900
  ```
- The first of the account lines should include:
  ```bash
  account required pam_tally2.so
  ```

Here, the deny parameter is set to limit the number of retries to 5 and the `unlock_time` has been set to 900 seconds to keep the account locked for 900 seconds before it is unlocked. These parameters may be configured appropriately to reflect your security policy requirements. A locked account can be manually unlocked using the `pam_tally2` utility:

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

You can use PAM to limit the reuse of recent passwords. The remember option for the `pam_unix` module can be set to remember the recent passwords and prevent their reuse. To accomplish this, edit the appropriate line in `/etc/pam.d/system-auth` to include the remember option.

For example:

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

You can set the number of previous passwords to remember to appropriately reflect your security policy requirements.

```bash
cd /etc
chown root:root passwd shadow group gshadow
chmod 644 passwd group
chmod 400 shadow gshadow
```
Encrypting Data and Database Connections

Best practices for implementing encryption and managing keys.

Encryption can be used to protect data in a Greenplum Database system in the following ways:

- Connections between clients and the master database can be encrypted with SSL. This is enabled by setting the `ssl` server configuration parameter to `on` and editing the `pg_hba.conf` file. See "Encrypting Client/Server Connections" in the Greenplum Database Administrator Guide for information about enabling SSL in Greenplum Database.
- Greenplum Database 4.2.1 and above allow SSL encryption of data in transit between the Greenplum parallel file distribution server, `gpfdist`, and segment hosts. See Encrypting `gpfdist` Connections for more information.
- Network communications between hosts in the Greenplum Database cluster can be encrypted using IPsec. An authenticated, encrypted VPN is established between every pair of hosts in the cluster. Check your operating system documentation for IPsec support, or consider a third-party solution such as that provided by Zettaset.
- The `pgcrypto` package of encryption/decryption functions protects data at rest in the database. Encryption at the column level protects sensitive information, such as passwords, Social Security numbers, or credit card numbers. See Encrypting Data in Tables using PGP for an example.

Best Practices

- Encryption ensures that data can be seen only by users who have the key required to decrypt the data.
- Encrypting and decrypting data has a performance cost; only encrypt data that requires encryption.
- Do performance testing before implementing any encryption solution in a production system.
- Server certificates in a production Greenplum Database system should be signed by a certificate authority (CA) so that clients can authenticate the server. The CA may be local if all clients are local to the organization.
- Client connections to Greenplum Database should use SSL encryption whenever the connection goes through an insecure link.
- A symmetric encryption scheme, where the same key is used to both encrypt and decrypt, has better performance than an asymmetric scheme and should be used when the key can be shared safely.
- Use functions from the `pgcrypto` package to encrypt data on disk. The data is encrypted and decrypted in the database process, so it is important to secure the client connection with SSL to avoid transmitting unencrypted data.
- Use the `gpfdists` protocol to secure ETL data as it is loaded into or unloaded from the database. See Encrypting `gpfdist` Connections.

Key Management

Whether you are using symmetric (single private key) or asymmetric (public and private key) cryptography, it is important to store the master or private key securely. There are many options for storing encryption keys, for example, on a file system, key vault, encrypted USB, trusted platform module (TPM), or hardware security module (HSM).

Consider the following questions when planning for key management:

- Where will the keys be stored?
- When should keys expire?
- How are keys protected?
- How are keys accessed?
- How can keys be recovered and revoked?
The Open Web Application Security Project (OWASP) provides a very comprehensive guide to securing encryption keys.

**Encrypting Data at Rest with pgcrypto**

The pgcrypto package for Greenplum Database provides functions for encrypting data at rest in the database. Administrators can encrypt columns with sensitive information, such as social security numbers or credit card numbers, to provide an extra layer of protection. Database data stored in encrypted form cannot be read by users who do not have the encryption key, and the data cannot be read directly from disk.

pgcrypto allows PGP encryption using symmetric and asymmetric encryption. Symmetric encryption encrypts and decrypts data using the same key and is faster than asymmetric encryption. It is the preferred method in an environment where exchanging secret keys is not an issue. With asymmetric encryption, a public key is used to encrypt data and a private key is used to decrypt data. This is slower than symmetric encryption and it requires a stronger key.

Using pgcrypto always comes at the cost of performance and maintainability. It is important to use encryption only with the data that requires it. Also, keep in mind that you cannot search encrypted data by indexing the data.

Before you implement in-database encryption, consider the following PGP limitations.

- No support for signing. That also means that it is not checked whether the encryption sub-key belongs to the master key.
- No support for encryption key as master key. This practice is generally discouraged, so this limitation should not be a problem.
- No support for several subkeys. This may seem like a problem, as this is common practice. On the other hand, you should not use your regular GPG/PGP keys with pgcrypto, but create new ones, as the usage scenario is rather different.

Greenplum Database is compiled with zlib by default; this allows PGP encryption functions to compress data before encrypting. When compiled with OpenSSL, more algorithms will be available.

Because pgcrypto functions run inside the database server, the data and passwords move between pgcrypto and the client application in clear-text. For optimal security, you should connect locally or use SSL connections and you should trust both the system and database administrators.

The pgcrypto package is not installed by default with Greenplum Database. You can download a pgcrypto package from Pivotal Network and use the Greenplum Package Manager (gppkg) to install pgcrypto across your cluster.

pgcrypto configures itself according to the findings of the main PostgreSQL configure script.

When compiled with zlib, pgcrypto encryption functions are able to compress data before encrypting.

Pgcrypto has various levels of encryption ranging from basic to advanced built-in functions. The following table shows the supported encryption algorithms.

<table>
<thead>
<tr>
<th>Value</th>
<th>Functionality</th>
<th>Built-in</th>
<th>With OpenSSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>yes</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>SHA1</td>
<td>yes</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>SHA224/256/384/512</td>
<td>yes</td>
<td></td>
<td>yes^6</td>
</tr>
</tbody>
</table>

^6 SHA2 algorithms were added to OpenSSL in version 0.9.8. For older versions, pgcrypto will use built-in code.
### Creating PGP Keys

To use PGP asymmetric encryption in Greenplum Database, you must first create public and private keys and install them.

This section assumes you are installing Greenplum Database on a Linux machine with the Gnu Privacy Guard (gpg) command line tool. Pivotal recommends using the latest version of GPG to create keys. Download and install Gnu Privacy Guard (GPG) for your operating system from [https://www.gnupg.org/download/](https://www.gnupg.org/download/). On the GnuPG website you will find installers for popular Linux distributions and links for Windows and Mac OS X installers.

1. As root, execute the following command and choose option 1 from the menu:

   ```bash
   # 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. Respond to the prompts and follow the instructions, as shown in this example:

   ```bash
   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
   ```

---

7 Any digest algorithm OpenSSL supports is automatically picked up. This is not possible with ciphers, which need to be supported explicitly.

8 AES is included in OpenSSL since version 0.9.7. For older versions, pgcrypto will use built-in code.
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. List the PGP keys by entering the following command:

```
gpg --list-secret-keys
/root/.gnupg/secring.gpg
```

```
sec  2048R/2027CC30 2015-01-13 [expires: 2016-01-13]
    ------------------------
      uid                  John Doe <jdoe@email.com>
      sub   2048R/4FD2EFBB 2015-01-13
```

2027CC30 is the public key and will be used to encrypt data in the database. 4FD2EFBB is the private
(secret) key and will be used to decrypt data.

4. Export the keys using the following commands:

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

See the pgcrypto documentation for for more information about PGP encryption functions.
Encrypting Data in Tables using PGP

This section shows how to encrypt data inserted into a column using the PGP keys you generated.

1. Dump the contents of the `public.key` file and then copy it to the clipboard:

   ```
   # cat public.key
   -----BEGIN PGP PUBLIC KEY BLOCK-----
   Version: GnuPG v2.0.14 (GNU/Linux)
   mQENBFS1z0BCADNW8qvl1VC36Kfcwd3Kpm/dijPfRyyEwB6pQkyA05jtWiXZTh
   2HislojSP6LI0cSKiQMu9LAlncec2hR1HbuvGkLg9d9texg2nnSL9Admqik/yX
   RSysVKG+qcdWuvyZ9g0oOomeyjh3n+kkkBTEMuM3f1ibMs8sh0zwMVstCUVmuHU/V
   . . .
   WH+N2laoUaoJjb2kQGhLOnFbJuevkyBylRz+hi/+8rJkc2OjQkmmK8Hkk8qb5x/
   HUMc55gHog2qQAOBpnJHg0Qq45q6p3g2/7Dbeb5WJ6K1wrFyS1aNLGWE8pvqEx
   /UUbz+dYqCtwtX0nnBulKNcmk2AeKF3YoiciCxmomdx0xhFOv9AKjjoJyC65KJci
   Pv2M1kFSzfKOAg1R3pMa8zdEte14w3vcPKQNnYuUtjfj6zoCxxv
   =XZ8J
   -----END PGP PUBLIC KEY BLOCK-----
   ```

2. Create a table called `userssn` and insert some sensitive data, social security numbers for Bob and Alice, in this example. Paste the public.key contents after "dearmor("

   ```sql
   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)
   mQENBFS1z0BCADNW8qvl1VC36Kfcwd3Kpm/dijPfRyyEwB6pQkyA05jtWiXZTh
   2HislojSP6LI0cSKiQMu9LAlncec2hR1HbuvGkLg9d9texg2nnSL9Admqik/yX
   RSysVKG+qcdWuvyZ9g0oOomeyjh3n+kkkBTEMuM3f1ibMs8sh0zwMVstCUVmuHU/V
   . . .
   WH+N2laoUaoJjb2kQGhLOnFbJuevkyBylRz+hi/+8rJkc2OjQkmmK8Hkk8qb5x/
   HUMc55gHog2qQAOBpnJHg0Qq45q6p3g2/7Dbeb5WJ6K1wrFyS1aNLGWE8pvqEx
   /UUbz+dYqCtwtX0nnBulKNcmk2AeKF3YoiciCxmomdx0xhFOv9AKjjoJyC65KJci
   Pv2M1kFSzfKOAg1R3pMa8zdEte14w3vcPKQNnYuUtjfj6zoCxxv
   =XZ8J
   -----END PGP PUBLIC KEY BLOCK-----' AS pubkey) AS keys;
   ```

3. Verify that the `ssn` column is encrypted.

   ```sql
   test_db=# select * from userssn;
   ssn_id | 1
   username | Alice
   ssn     | 301\337\273\011\010\000\272\227\010\341\216\360\217C\020\261\136
   [\217\034\313\C\354\037\233\000\326\005\E\364W\252=zz\010(\367R
   \213\377\256\247\364f\034\331\033\323\021\005\242\022\473\271\228\225\032\271\228\010\337\255\322\021\330\"\020\%\254\274]\R
   \213\377\256\247\364f\034\331\033\323\021\005\242\022\473\271\228\225\032\271\228\010\337\255\322\021\330\"\020\%\254\274]\R
   \213\377\256\247\364f\034\331\033\323\021\005\242\022\473\271\228\225\032\271\228\010\337\255\322\021\330\"\020\%\254\274]\R
   \213\377\256\247\364f\034\331\033\323\021\005\242\022\473\271\228\225\032\271\228\010\337\255\322\021\330\"\020\%\254\274]\R
   \213\377\256\247\364f\034\331\033\323\021\005\242\022\473\271\228\225\032\271\228\010\337\255\322\021\330\"\020\%\254\274]\R
   \213\377\256\247\364f\034\331\033\323\021\005\242\022\473\271\228\225\032\271\228\010\337\255\322\021\330\"\020\%\254\274]\R
   \213\377\256\247\364f\034\331\033\323\021\005\242\022\473\271\228\225\032\271\228\010\337\255\322\021\330\"\020\%\254\274]\R
   ```
4. Extract the public.key ID from the database:

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

mQENBFS1z0BCADNw80vk1VC36Kfcd3Kpm/dijPfRyyEwB6QyA05jtWizZTh
2HislojSP6L1cSKiQM9ALnecce20RIhBhuvGkJGsd9teXg2nnSL9Admqik/yX
R5sysyVGQ+qcdDwuyv29g0oOmeyjhc3n+kkbRTEmuM3f1bsM68h0zwMvstCUMuH/U/V
. . .
WH+N2las0UaoJj2kQGhLONfBruevkyBy1Rz+hI/+8rJkc20jQkmmK8Hkk8q5b5x/
HMUC55Hq0q2qAY0BpmJHgOQq45Q6p3G2/7Dbek5WJ6KUrFy51sN1GWE8pvqEx
/002B+YdCyvtxVOnnB1KNCmk2AKeEcFK3YoilCxm0DxhFv9AkjjoDyC65KJc1
Pv2MikP2fKOa1R3lpMa8zdDeEt1w3vckP0NnYU7tjfj6zOcxv
=2X8J
-----END PGP PUBLIC KEY BLOCK-----'));
```

This shows that the PGP key ID used to encrypt the **ssn** column is 9D4D255F4FD2EFBB. It is recommended to perform this step whenever a new key is created and then store the ID for tracking.
You can use this key to see which key pair was used to encrypt the data:

```sql
SELECT username, pgp_key_id(ssn) As key_used 
FROM userssn;
```

<table>
<thead>
<tr>
<th>username</th>
<th>key_used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>9D4D255F4FD2EFBB</td>
</tr>
<tr>
<td>Alice</td>
<td>9D4D255F4FD2EFBB</td>
</tr>
</tbody>
</table>

**Note**: Different keys may have the same ID. This is rare, but is a normal event. The client application should try to decrypt with each one to see which fits — like handling `ANYKEY`. See `pgp_key_id()` in the pgcrypto documentation.

5. Decrypt the data using the private key.

```sql
SELECT username, pgp_pub_decrypt(ssn, keys.privkey) AS decrypted_ssn FROM userssn 
CROSS JOIN 
(SELECT dearmor('-----BEGIN PGP PRIVATE KEY BLOCK-----
Version: GnuPG v2.0.14 (GNU/Linux)
lQOYBFS12f0BdNw8Qyk1V1C36Kfcdw3kP/jm/djFrPryyEB6pQkyA05jtWixZTh2HsiolojSP6I10cSKiqMU9LAlncec2hRIHbvuVGKLGSg9d5eq2nnSL9Admqj/k/yX
R5syVKGq+cDwvY29g9O0Omyjhc3n/kkkBRTEmU3f1bMs8shOwMrVstCUVuHu/U/vGv5rjAae8PvYuDSJ7C7416w7SOH3RiRc7fjR6xyDd4V213ctd44b18/I71hqQvY2N2
/Hzb5j2ym7ttw3jsXAwX2gPI9nsDqGoyQDy/o9nNgCBE9igiq6ZFnFE6wbh+icSM0DIK5/GAlR6HcOZ1HF8Ecvrulq1kJnAEBAAEAEB/wNfjvjPlbrfJjIm/j
Xw1NNm+i4v2Ur7zQCGPf67lVqcy2JuoqVzr8zB16mv16x5EU1I2Y7BNAA
fe0PaM4Wv+Xr94Z2cPbPgwanRN33GQy41rBTrvgqWy/kmpbd87I7w12ZdNNXm
02iZsfrq41Rt02xZ1Jhr4kP6f7tmoOHTv1rSR0bW0vHuEy7TcwMde9H9BQe51Fr
n9C11uQgBn4/acTtCC1WAjRrWz0XAsXixTICTlV+9n09ymecMxWnNFPIHyAM
OPFnuZ/D1z1rCRJhNh5b1J62yUM5zdQvUNvznxktqReOENsXm0gfGMgpxQHqgUmZb7c
6UyBBADSCXHHPsf/VpVtMm5p1GrNORz2r2rzU9+j+po2Zd2gjkt5/g/IXRlkB4uQ1l
emu27zwr9d9EX7msOnvDqS5ibqQ4dOjDJIdcHMeSRQZ1urErb1B75VJ3htImb1jpn
4Jx6SWRXPUJPGXGI87u0uBHOLwij7M2PFW711aa+MLEA9j AJQwQA+sr9BKPL4YA2
r5nE72gsClCkC0wIcdRf1GtbwYDmpMy2oArKJK0TMG6rCXJxrf6lqin8lW/L
/gZntmTch35Mq/ZMa/bn4VMPE/11wXxZVkJ3ls,qyQX/A7ac7B7DHOKxFciExW
M50A/AhMkmX1W1RmW3cnYi8w3q7X40EAL/w54FVvPp3pscD656AaP4MU02R3
TisuNvmEmwD5GxuW8A857rveVU5y24B8lvCuj1C7cena/MLhIt8f0rsCJC057
Bf+iIC/VNEv4k4uDb410gqHNJYyfBFwClw+n/EnXCZ2YQINMLa14MFvug/RfTH4
nkwtZ/91lsAh/20HHR1Cy9aQx520X3a52vAtXQYvYwLsNcv0v6b764EwEWEc
ACgsFA1s12u0OCWmFCQHHm4AGCwKibmCvBUIAgkKcWcM4tAh4BAHAAoEAJEd9
cl4Jq88bwvF/3YVsPsQ1llowRJnxvXgt1bY7brfVU5yk/PPZYQes9UpdCL3CMRk
8gAm9bX55k08q2UXS2LC6F0PoEW4uWmGfYf8JR0C3oeezTkmCBW81lbU0QGetvX
vdxLPuGFCE7hWvQe9HStntiTLxGov1mJAw7TAoCCXLbyu2h9RF5v5L0QdKzcOHY
h5iQxaQnt0TTeEepb9T1Iwcnt3CWSU50PDoGAoAanjDzKB8Q87v74fhhE1EZV
zhB8pajR6C2CS5HFkgpBqJNxTOk54NBXADn4eUTXPSwNPi46qoAp9UQqsgfSyB1X
DOTT2UGhutAMCwAM7TepePV7jZ0/INfsBdA5gELV1V/QEIANafbfD+8q8MCADOoi
pMlBF/jRqt3Uoc4BtQIcAdyazFz0UtUSF/7Zro2us99G1ARqLWd8Ecqlj/xmcfJ
i2yUam62AzxFXCNgH5ydStMTJ2dp5wOjwCGW/CZLZ4uwz0FPdzkiPv9RDw6e5
MNLRtJrSp4h5so2aDkb04Exs304mJynaw/riEDDCWU4T0lv3hSKCpke6LcwsX7-7
liozp+aANm0P0ywpf14R3UUMP70VlbeFwQ2bVZL3lLOuHRPcCZla+pZzbEks16
j7q7v9Gk9DZC7z5eTlwAljGIrlFk03v9Qh0v/8YAnkAmow2zR1blyFg2KbzhunY
mn2YvKuUAEQEAQAQ/A7d4hDrmmmz3XjQ6uFwAselP1Rbn7in2E2JEN8aUpF05Q2pZKU
cl1STZwjgtqMAIAgYAsDkOhfhcxancr0UVTvyVgKyTT3e5M1RFpKjRX2h0mZKdwy67B
Mu95V2Okl16OCo8d06+s2fmgKxGqC42sysJ+qxtk33H9GYXxMC0dAV2vR3N4Twt3
Uct7/zh/k16ImbaLd7PD7dX7THn2QsMo81lrgtPE8qrDT0oomTnRUodToyejEqKyIR
Owasp8n8g2CSpaX5sreY06HFrYFX5x2ea1hOQYLGZ0D0RmxV7QfCn/7n+iHjupndC
Nd2aB8DvX3yQ3sdAmvLzhFacd7jX311wv/j0moFOK6AOcZ3e63SKNNknniQ11RMJp
gavo6Ux/ZGMLsTzWnouI+Kr8/db0GIAY2Zu0Ab4tfQXEAPo9X4A92KqjOqOX
```
If you created a key with passphrase, you may have to enter it here. However for the purpose of this example, the passphrase is blank.

### Encrypting gpfdist Connections

The gpfdists protocol is a secure version of the gpfdist protocol that securely identifies the file server and the Greenplum Database and encrypts the communications between them. Using gpfdists protects against eavesdropping and man-in-the-middle attacks.

The gpfdists protocol implements client/server SSL security with the following notable features:

- Client certificates are required.
- Multilingual certificates are not supported.
- A Certificate Revocation List (CRL) is not supported.
- The TLSv1 protocol is used with the TLS_RSA_WITH_AES_128_CBC_SHA encryption algorithm. These SSL parameters cannot be changed.
- SSL renegotiation is supported.
- The SSL ignore host mismatch parameter is set to false.
- Private keys containing a passphrase are not supported for the gpfdist file server (server.key) or for the Greenplum Database (client.key).
- It is the user's responsibility to issue certificates that are appropriate for the operating system in use. Generally, converting certificates to the required format is supported, for example using the SSL Converter at https://www.sslshopper.com/ssl-converter.html.

A gpfdist server started with the --ssl option can only communicate with the gpfdists protocol. A gpfdist server started without the --ssl option can only communicate with the gpfdist protocol. For more detail about gpfdist refer to the Greenplum Database Administrator Guide.

There are two ways to enable the gpfdists protocol:

- Run gpfdist with the --ssl option and then use the gpfdists protocol in the LOCATION clause of a CREATE EXTERNAL TABLE statement.
- Use a YAML control file with the SSL option set to true and run gpload. Running gpload starts the gpfdist server with the --ssl option and then uses the gpfdists protocol.

When using gpfdists, the following client certificates must be located in the $PGDATA/gpfdists directory on each segment:
• The client certificate file, client.crt
• The client private key file, client.key
• The trusted certificate authorities, root.crt

**Important:** Do not protect the private key with a passphrase. The server does not prompt for a passphrase for the private key, and loading data fails with an error if one is required.

When using `gpload` with SSL you specify the location of the server certificates in the YAML control file. When using `gpfdist` with SSL, you specify the location of the server certificates with the --ssl option.

The following example shows how to securely load data into an external table. The example creates a readable external table named `ext_expenses` from all files with the `.txt` extension, using the `gpfdists` protocol. The files are formatted with a pipe (`|`) as the column delimiter and an empty space as null.

1. Run `gpfdist` with the --ssl option on the segment hosts.
2. Log into the database and execute the following command:

```sql
=# 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 ' ' ) ;
```
Accessing a Kerberized Hadoop Cluster

Using external tables and the gphdfs protocol, Greenplum Database can read files from and write files to a Hadoop File System (HDFS). Greenplum segments read and write files in parallel from HDFS for fast performance.

When a Hadoop cluster is secured with Kerberos (“Kerberized”), Greenplum Database must be configured to allow the Greenplum Database gpadmin role, which owns external tables in HDFS, to authenticate through Kerberos. This topic provides the steps for configuring Greenplum Database to work with a Kerberized HDFS, including verifying and troubleshooting the configuration.

- Prerequisites
- Configuring the Greenplum Cluster
- Creating and Installing Keytab Files
- Configuring gphdfs for Kerberos
- Testing Greenplum Database Access to HDFS
- Troubleshooting HDFS with Kerberos

Prerequisites

Make sure the following components are functioning and accessible on the network:

- Greenplum Database cluster—either a Pivotal Greenplum Database software-only cluster, or a Dell EMC Data Computing Appliance (DCA).
- Kerberos-secured Hadoop cluster. See the Greenplum Database Release Notes for supported Hadoop versions.
- Kerberos Key Distribution Center (KDC) server.

Configuring the Greenplum Cluster

The hosts in the Greenplum Cluster must have a Java JRE, Hadoop client files, and Kerberos clients installed.

Follow these steps to prepare the Greenplum Cluster.

1. Install a Java 1.6 or later JRE on all Greenplum cluster hosts.
   Match the JRE version the Hadoop cluster is running. You can find the JRE version by running `java --version` on a Hadoop node.

2. (Optional) Confirm that Java Cryptography Extension (JCE) is present.
   The default location of the JCE libraries is JAVA_HOME/lib/security. If a JDK is installed, the directory is JAVA_HOME/jre/lib/security. The files local_policy.jar and US_export_policy.jar should be present in the JCE directory.

   The Greenplum cluster and the Kerberos server should, preferably, use the same version of the JCE libraries. You can copy the JCE files from the Kerberos server to the Greenplum cluster, if needed.

3. Set the JAVA_HOME environment variable to the location of the JRE in the .bashrc or .bash_profile file for the gpadmin account. For example:

   ```bash
   export JAVA_HOME=/usr/java/default
   ```

4. Source the .bashrc or .bash_profile file to apply the change to your environment. For example:

   ```bash
   $ source ~/.bashrc
   ```
5. Install the Kerberos client utilities on all cluster hosts. Ensure the libraries match the version on the KDC server before you install them.

   For example, the following command installs the Kerberos client files on Red Hat or CentOS Linux:

   ```bash
   $ sudo yum install krb5-libs krb5-workstation
   ```

   Use the `kinit` command to confirm the Kerberos client is installed and correctly configured.

6. Install Hadoop client files on all hosts in the Greenplum Cluster. Refer to the documentation for your Hadoop distribution for instructions.

7. Set the Greenplum Database server configuration parameters for Hadoop. The `gp_hadoop_target_version` parameter specifies the version of the Hadoop cluster. See the `Greenplum Database Release Notes` for the target version value that corresponds to your Hadoop distribution. The `gp_hadoop_home` parameter specifies the Hadoop installation directory.

   ```bash
   $ gpconfig -c gp_hadoop_target_version -v "hdp2"
   $ gpconfig -c gp_hadoop_home -v "/usr/lib/hadoop"
   ```

   See the `Greenplum Database Reference Guide` for more information.

8. Reload the updated `postgresql.conf` files for master and segments:

   ```bash
   gpstop -u
   ```

   You can confirm the changes with the following commands:

   ```bash
   $ gpconfig -s gp_hadoop_target_version
   $ gpconfig -s gp_hadoop_home
   ```

9. Grant Greenplum Database gphdfs protocol privileges to roles that own external tables in HDFS, including `gpadmin` and other superuser roles. Grant `SELECT` privileges to enable creating readable external tables in HDFS. Grant `INSERT` privileges to enable creating writable external tables on HDFS.

   ```bash
   #= GRANT SELECT ON PROTOCOL gphdfs TO gpadmin;
   #= GRANT INSERT ON PROTOCOL gphdfs TO gpadmin;
   ```

10. Grant Greenplum Database external table privileges to external table owner roles:

    ```sql
    ALTER ROLE HDFS_USER CREATEEXXTABLE (type='readable');
    ALTER ROLE HDFS_USER CREATEEXXTABLE (type='writable');
    ```

    **Note:** It is best practice to review database privileges, including gphdfs external table privileges, at least annually.

**Creating and Installing Keytab Files**

1. Log in to the KDC server as root.
2. Use the `kadmin.local` command to create a new principal for the `gpadmin` user:

   ```bash
   # kadmin.local -q "addprinc -randkey gpadmin@LOCAL.DOMAIN"
   ```

3. Use `kadmin.local` to generate a Kerberos service principal for each host in the Greenplum Database cluster. The service principal should be of the form `name@role@REALM`, where:

   - `name` is the gphdfs service user name. This example uses `gphdfs`.
   - `role` is the DNS-resolvable host name of a Greenplum cluster host (the output of the `hostname -f` command).
   - `REALM` is the Kerberos realm, for example `LOCAL.DOMAIN`. 
For example, the following commands add service principals for four Greenplum Database hosts, mdw.example.com, smdw.example.com, sdw1.example.com, and sdw2.example.com:

```
# kadmin.local -q "addprinc -randkey gphdfs/mdw.example.com@LOCAL.DOMAIN"
# kadmin.local -q "addprinc -randkey gphdfs/smdw.example.com@LOCAL.DOMAIN"
# kadmin.local -q "addprinc -randkey gphdfs/sdw1.example.com@LOCAL.DOMAIN"
# kadmin.local -q "addprinc -randkey gphdfs/sdw2.example.com@LOCAL.DOMAIN"
```

Create a principal for each Greenplum cluster host. Use the same principal name and realm, substituting the fully-qualified domain name for each host.

4. Generate a keytab file for each principal that you created (gpadmin and each gphdfs service principal). You can store the keytab files in any convenient location (this example uses the directory `/etc/security/keytabs`). You will deploy the service principal keytab files to their respective Greenplum host machines in a later step:

```
# kadmin.local -q "xst -k /etc/security/keytabs/gphdfs.service.keytab gpadmin@LOCAL.DOMAIN"
# kadmin.local -q "xst -k /etc/security/keytabs/mdw.service.keytab gpadmin/mdw gphdfs/mdw.example.com@LOCAL.DOMAIN"
# kadmin.local -q "xst -k /etc/security/keytabs/smdw.service.keytab gpadmin/smdw gphdfs/smdw.example.com@LOCAL.DOMAIN"
# kadmin.local -q "xst -k /etc/security/keytabs/sdw1.service.keytab gpadmin/sdw1 gphdfs/sdw1.example.com@LOCAL.DOMAIN"
# kadmin.local -q "xst -k /etc/security/keytabs/sdw2.service.keytab gpadmin/sdw2 gphdfs/sdw2.example.com@LOCAL.DOMAIN"
# kadmin.local -q "listprincs"
```

5. Change the ownership and permissions on `gphdfs.service.keytab` as follows:

```
# chown gpadmin:gpadmin /etc/security/keytabs/gphdfs.service.keytab
# chmod 440 /etc/security/keytabs/gphdfs.service.keytab
```

6. Copy the keytab file for `gpadmin@LOCAL.DOMAIN` to the Greenplum master host:

```
# scp /etc/security/keytabs/gphdfs.service.keytab mdw_fqdn:/home/gpadmin/gphdfs.service.keytab
```

7. Copy the keytab file for each service principal to its respective Greenplum host:

```
# scp /etc/security/keytabs/mdw.service.keytab mdw_fqdn:/home/gpadmin/mdw.service.keytab
# scp /etc/security/keytabs/smdw.service.keytab smdw_fqdn:/home/gpadmin/smdw.service.keytab
# scp /etc/security/keytabs/sdw1.service.keytab sdw1_fqdn:/home/gpadmin/sdw1.service.keytab
# scp /etc/security/keytabs/sdw2.service.keytab sdw2_fqdn:/home/gpadmin/sdw2.service.keytab
```

**Configuring gphdfs for Kerberos**

1. Edit the Hadoop `core-site.xml` client configuration file on all Greenplum cluster hosts. Enable service-level authorization for Hadoop by setting the `hadoop.security.authorization` property to true. For example:

```
<property>
  <name>hadoop.security.authorization</name>
  <value>true</value>
</property>
```
2. Edit the yarn-site.xml client configuration file on all cluster hosts. Set the resource manager address and yarn Kerberos service principle. For example:

```xml
  <property>
    <name>yarn.resourcemanager.address</name>
    <value>hostname:8032</value>
  </property>
  <property>
    <name>yarn.resourcemanager.principal</name>
    <value>yarn/hostname@DOMAIN</value>
  </property>
```

3. Edit the hdfs-site.xml client configuration file on all cluster hosts. Set properties to identify the NameNode Kerberos principals, the location of the Kerberos keytab file, and the principal it is for:

- **dfs.namenode.kerberos.principal** - the Kerberos principal name the gphdfs protocol will use for the NameNode, for example gpadmin@LOCAL.DOMAIN.
- **dfs.namenode.https.principal** - the Kerberos principal name the gphdfs protocol will use for the NameNode's secure HTTP server, for example gpadmin@LOCAL.DOMAIN.
- **com.emc.greenplum.gpdb.hdfsconnector.security.user.keytab.file** - the path to the keytab file for the Kerberos HDFS service, for example /home/gpadmin/mdw.service.keytab.
- **com.emc.greenplum.gpdb.hdfsconnector.security.user.name** - the gphdfs service principal for the host, for example gphdfs/mdw.example.com@LOCAL.DOMAIN.

For example:

```xml
  <property>
    <name>dfs.namenode.kerberos.principal</name>
    <value>gphdfs/gpadmin@LOCAL.DOMAIN</value>
  </property>
  <property>
    <name>dfs.namenode.https.principal</name>
    <value>gphdfs/gpadmin@LOCAL.DOMAIN</value>
  </property>
  <property>
    <name>com.emc.greenplum.gpdb.hdfsconnector.security.user.keytab.file</name>
    <value>/home/gpadmin/gpadmin.hdfs.keytab</value>
  </property>
  <property>
    <name>com.emc.greenplum.gpdb.hdfsconnector.security.user.name</name>
    <value>gpadmin@LOCAL.DOMAIN</value>
  </property>
```

**Testing Greenplum Database Access to HDFS**

Confirm that HDFS is accessible via Kerberos authentication on all hosts in the Greenplum cluster. For example, enter the following command to list an HDFS directory:

```
hdfs dfs -ls hdfs://namenode:8020
```

**Create a Readable External Table in HDFS**

Follow these steps to verify that you can create a readable external table in a Kerberized Hadoop cluster.

1. Create a comma-delimited text file, `test1.txt`, with contents such as the following:

```
25, Bill
19, Anne
32, Greg
```
2. Persist the sample text file in HDFS:

```bash
hdfs dfs -put test1.txt hdfs://namenode:8020/tmp
```

3. Log in to Greenplum Database and create a readable external table that points to the `test1.txt` file in Hadoop:

```sql
CREATE EXTERNAL TABLE test_hdfs (age int, name text)
LOCATION('gphdfs://namenode:8020/tmp/test1.txt')
FORMAT 'text' (delimiter ',');
```

4. Read data from the external table:

```sql
SELECT * FROM test_hdfs;
```

## Create a Writable External Table in HDFS

Follow these steps to verify that you can create a writable external table in a Kerberized Hadoop cluster. The steps use the `test_hdfs` readable external table created previously.

1. Log in to Greenplum Database and create a writable external table pointing to a text file in HDFS:

```sql
CREATE WRITABLE EXTERNAL TABLE test_hdfs2 (LIKE test_hdfs)
LOCATION ('gphdfs://namenode:8020/tmp/test2.txt')
FORMAT 'text' (DELIMITER ',');
```

2. Load data into the writable external table:

```sql
INSERT INTO test_hdfs2
SELECT * FROM test_hdfs;
```

3. Check that the file exists in HDFS:

```bash
hdfs dfs -ls hdfs://namenode:8020/tmp/test2.txt
```

4. Verify the contents of the external file:

```bash
hdfs dfs -cat hdfs://namenode:8020/tmp/test2.txt
```

## Troubleshooting HDFS with Kerberos

### Forcing Classpaths

If you encounter "class not found" errors when executing `SELECT` statements from `gphdfs` external tables, edit the `$GPHOME/lib/hadoop-env.sh` file and add the following lines towards the end of the file, before the `JAVA_LIBRARY_PATH` is set. Update the script on all of the cluster hosts.

```bash
if [ -d "/usr/hdp/current" ]; then
    for f in /usr/hdp/current/**/*.*.jar; do
        CLASSPATH=$CLASSPATH:$f;
    done
fi
```
**Enabling Kerberos Client Debug Messages**
To see debug messages from the Kerberos client, edit the `$GPHOME/lib/hadoop-env.sh` client shell script on all cluster hosts and set the `HADOOP_OPTS` variable as follows:

```shell
export HADOOP_OPTS="-Djava.net.preferredIPv4Stack=true -Dsun.security.krb5.debug=true ${HADOOP_OPTS}"
```

**Adjusting JVM Process Memory on Segment Hosts**
Each segment launches a JVM process when reading or writing an external table in HDFS. To change the amount of memory allocated to each JVM process, configure the `GP_JAVA_OPT` environment variable.

Edit the `$GPHOME/lib/hadoop-env.sh` client shell script on all cluster hosts.

For example:

```shell
export GP_JAVA_OPT=-Xmx1000m
```

**Verify Kerberos Security Settings**
Review the `/etc/krb5.conf` file:

- If AES256 encryption is not disabled, ensure that all cluster hosts have the JCE Unlimited Strength Jurisdiction Policy Files installed.
- Ensure all encryption types in the Kerberos keytab file match definitions in the `krb5.conf` file.

```shell
cat /etc/krb5.conf | egrep supported_enctypes
```

**Test Connectivity on an Individual Segment Host**
Follow these steps to test that a single Greenplum Database host can read HDFS data. This test method executes the Greenplum HDFSReader Java class at the command-line, and can help to troubleshoot connectivity problems outside of the database.

1. Save a sample data file in HDFS.

```shell
hdfs dfs -put test1.txt hdfs://namenode:8020/tmp
```

2. On the segment host to be tested, create an environment script, `env.sh`, like the following:

```shell
export JAVA_HOME=/usr/java/default
echo HADOOP_HOME=/usr/lib/hadoop
echo GP_HADOOP_CON_VERSION=hdp2
echo GP_HADOOP_CON_JARDIR=/usr/lib/hadoop
```

3. Source all environment scripts:

```shell
source /usr/local/greenplum-db/greenplum_path.sh
source env.sh
source $GPHOME/lib/hadoop-env.sh
```

4. Test the Greenplum Database HDFS reader:

```shell
java com.emc.greenplum.gpdb.hdfsconnector.HDFSReader 0 32 TEXT hdp2 gphdfs://namenode:8020/tmp/test1.txt
```
Tuning SQL Queries

The Greenplum Database cost-based optimizer evaluates many strategies for executing a query and chooses the least costly method.

Like other RDBMS optimizers, the Greenplum optimizer takes into account factors such as the number of rows in tables to be joined, availability of indexes, and cardinality of column data when calculating the costs of alternative execution plans. The optimizer also accounts for the location of the data, preferring to perform as much of the work as possible on the segments and to minimize the amount of data that must be transmitted between segments to complete the query.

When a query runs slower than you expect, you can view the plan the optimizer selected as well as the cost it calculated for each step of the plan. This will help you determine which steps are consuming the most resources and then modify the query or the schema to provide the optimizer with more efficient alternatives. You use the SQL `EXPLAIN` statement to view the plan for a query.

The optimizer produces plans based on statistics generated for tables. It is important to have accurate statistics to produce the best plan. See `Updating Statistics with ANALYZE` in this guide for information about updating statistics.

How to Generate Explain Plans

The `EXPLAIN` and `EXPLAIN ANALYZE` statements are useful tools to identify opportunities to improve query performance. `EXPLAIN` displays the query plan and estimated costs for a query, but does not execute the query. `EXPLAIN ANALYZE` executes the query in addition to displaying the query plan. `EXPLAIN ANALYZE` discards any output from the `SELECT` statement; however, other operations in the statement are performed (for example, `INSERT`, `UPDATE`, or `DELETE`). To use `EXPLAIN ANALYZE` on a DML statement without letting the command affect the data, explicitly use `EXPLAIN ANALYZE` in a transaction (`BEGIN; EXPLAIN ANALYZE ...; ROLLBACK;`).

`EXPLAIN ANALYZE` runs the statement in addition to displaying the plan with additional information as follows:

- Total elapsed time (in milliseconds) to run the query
- Number of workers (segments) involved in a plan node operation
- Maximum number of rows returned by the segment (and its segment ID) that produced the most rows for an operation
- The memory used by the operation
- Time (in milliseconds) it took to retrieve the first row from the segment that produced the most rows, and the total time taken to retrieve all rows from that segment.

How to Read Explain Plans

An explain plan is a report detailing the steps the Greenplum Database optimizer has determined it will follow to execute a query. The plan is a tree of nodes, read from bottom to top, with each node passing its result to the node directly above. Each node represents a step in the plan, and one line for each node identifies the operation performed in that step—for example, a scan, join, aggregation, or sort operation. The node also identifies the method used to perform the operation. The method for a scan operation, for example, may be a sequential scan or an index scan. A join operation may perform a hash join or nested loop join.

Following is an explain plan for a simple query. This query finds the number of rows in the contributions table stored at each segment.
This plan has seven nodes – Dynamic Table Scan, Function Scan, Result, Sequence, Redistribute Motion, HashAggregate, and finally Gather Motion. Each node contains three cost estimates: cost (in sequential page reads), the number of rows, and the width of the rows.

The cost is a two-part estimate. A cost of 1.0 is equal to one sequential disk page read. The first part of the estimate is the start-up cost, which is the cost of getting the first row. The second estimate is the total cost, the cost of getting all of the rows.

The rows estimate is the number of rows output by the plan node. The number may be lower than the actual number of rows processed or scanned by the plan node, reflecting the estimated selectivity of WHERE clause conditions. The total cost assumes that all rows will be retrieved, which may not always be the case (for example, if you use a LIMIT clause).

The width estimate is the total width, in bytes, of all the columns output by the plan node.

The cost estimates in a node include the costs of all its child nodes, so the top-most node of the plan, usually a Gather Motion, has the estimated total execution costs for the plan. This is this number that the query planner seeks to minimize.

Scan operators scan through rows in a table to find a set of rows. There are different scan operators for different types of storage. They include the following:

- Seq Scan on heap tables — scans all rows in the table.
- Append-only Scan — scans rows in row-oriented append-only tables.
- Append-only Columnar Scan — scans rows in column-oriented append-only tables.
- Index Scan — traverses a B-tree index to fetch the rows from the table.
- Bitmap Append-only Row-oriented Scan — gathers pointers to rows in an append-only table from an index and sorts by location on disk.
- Dynamic Table Scan — chooses partitions to scan using a partition selection function. The Function Scan node contains the name of the partition selection function, which can be one of the following:
  - gp_partition_expansion — selects all partitions in the table. No partitions are eliminated.
  - gp_partition_selection — chooses a partition based on an equality expression.
  - gp_partition_inversion — chooses partitions based on a range expression.

The Function Scan node passes the dynamically selected list of partitions to the Result node which is passed to the Sequence node.

Join operators include the following:

- Hash Join – builds a hash table from the smaller table with the join column(s) as hash key. Then scans the larger table, calculating the hash key for the join column(s) and probing the hash table to find the rows with the same hash key. Hash joins are typically the fastest joins in Greenplum Database. The Hash Cond in the explain plan identifies the columns that are joined.
- Nested Loop – iterates through rows in the larger dataset, scanning the rows in the smaller dataset on each iteration. The Nested Loop join requires the broadcast of one of the tables so that all rows in one table can be compared to all rows in the other table. It performs well for small tables or tables that are limited by using an index. It is also used for Cartesian joins and range joins. There are performance implications when using a Nested Loop join with large tables. For plan nodes that contain a Nested Loop join operator, validate the SQL and ensure that the results are what is intended. Set the enable_nestloop server configuration parameter to OFF (default) to favor Hash Join.
- Merge Join – sorts both datasets and merges them together. A merge join is fast for pre-ordered data, but is very rare in the real world. To favor Merge Joins over Hash Joins, set the enable_mergejoin system configuration parameter to ON.

Some query plan nodes specify motion operations. Motion operations move rows between segments when required to process the query. The node identifies the method used to perform the motion operation. Motion operators include the following:

- Broadcast motion – each segment sends its own, individual rows to all other segments so that every segment instance has a complete local copy of the table. A Broadcast motion may not be as optimal
as a Redistribute motion, so the optimizer typically only selects a Broadcast motion for small tables. A Broadcast motion is not acceptable for large tables. In the case where data was not distributed on the join key, a dynamic redistribution of the needed rows from one of the tables to another segment is performed.

- Redistribute motion – each segment rehashes the data and sends the rows to the appropriate segments according to hash key.
- Gather motion – result data from all segments is assembled into a single stream. This is the final operation for most query plans.

Other operators that occur in query plans include the following:

- Materialize – the planner materializes a subselect once so it does not have to repeat the work for each top-level row.
- InitPlan – a pre-query, used in dynamic partition elimination, performed when the values the planner needs to identify partitions to scan are unknown until execution time.
- Sort – sort rows in preparation for another operation requiring ordered rows, such as an Aggregation or Merge Join.
- Group By – groups rows by one or more columns.
- Group/Hash Aggregate – aggregates rows using a hash.
- Append – concatenates data sets, for example when combining rows scanned from partitions in a partitioned table.
- Filter – selects rows using criteria from a WHERE clause.
- Limit – limits the number of rows returned.

### Optimizing Greenplum Queries

This topic describes Greenplum Database features and programming practices that can be used to enhance system performance in some situations.

To analyze query plans, first identify the plan nodes where the estimated cost to perform the operation is very high. Determine if the estimated number of rows and cost seems reasonable relative to the number of rows for the operation performed.

If using partitioning, validate that partition elimination is achieved. To achieve partition elimination the query predicate (WHERE clause) must be the same as the partitioning criteria. Also, the WHERE clause must not contain an explicit value and cannot contain a subquery.

Review the execution order of the query plan tree. Review the estimated number of rows. You want the execution order to build on the smaller tables or hash join result and probe with larger tables. Optimally, the largest table is used for the final join or probe to reduce the number of rows being passed up the tree to the topmost plan nodes. If the analysis reveals that the order of execution builds and/or probes is not optimal ensure that database statistics are up to date. Running ANALYZE will likely address this and produce an optimal query plan.

Look for evidence of computational skew. Computational skew occurs during query execution when execution of operators such as Hash Aggregate and Hash Join cause uneven execution on the segments. More CPU and memory are used on some segments than others, resulting in less than optimal execution. The cause could be joins, sorts, or aggregations on columns that have low cardinality or non-uniform distributions. You can detect computational skew in the output of the EXPLAIN ANALYZE statement for a query. Each node includes a count of the maximum rows processed by any one segment and the average rows processed by all segments. If the maximum row count is much higher than the average, at least one segment has performed much more work than the others and computational skew should be suspected for that operator.

Identify plan nodes where a Sort or Aggregate operation is performed. Hidden inside an Aggregate operation is a Sort. If the Sort or Aggregate operation involves a large number of rows, there is an opportunity to improve query performance. A HashAggregate operation is preferred over Sort and Aggregate operations when a large number of rows are required to be sorted. Usually a Sort operation is
chosen by the optimizer due to the SQL construct; that is, due to the way the SQL is written. Most Sort operations can be replaced with a HashAggregate if the query is rewritten. To favor a HashAggregate operation over a Sort and Aggregate operation ensure that the `enable_groupagg` server configuration parameter is set to `ON`.

When an explain plan shows a broadcast motion with a large number of rows, you should attempt to eliminate the broadcast motion. One way to do this is to use the `gp_segments_for_planner` server configuration parameter to increase the cost estimate of the motion so that alternatives are favored. The `gp_segments_for_planner` variable tells the query planner how many primary segments to use in its calculations. The default value is zero, which tells the planner to use the actual number of primary segments in estimates. Increasing the number of primary segments increases the cost of the motion, thereby favoring a redistribute motion over a broadcast motion. For example, setting `gp_segments_for_planner = 100000` tells the planner that there are 100,000 segments. Conversely, to influence the optimizer to broadcast a table and not redistribute it, set `gp_segments_for_planner` to a low number, for example 2.

**Greenplum Grouping Extensions**

Greenplum Database aggregation extensions to the `GROUP BY` clause can perform some common calculations in the database more efficiently than in application or procedure code:

- `GROUP BY ROLLUP(col1, col2, col3)`
- `GROUP BY CUBE(col1, col2, col3)`
- `GROUP BY GROUPING SETS((col1, col2), (col1, col3))`

A `ROLLUP` grouping creates aggregate subtotals that roll up from the most detailed level to a grand total, following a list of grouping columns (or expressions). `ROLLUP` takes an ordered list of grouping columns, calculates the standard aggregate values specified in the `GROUP BY` clause, then creates progressively higher-level subtotals, moving from right to left through the list. Finally, it creates a grand total.

A `CUBE` grouping creates subtotals for all of the possible combinations of the given list of grouping columns (or expressions). In multidimensional analysis terms, `CUBE` generates all the subtotals that could be calculated for a data cube with the specified dimensions.

You can selectively specify the set of groups that you want to create using a `GROUPING SETS` expression. This allows precise specification across multiple dimensions without computing a whole `ROLLUP` or `CUBE`.

Refer to the *Greenplum Database Reference Guide* for details of these clauses.

**Window Functions**

Window functions apply an aggregation or ranking function over partitions of the result set—for example, `sum(population) over (partition by city)`. Window functions are powerful and, because they do all of the work in the database, they have performance advantages over front-end tools that produce similar results by retrieving detail rows from the database and reprocessing them.

- The `row_number()` window function produces row numbers for the rows in a partition, for example, `row_number() over (order by id)`.
- When a query plan indicates that a table is scanned in more than one operation, you may be able to use window functions to reduce the number of scans.
- It is often possible to eliminate self joins by using window functions.
High Availability

Greenplum Database supports highly available, fault-tolerant database services when you enable and properly configure Greenplum high availability features. To guarantee a required level of service, each component must have a standby ready to take its place if it should fail.

**Disk Storage**

With the Greenplum Database "shared-nothing" MPP architecture, the master host and segment hosts each have their own dedicated memory and disk storage, and each master or segment instance has its own independent data directory. For both reliability and high performance, Pivotal recommends a hardware RAID storage solution with from 8 to 24 disks. A larger number of disks improves I/O throughput when using RAID 5 (or 6) because striping increases parallel disk I/O. The RAID controller can continue to function with a failed disk because it saves parity data on each disk in a way that it can reconstruct the data on any failed member of the array. If a hot spare is configured (or an operator replaces the failed disk with a new one) the controller rebuilds the failed disk automatically.

RAID 1 exactly mirrors disks, so if a disk fails, a replacement is immediately available with performance equivalent to that before the failure. With RAID 5 each I/O for data on the failed array member must be reconstructed from data on the remaining active drives until the replacement disk is rebuilt, so there is a temporary performance degradation. If the Greenplum master and segments are mirrored, you can switch any affected Greenplum instances to their mirrors during the rebuild to maintain acceptable performance.

A RAID disk array can still be a single point of failure, for example, if the entire RAID volume fails. At the hardware level, you can protect against a disk array failure by mirroring the array, using either host operating system mirroring or RAID controller mirroring, if supported.

It is important to regularly monitor available disk space on each segment host. Query the `gp_disk_free` external table in the `gptoolkit` schema to view disk space available on the segments. This view runs the Linux `df` command. Be sure to check that there is sufficient disk space before performing operations that consume large amounts of disk, such as copying a large table.

See `gp_toolkit.gp_disk_free` in the *Greenplum Database Reference Guide*.

**Best Practices**

- Use a hardware RAID storage solution with 8 to 24 disks.
- Use RAID 1, 5, or 6 so that the disk array can tolerate a failed disk.
- Configure a hot spare in the disk array to allow rebuild to begin automatically when disk failure is detected.
- Protect against failure of the entire disk array and degradation during rebuilds by mirroring the RAID volume.
- Monitor disk utilization regularly and add additional space when needed.
- Monitor segment skew to ensure that data is distributed evenly and storage is consumed evenly at all segments.

**Master Mirroring**

The Greenplum Database master instance is clients' single point of access to the system. The master instance stores the global system catalog, the set of system tables that store metadata about the database instance, but no user data. If an unmirrored master instance fails or becomes inaccessible, the Greenplum instance is effectively off-line, since the entry point to the system has been lost. For this reason, a standby master must be ready to take over if the primary master fails.

Master mirroring uses two processes, a sender on the active master host and a receiver on the mirror host, to synchronize the mirror with the master. As changes are applied to the master system catalogs,
the active master streams its write-ahead log (WAL) to the mirror so that each transaction applied on the
master is applied on the mirror.

The mirror is a warm standby. If the primary master fails, switching to the standby requires an
administrative user to run the `gpactivatestandby` utility on the standby host so that it begins to
accept client connections. Clients must reconnect to the new master and will lose any work that was not
committed when the primary failed.

See "Enabling High Availability Features" in the *Greenplum Database Administrator Guide* for more
information.

**Best Practices**

- Set up a standby master instance—a mirror—to take over if the primary master fails.
- The standby can be on the same host or on a different host, but it is best practice to place it on a
different host from the primary master to protect against host failure.
- Plan how to switch clients to the new master instance when a failure occurs, for example, by updating
the master address in DNS.
- Set up monitoring to send notifications in a system monitoring application or by email when the primary
fails.

**Segment Mirroring**

Greenplum Database segment instances each store and manage a portion of the database data, with
coordination from the master instance. If any unmirrored segment fails, the database may have to be
shutdown and recovered, and transactions occurring after the most recent backup could be lost. Mirroring
segments is, therefore, an essential element of a high availability solution.

A segment mirror is a hot standby for a primary segment. Greenplum Database detects when a segment
is unavailable and automatically activates the mirror. During normal operation, when the primary segment
instance is active, data is replicated from the primary to the mirror in two ways:

- The transaction commit log is replicated from the primary to the mirror before the transaction is
committed. This ensures that if the mirror is activated, the changes made by the last successful
transaction at the primary are present at the mirror. When the mirror is activated, transactions in the log
are applied to tables in the mirror.
- Second, segment mirroring uses physical file replication to update heap tables. Greenplum Server
stores table data on disk as fixed-size blocks packed with tuples. To optimize disk I/O, blocks are
cached in memory until the cache fills and some blocks must be evicted to make room for newly
updated blocks. When a block is evicted from the cache it is written to disk and replicated over the
network to the mirror. Because of the caching mechanism, table updates at the mirror can lag behind
the primary. However, because the transaction log is also replicated, the mirror remains consistent
with the primary. If the mirror is activated, the activation process updates the tables with any unapplied
changes in the transaction commit log.

When the acting primary is unable to access its mirror, replication stops and state of the primary changes
to “Change Tracking.” The primary saves changes that have not been replicated to the mirror in a system
table to be replicated to the mirror when it is back on-line.

The master automatically detects segment failures and activates the mirror. Transactions in progress at the
time of failure are restarted using the new primary. Depending on how mirrors are deployed on the hosts,
the database system may be unbalanced until the original primary segment is recovered. For example, if
each segment host has four primary segments and four mirror segments, and a mirror is activated on one
host, that host will have five active primary segments. Queries are not complete until the last segment has
finished its work, so performance can be degraded until the balance is restored by recovering the original
primary.

Administrators perform the recovery while Greenplum Database is up and running by running the
`gprecoverseg` utility. This utility locates the failed segments, verifies they are valid, and compares the
Greenplum Database Best Practices

(transactional state with the currently active segment to determine changes made while the segment was offline. `gprecoverseg` synchronizes the changed database files with the active segment and brings the segment back online.

It is important to reserve enough memory and CPU resources on segment hosts to allow for increased activity from mirrors that assume the primary role during a failure. The formulas provided in Configuring Memory for Greenplum Database for configuring segment host memory include a factor for the maximum number of primary hosts on any one segment during a failure. The arrangement of mirrors on the segment hosts affects this factor and how the system will respond during a failure. See Segment Mirroring Configuration for a discussion of segment mirroring options.

**Best Practices**

- Set up mirrors for all segments.
- Locate primary segments and their mirrors on different hosts to protect against host failure.
- Mirrors can be on a separate set of hosts or co-located on hosts with primary segments.
- Set up monitoring to send notifications in a system monitoring application or by email when a primary segment fails.
- Recover failed segments promptly, using the `gprecoverseg` utility, to restore redundancy and return the system to optimal balance.

**Dual Clusters**

For some use cases, an additional level of redundancy can be provided by maintaining two Greenplum Database clusters that store the same data. The decision to implement dual clusters should be made with business requirements in mind.

There are two recommended methods for keeping the data synchronized in a dual cluster configuration. The first method is called Dual ETL. ETL (extract, transform, and load) is the common data warehousing process of cleansing, transforming, validating, and loading data into a data warehouse. With Dual ETL, the ETL processes are performed twice, in parallel on each cluster, and validated each time. Dual ETL provides for a complete standby cluster with the same data. It also provides the capability to query the data on both clusters, doubling the processing throughput. The application can take advantage of both clusters as needed and also ensure that the ETL is successful and validated on both sides.

The second mechanism for maintaining dual clusters is backup and restore. The data is backed up on the primary cluster, then the backup is replicated to and restored on the second cluster. The backup and restore mechanism has higher latency than Dual ETL, but requires less application logic to be developed. Backup and restore is ideal for use cases where data modifications and ETL are done daily or less frequently.

**Best Practices**

- Consider a Dual Cluster configuration to provide an additional level of redundancy and additional query processing throughput.

**Backup and Restore**

Backups are recommended for Greenplum Database databases unless the data in the database can be easily and cleanly regenerated from source data. Backups protect from operational, software, or hardware errors.

The `gpcrondump` utility makes backups in parallel across the segments, so that backups scale as the cluster grows in hardware size.

Incremental backups can significantly reduce backup sizes in some cases. An incremental backup saves all heap tables, but only the append-optimized and column-oriented partitions that have changed since the previous backup. When the database has large fact tables with many partitions and modifications are
confined to one or a few partitions in a time period, incremental backups can save a large amount of disk space. Conversely, a database with large, unpartitioned fact tables is a poor application for incremental backups.

A backup strategy must consider where the backups will be written and where they will be stored. Backups can be taken to the local cluster disks, but they should not be stored there permanently. If the database and its backup are on the same storage, they can be lost simultaneously. The backup also occupies space that could be used for database storage or operations. After performing a local backup, the files should be copied to a safe, off-cluster location.

An alternative is to back up directly to an NFS mount. If each host in the cluster has an NFS mount, the backups can be written directly to NFS storage. A scale-out NFS solution is recommended to ensure that backups do not bottleneck on the IO throughput of the NFS device. Dell EMC Isilon is an example of this type of solution and can scale alongside the Greenplum cluster.

Finally, through native API integration, Greenplum Database can stream backups directly to Dell EMC Data Domain or Veritas NetBackup enterprise backup platforms.

**Best Practices**
- Back up Greenplum databases regularly unless the data is easily restored from sources.
- Use the `gpcrondump -s, -S, -t, or -T` options to specify only the schema and tables that need to be backed up. See the `gpcrondump` reference in the *Greenplum Database Utility Reference Guide* for more information.
- Back up one schema at a time to reduce the length of time the `pg_class` table is locked. At the start of the backup, `gpcrondump` places an EXCLUSIVE lock on the `pg_class` table, which prevents creating, altering, or dropping tables. This lock is held while `gpcrondump` computes the set of tables to back up and then places SHARED ACCESS locks on the tables. Once the tables are locked and backup processes started on the segments, the EXCLUSIVE lock on `pg_class` is released. In a database with many objects the duration of the EXCLUSIVE lock may be disruptive. Backing up one schema at a time can reduce the duration of the EXCLUSIVE lock. Backups with fewer tables are also more efficient for selectively restoring schemas and tables, since `gpdbrestore` does not have to search through the entire database. See *Backup Process and Locks* for more information.
- Use incremental backups when heap tables are relatively small and few append-optimized or column-oriented partitions are modified between backups.
- If backups are saved to local cluster storage, move the files to a safe, off-cluster location when the backup is complete. Backup files and database files that reside on the same storage can be lost simultaneously.
- If your operating system supports direct I/O, set the `gp_backup_directIO` configuration parameter to reduce CPU usage during backups. See *Using Direct I/O* for more information.
- If backups are saved to NFS mounts, use a scale-out NFS solution such as Dell EMC Isilon to prevent IO bottlenecks.
- Consider using Pivotal Greenplum Database integration to stream backups to the Dell EMC Data Domain or Veritas NetBackup enterprise backup platforms.

**Detecting Failed Master and Segment Instances**

Recovering from system failures requires intervention from a system administrator, even when the system detects a failure and activates a standby for the failed component. In each case, the failed component must be replaced or recovered to restore full redundancy. Until the failed component is recovered, the active component lacks a standby, and the system may not be executing optimally. For these reasons, it is important to perform recovery operations promptly. Constant system monitoring and automated fault notifications through SNMP and email ensure that administrators are aware of failures that demand their attention.

The Greenplum Database server `ftsprobe` subprocess handles fault detection. `ftsprobe` connects to and scans all segments and database processes at intervals that you can configure with the
gp_fts_probe_interval configuration parameter. If ftsprobe cannot connect to a segment, it marks the segment “down” in the Greenplum Database system catalog. The segment remains down until an administrator runs the gprecoverseg recovery utility.

You can configure a Greenplum Database system to trigger SNMP (Simple Network Management Protocol) alerts or send email notifications to system administrators if certain database events occur. See “Using SNMP with a Greenplum Database System” in the Pivotal Greenplum Database Administrator Guide for instructions to set up SNMP.

Best Practices

- Run the gpstate utility to see the overall state of the Greenplum system.
- Configure Greenplum Database to send SNMP notifications to your network monitor.
- Set up email notification in the SMASTER_DATA_DIRECTORY/postgresql.conf configuration file so that the Greenplum system can email administrators when a critical issue is detected.

Additional Information

Greenplum Database Administrator Guide:

- Monitoring a Greenplum System
- Recovering a Failed Segment
- Using SNMP with a Greenplum System
- Enabling Email Notifications

Greenplum Database Utility Guide:

- gpstate—view state of the Greenplum system
- gprecoverseg—recover a failed segment
- gpactivate_standby—make the standby master the active master

RDBMS MIB Specification

Segment Mirroring Configuration

Segment mirroring allows database queries to fail over to a backup segment if the primary segment fails or becomes unavailable. Pivotal requires mirroring for supported production Greenplum Database systems.

A primary segment and its mirror must be on different hosts to ensure high availability. Each host in a Greenplum Database system has the same number of primary segments and mirror segments. Multihomed hosts should have the same numbers of primary and mirror segments on each interface. This ensures that segment hosts and network resources are equally loaded when all primary segments are operational and brings the most resources to bear on query processing.

When a segment becomes unavailable, its mirror segment on another host becomes the active primary and processing continues. The additional load on the host creates skew and degrades performance, but should allow the system to continue. A database query is not complete until all segments return results, so a single host with an additional active primary segment has the same effect as adding an additional primary segment to every host in the cluster.

The least amount of performance degradation in a failover scenario occurs when no host has more than one mirror assuming the primary role. If multiple segments or hosts fail, the amount of degradation is determined by the host or hosts with the largest number of mirrors assuming the primary role. Spreading a host’s mirrors across the remaining hosts minimizes degradation when any single host fails.

It is important, too, to consider the cluster’s tolerance for multiple host failures and how to maintain a mirror configuration when expanding the cluster by adding hosts. There is no mirror configuration that is ideal for every situation.

You can allow Greenplum Database to arrange mirrors on the hosts in the cluster using one of two standard configurations, or you can design your own mirroring configuration.
The two standard mirroring arrangements are group mirroring and spread mirroring:

- **Group mirroring** — Each host mirrors another host's primary segments. This is the default for `gpinit` and `gpaddmirrors`.
- **Spread mirroring** — Mirrors are spread across the available hosts. This requires that the number of hosts in the cluster is greater than the number of segments per host.

You can design a custom mirroring configuration and use the Greenplum `gpaddmirrors` or `gpmovemirrors` utilities to set up the configuration.

*Block mirroring* is a custom mirror configuration that divides hosts in the cluster into equally sized blocks and distributes mirrors evenly to hosts within the block. If a primary segment fails, its mirror on another host within the same block becomes the active primary. If a segment host fails, mirror segments on each of the other hosts in the block become active.

The following sections compare the group, spread, and block mirroring configurations.

**Group Mirroring**

Group mirroring is easiest to set up and is the default Greenplum mirroring configuration. It is least expensive to expand, since it can be done by adding as few as two hosts. There is no need to move mirrors after expansion to maintain a consistent mirror configuration.

The following diagram shows a group mirroring configuration with eight primary segments on four hosts.
Unless both the primary and mirror of the same segment instance fail, up to half of your hosts can fail and the cluster will continue to run as long as resources (CPU, memory, and IO) are sufficient to meet the needs.

Any host failure will degrade performance by half or more because the host with the mirrors will have twice the number of active primaries. If your resource utilization is normally greater than 50%, you will have to adjust your workload until the failed host is recovered or replaced. If you normally run at less than 50% resource utilization the cluster can continue to operate at a degraded level of performance until the failure is corrected.

**Spread Mirroring**

With spread mirroring, mirrors for each host's primary segments are spread across as many hosts as there are segments per host. Spread mirroring is easy to set up when the cluster is initialized, but requires that the cluster have at least one more host than there are segments per host.

The following diagram shows the spread mirroring configuration for a cluster with three primaries on four hosts.
Expanding a cluster with spread mirroring requires more planning and may take more time. You must either add a set of hosts equal to the number of primaries per host plus one, or you can add two nodes in a group mirroring configuration and, when the expansion is complete, move mirrors to recreate the spread mirror configuration.

Spread mirroring has the least performance impact for a single failed host because each host's mirrors are spread across the maximum number of hosts. Load is increased by 1/Nth, where N is the number of primaries per host. Spread mirroring is, however, the most likely configuration to suffer catastrophic failure if two or more hosts fail simultaneously.

**Block Mirroring**

With block mirroring, nodes are divided into blocks, for example a block of four or eight hosts, and the mirrors for segments on each host are placed on other hosts within the block. Depending on the number of hosts in the block and the number of primary segments per host, each host maintains more than one mirror for each other host's segments.

The following diagram shows a single block mirroring configuration for a block of four hosts, each with eight primary segments:
If there are eight hosts, an additional four-host block is added with the mirrors for primary segments 32 through 63 set up in the same pattern.

A cluster with block mirroring is easy to expand because each block is a self-contained primary mirror group. The cluster is expanded by adding one or more blocks. There is no need to move mirrors after expansion to maintain a consistent mirror setup. This configuration is able to survive multiple host failures as long as the failed hosts are in different blocks.

Because each host in a block has multiple mirror instances for each other host in the block, block mirroring has a higher performance impact for host failures than spread mirroring, but a lower impact than group mirroring. The expected performance impact varies by block size and primary segments per node. As with group mirroring, if the resources are available, performance will be negatively impacted but the cluster will remain available. If resources are insufficient to accommodate the added load you must reduce the workload until the failed node is replaced.

**Implementing Block Mirroring**

Block mirroring is not one of the automatic options Greenplum Database offers when you set up or expand a cluster. To use it, you must create your own configuration.
For a new Greenplum system, you can initialize the cluster without mirrors, and then run `gpaddmirrors -i mirror_config_file` with a custom mirror configuration file to create the mirrors for each block. You must create the file system locations for the mirror segments before you run `gpaddmirrors`. See the `gpaddmirrors` reference page in the *Greenplum Database Management Utility Guide* for details.

If you expand a system that has block mirroring or you want to implement block mirroring at the same time you expand a cluster, it is recommended that you complete the expansion first, using the default grouping mirror configuration, and then use the `gpmovemirrors` utility to move mirrors into the block configuration.

To implement block mirroring with an existing system that has a different mirroring scheme, you must first determine the desired location for each mirror according to your block configuration, and then determine which of the existing mirrors must be relocated. Follow these steps:

1. Run the following query to find the current locations of the primary and mirror segments:

   ```sql
   SELECT dbid, content, address, port, replication_port, fselocation as datadir
   FROM gp_segment_configuration, pg_filespace_entry
   WHERE dbid=fsedbid AND content > -1
   ORDER BY dbid;
   ```

   The `gp_segment_configuration`, `pg_filespace`, and `pg_filespace_entry` system catalog tables contain the current segment configuration.

2. Create a list with the current mirror location and the desired block mirroring location, then remove any mirrors from the list that are already on the correct host.

3. Create an input file for the `gpmovemirrors` utility with an entry for each mirror that must be moved.

   The `gpmovemirrors` input file has the following format:

   ```text
   filespaceOrder=[filespace1_fsname[:filespace2_fsname:...]
   old_address:port:fselocation
   new_address:port:replication_port:fselocation[:fselocation:...]
   ```

   The first non-comment line must be a line beginning with `filespaceOrder=`. Do not include the default `pg_system` filesystem in the list of filesystems. Leave the list empty if you are using only the `pg_system` filesystem.

   The following example `gpmovemirrors` input file specifies three mirror segments to move.

   ```text
   filespaceOrder=
sdw2:50001:/data2/mirror/gpseg1 sdw3:50000:51000:/data/mirror/gpseg1
dsdw2:50001:/data2/mirror/gpseg2 sdw4:50000:51000:/data/mirror/gpseg2
dsdw3:50001:/data2/mirror/gpseg3 sdw1:50000:51000:/data/mirror/gpseg3
   ```

4. Run `gpmovemirrors` with a command like the following:

   ```bash
   gpmovemirrors -i mirror_config_file
   ```

   The `gpmovemirrors` utility validates the input file, calls `gp_recoverseg` to relocate each specified mirror, and removes the original mirror. It creates a backout configuration file which can be used as input to `gpmovemirrors` to undo the changes that were made. The backout file has the same name as the input file, with the suffix `_backout_timestamp` added.

   See the *Greenplum Database Management Utility Reference* for complete information about the `gpmovemirrors` utility.
Chapter 13

Greenplum Database Utility Guide

Reference information for command-line utilities, client programs, and Oracle compatibility functions.
Management Utility Reference

Describes the command-line management utilities provided with Greenplum Database.

Greenplum Database uses the standard PostgreSQL client and server programs and provides additional management utilities for administering a distributed Greenplum Database DBMS. Greenplum Database management utilities reside in $GPHOME/bin.

**Note:** When referencing IPv6 addresses in gpfdist URLs or when using numeric IP addresses instead of hostnames in any management utility, always enclose the IP address in brackets. For command prompt use, the best practice is to escape any brackets or put them inside quotation marks. For example, use either:

```
gpdbrestore -R \[2620:0:170:610::11\]
gpdbrestore -R '[2620:0:170:610::11]'
```

The following are the Greenplum Database management utilities.

<table>
<thead>
<tr>
<th>Management Utility Reference</th>
<th>Management Utility Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>analyzedb</td>
<td>gpmfr</td>
</tr>
<tr>
<td>gpactivatestandby</td>
<td>gpperfmon_install</td>
</tr>
<tr>
<td>gpaddmirrors</td>
<td>gpkg</td>
</tr>
<tr>
<td>gpcheck</td>
<td>gprebuildsystem (deprecated)</td>
</tr>
<tr>
<td>gpcheckcat</td>
<td>gprecoverseg</td>
</tr>
<tr>
<td>gpchecknet (deprecated)</td>
<td>gpreload</td>
</tr>
<tr>
<td>gpcheckos (deprecated)</td>
<td>gp_restore (deprecated)</td>
</tr>
<tr>
<td>gpcheckperf</td>
<td>gpsizecalc (deprecated)</td>
</tr>
<tr>
<td>gpconfig</td>
<td>gpscp</td>
</tr>
<tr>
<td>gpcrondump</td>
<td>gpskew (deprecated)</td>
</tr>
<tr>
<td>gpdbrestore</td>
<td>gpseginstall</td>
</tr>
<tr>
<td>gpdeleteystem</td>
<td>gpssh</td>
</tr>
<tr>
<td>gpdetective (deprecated)</td>
<td>gpssh-exkeys</td>
</tr>
<tr>
<td>gp_dump (deprecated)</td>
<td>gpstart</td>
</tr>
<tr>
<td>gpexpand</td>
<td>gpstate</td>
</tr>
<tr>
<td>gpdist</td>
<td>gpstop</td>
</tr>
<tr>
<td>gpfilespace</td>
<td>gpsy1</td>
</tr>
<tr>
<td>gpinitstandby</td>
<td>gptransfer</td>
</tr>
<tr>
<td>gpinitssystem</td>
<td>pgbouncer</td>
</tr>
<tr>
<td>gpload</td>
<td></td>
</tr>
<tr>
<td>gplogfilter</td>
<td></td>
</tr>
<tr>
<td>gpmapreduce</td>
<td></td>
</tr>
</tbody>
</table>
## Backend Server Programs

The following standard PostgreSQL server management programs are provided with Greenplum Database and reside in `$GPHOME/bin`. They are modified to handle the parallelism and distribution of a Greenplum Database system. You access these programs only through the Greenplum Database management tools and utilities.

### Table 68: Greenplum Database Backend Server Programs

<table>
<thead>
<tr>
<th>Program Name</th>
<th>Description</th>
<th>Use Instead</th>
</tr>
</thead>
<tbody>
<tr>
<td>initdb</td>
<td>This program is called by <code>gpinitsystem</code> when initializing a Greenplum Database array. It is used internally to create the individual segment instances and the master instance.</td>
<td><code>gpinitsystem</code></td>
</tr>
<tr>
<td>ipcclean</td>
<td>Not used in Greenplum Database</td>
<td>N/A</td>
</tr>
<tr>
<td>gpsyncmaster</td>
<td>This is the Greenplum program that starts the <code>gpsyncagent</code> process on the standby master host. Administrators do not call this program directly, but do so through the management scripts that initialize and/or activate a standby master for a Greenplum Database system. This process is responsible for keeping the standby master up to date with the primary master via a transaction log replication process.</td>
<td><code>gpinitstandby</code>, <code>gpactivatestandby</code></td>
</tr>
<tr>
<td>pg_controldata</td>
<td>Not used in Greenplum Database</td>
<td><code>gpstate</code></td>
</tr>
<tr>
<td>pg_ctl</td>
<td>This program is called by <code>gpstart</code> and <code>gpstop</code> when starting or stopping a Greenplum Database array. It is used internally to stop and start the individual segment instances and the master instance in parallel and with the correct options.</td>
<td><code>gpstart</code>, <code>gpstop</code></td>
</tr>
<tr>
<td>pg_resetxlog</td>
<td><strong>Warning:</strong> This program might cause data loss or cause data to become unavailable. If this program is used, the Pivotal Greenplum Database cluster is not supported. The cluster must be reinitialized and restored by the customer.</td>
<td>N/A</td>
</tr>
<tr>
<td>Program Name</td>
<td>Description</td>
<td>Use Instead</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>postgres</td>
<td>The <code>postgres</code> executable is the actual PostgreSQL server process that processes queries.</td>
<td>The main <code>postgres</code> process (postmaster) creates other <code>postgres</code> subprocesses and <code>postgres</code> session as needed to handle client connections.</td>
</tr>
<tr>
<td>postmaster</td>
<td><code>postmaster</code> starts the <code>postgres</code> database server listener process that accepts client connections. In Greenplum Database, a <code>postgres</code> database listener process runs on the Greenplum master Instance and on each Segment Instance.</td>
<td>In Greenplum Database, you use <code>gpstart</code> and <code>gpstop</code> to start all postmasters (<code>postgres</code> processes) in the system at once in the correct order and with the correct options.</td>
</tr>
</tbody>
</table>

**analyzedb**

A utility that performs `ANALYZE` operations on tables incrementally and concurrently. For append optimized tables, `analyzedb` updates statistics only if the statistics are not current.

**Synopsis**

```
analyzedb -d dbname
   { -s schema | -t schema.table
     [ -i col1[], col2, ... ] | -x col1[, col2, ... ] } |
     { -f | --file } config-file }
   [-l | --list ]
   [-p parallel-level ]
   [ --full ]
   [ --skip_root_stats ]
   [-v | --verbose ]
   [ --debug ]
   [-a ]
analyzedb { --clean_last | --clean_all }  
analyzedb --version  
analyzedb { -? | -h | --help }  
```

**Description**

The `analyzedb` utility updates statistics on table data for the specified tables in a Greenplum database incrementally and concurrently.

While performing `ANALYZE` operations, `analyzedb` creates a snapshot of the table metadata and stores it on disk on the master host. An `ANALYZE` operation is performed only if the table has been modified. If a table or partition has not been modified since the last time it was analyzed, `analyzedb` automatically skips the table or partition because it already contains up-to-date statistics.

- For append optimized tables, `analyzedb` updates statistics incrementally, if the statistics are not current. For example, if table data is changed after statistics were collected for the table. If there are no statistics for the table, statistics are collected.
- For heap tables, statistics are always updated.

Specify the `--full` option to update append-optimized table statistics even if the table statistics are current.
By default, `analyzedb` creates a maximum of 5 concurrent sessions to analyze tables in parallel. For each session, `analyzedb` issues an `ANALYZE` command to the database and specifies different table names. The `-p` option controls the maximum number of concurrent sessions.

### Partitioned Append-Optimized Tables

For a partitioned, append-optimized table, `analyzedb` checks the partitioned table root partition and leaf partitions. If needed, the utility updates statistics for non-current partitions and the root partition.

The root partition statistics is required by GPORCA. By default, the `analyzedb` utility collects statistics on the root partition of a partitioned table if the statistics do not exist. If any of the leaf partitions have stale statistics, `analyzedb` also refreshes the root partition statistics. The cost of refreshing the root level statistics is comparable to analyzing one leaf partition. You can specify the option `--skip_root_stats` to disable collection of statistics on the root partition of a partitioned table.

### Notes

The `analyzedb` utility updates append optimized table statistics if the table has been modified by DML or DDL commands, including `INSERT`, `DELETE`, `UPDATE`, `CREATE TABLE`, `ALTER TABLE` and `TRUNCATE`. The utility determines if a table has been modified by comparing catalog metadata of tables with the snapshot of metadata taken during a previous `analyzedb` operation. The snapshots of table metadata are stored as state files in the directory `db_analyze` in the Greenplum Database master data directory. You can specify the `--clean_last` or `--clean_all` option to remove state files generated by `analyzedb`.

If you do not specify a table, set of tables, or schema, the `analyzedb` utility collects the statistics as needed on all system catalog tables and user-defined tables in the database.

External tables are not affected by `analyzedb`.

Table names that contain spaces are not supported.

Running the `ANALYZE` command on a table, not using the `analyzedb` utility, does not update the table metadata that the `analyzedb` utility uses to determine whether table statistics are up to date.

### Options

`--clean_last`
Remove the state files generated by last `analyzedb` operation. All other options except `-d` are ignored.

`--clean_all`
Remove all the state files generated by `analyzedb`. All other options except `-d` are ignored.

`-d dbname`
Specifies the name of the database that contains the tables to be analyzed. If this option is not specified, the database name is read from the environment variable `PGDATABASE`. If `PGDATABASE` is not set, the user name specified for the connection is used.

`--debug`
If specified, sets the logging level to debug. During command execution, debug level information is written to the log file and to the command line. The information includes the commands executed by the utility and the duration of each `ANALYZE` operation.

`-f config-file | --file config-file`
Text file that contains a list of tables to be analyzed. A relative file path from current directory can be specified.

The file lists one table per line. Table names must be qualified with a schema name. Optionally, a list of columns can be specified using the `-i` or `-x`. No other options are allowed in the file. Other options such as `--full` must be specified on the command line.
Only one of the options can be used to specify the files to be analyzed: -f or --file, -t, or -s.

When performing ANALYZE operations on multiple tables, analyzedb creates concurrent sessions to analyze tables in parallel. The -p option controls the maximum number of concurrent sessions.

In the following example, the first line performs an ANALYZE operation on the table public.nation, the second line performs an ANALYZE operation only on the columns l_shipdate and l_receiptdate in the table public.lineitem.

```
public.nation
public.lineitem -i l_shipdate, l_receiptdate
```

--full

Perform an ANALYZE operation on all the specified tables. The operation is performed even if the statistics are up to date.

-i col1, col2, ...

Optional. Must be specified with the -t option. For the table specified with the -t option, collect statistics only for the specified columns.

Only -i, or -x can be specified. Both options cannot be specified.

-l | --list

Lists the tables that would have been analyzed with the specified options. The ANALYZE operations are not performed.

-p parallel-level

The number of tables that are analyzed in parallel. parallel level can be an integer between 1 and 10, inclusive. Default value is 5.

--skip_root_stats

Skip refreshing root partition statistics if any of the leaf partitions that are also analyzed require updating.

Do not specify this option if you use GPORCA to execute queries against partitioned tables (the default).

-s schema

Specify a schema to analyze. All tables in the schema will be analyzed. Only a single schema name can be specified on the command line.

Only one of the options can be used to specify the files to be analyzed: -f or --file, -t, or -s.

-t schema.table

Collect statistics only on schema.table. The table name must be qualified with a schema name. Only a single table name can be specified on the command line. You can specify the -f option to specify multiple tables in a file or the -s option to specify all the tables in a schema.

Only one of these options can be used to specify the files to be analyzed: -f or --file, -t, or -s.

-x col1, col2, ...

Optional. Must be specified with the -t option. For the table specified with the -t option, exclude statistics collection for the specified columns. Statistics are collected only on the columns that are not listed.

Only -i, or -x can be specified. Both options cannot be specified.

-a
Quiet mode. Do not prompt for user confirmation.

-h | -? | --help
Displays the online help.

-v | --verbose
If specified, sets the logging level to verbose to write additional information the log file and to the command line during command execution. The information includes a list of all the tables to be analyzed (including child leaf partitions of partitioned tables). Output also includes the duration of each ANALYZE operation.

--version
Displays the version of this utility.

Examples
An example that collects statistics only on a set of table columns. In the database mytest, collect statistics on the columns shipdate and receiptdate in the table public.orders:

```
analyzedb -d mytest -t public.orders -i shipdate, receiptdate
```

An example that collects statistics on a table and exclude a set of columns. In the database mytest, collect statistics on the table public.foo, and do not collect statistics on the columns bar and test2.

```
analyzedb -d mytest -t public.foo -x bar, test2
```

An example that specifies a file that contains a list of tables. This command collect statistics on the tables listed in the file analyze-tables in the database named mytest.

```
analyzedb -d mytest -f analyze-tables
```

If you do not specify a table, set of tables, or schema, the analyzedb utility collects the statistics as needed on all catalog tables and user-defined tables in the specified database. This command refreshes table statistics on the system catalog tables and user-defined tables in the database mytest.

```
analyzedb -d mytest
```

You can create a PL/Python function to run the analyzedb utility as a Greenplum Database function. This example CREATE FUNCTION command creates a user defined PL/Python function that runs the analyzedb utility and displays output on the command line. Specify analyzedb options as the function parameter.

```
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;
```
When this `SELECT` command is run by the `gpadmin` user, the `analyzedb` utility performs an analyze operation on the table `public.mytable` that is in the database `mytest`.

```sql
SELECT analyzedb('-d mytest -t public.mytable') ;
```

**Note:** To create a PL/Python function, the PL/Python procedural language must be registered as a language in the database. For example, this `CREATE LANGUAGE` command run as `gpadmin` registers PL/Python as an untrusted language:

```sql
CREATE LANGUAGE plpythonu;
```

**See Also**

`ANALYZE` in the *Greenplum Database Reference Guide*

---

**gpactivatestandby**

Activates a standby master host and makes it the active master for the Greenplum Database system.

**Synopsis**

```bash
gpactivatestandby [-d standby_master_datadir] [-f] [-a] [-q] [-l logfile_directory]
gpactivatestandby -v
gpactivatestandby -? | -h | --help
```

**Description**

The `gpactivatestandby` utility activates a backup, standby master host and brings it into operation as the active master instance for a Greenplum Database system. The activated standby master effectively becomes the Greenplum Database master, accepting client connections on the master port.

When you initialize a standby master, the default is to use the same port as the active master. For information about the master port for the standby master, see `gpinitstandby`.

You must run this utility from the master host you are activating, not the failed master host you are disabling. Running this utility assumes you have a standby master host configured for the system (see `gpinitstandby`).

The utility will perform the following steps:

- Stops the synchronization process (`walreceiver`) on the standby master
- Updates the system catalog tables of the standby master using the logs
- Activates the standby master to be the new active master for the system
- Restarts the Greenplum Database system with the new master host

A backup, standby Greenplum master host serves as a 'warm standby' in the event of the primary Greenplum master host becoming non-operational. The standby master is kept up to date by transaction log replication processes (the `walsender` and `walreceiver`), which run on the primary master and standby master hosts and keep the data between the primary and standby master hosts synchronized.

If the primary master fails, the log replication process is shutdown, and the standby master can be activated in its place by using the `gpactivatestandby` utility. Upon activation of the standby master, the replicated logs are used to reconstruct the state of the Greenplum master host at the time of the last successfully committed transaction.
In order to use `gpactivatestandby` to activate a new primary master host, the master host that was previously serving as the primary master cannot be running. The utility checks for a `postmaster.pid` file in the data directory of the disabled master host, and if it finds it there, it will assume the old master host is still active. In some cases, you may need to remove the `postmaster.pid` file from the disabled master host data directory before running `gpactivatestandby` (for example, if the disabled master host process was terminated unexpectedly).

After activating a standby master, run `ANALYZE` to update the database query statistics. For example:

```bash
psql dbname -c 'ANALYZE;'
```

After you activate the standby master as the primary master, the Greenplum Database system no longer has a standby master configured. You might want to specify another host to be the new standby with the `gpinitstandby` utility.

**Options**

- `-a` **(do not prompt)**
  Do not prompt the user for confirmation.

- `-d **standby_master_datadir**`
  The absolute path of the data directory for the master host you are activating.

  If this option is not specified, `gpactivatestandby` uses the value of the `MASTER_DATA_DIRECTORY` environment variable setting on the master host you are activating. If this option is specified, it overrides any setting of `MASTER_DATA_DIRECTORY`.

  If a directory cannot be determined, the utility returns an error.

- `-f` **(force activation)**
  Use this option to force activation of the backup master host. Use this option only if you are sure that the standby and primary master hosts are consistent.

- `-l **logfile_directory**`
  The directory to write the log file. Defaults to `~/gpAdminLogs`.

- `-q` **(no screen output)**
  Run in quiet mode. Command output is not displayed on the screen, but is still written to the log file.

- `-v` **(show utility version)**
  Displays the version, status, last updated date, and check sum of this utility.

- `-?` | `-h` | `--help` **(help)**
  Displays the online help.

**Example**

Activate the standby master host and make it the active master instance for a Greenplum Database system (run from backup master host you are activating):

```bash
gpactivatestandby -d /gpdata
```

**See Also**

`gpinitsystem`, `gpinitstandby`

**gpaddmirrors**

Adds mirror segments to a Greenplum Database system that was initially configured without mirroring.
Synopsis

```
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 [-s] [-a] [-d master_data_directory]
             [-B parallel_processes] [-l logfile_directory] [-v]

gpaddmirrors -o output_sample_mirror_config [-m datadir_config_file]

gpaddmirrors -?

gpaddmirrors --version
```

Description

The `gpaddmirrors` utility configures mirror segment instances for an existing Greenplum Database system that was initially configured with primary segment instances only. The utility will create the mirror instances and begin the online replication process between the primary and mirror segment instances. Once all mirrors are synchronized with their primaries, your Greenplum Database system is fully data redundant.

**Important:** During the online replication process, Greenplum Database should be in a quiescent state, workloads and other queries should not be running.

By default, the utility will prompt you for the file system location(s) where it will create the mirror segment data directories. If you do not want to be prompted, you can pass in a file containing the file system locations using the `--m` option.

The mirror locations and ports must be different than your primary segment data locations and ports. If you have created additional filespaces, you will also be prompted for mirror locations for each of your filespaces.

The utility creates a unique data directory for each mirror segment instance in the specified location using the predefined naming convention. There must be the same number of file system locations declared for mirror segment instances as for primary segment instances. It is OK to specify the same directory name multiple times if you want your mirror data directories created in the same location, or you can enter a different data location for each mirror. Enter the absolute path. For example:

```
Enter mirror segment data directory location 1 of 2 > /gpdb/mir1
Enter mirror segment data directory location 2 of 2 > /gpdb/mir2
```

OR

```
Enter mirror segment data directory location 1 of 2 > /gpdb/m1
Enter mirror segment data directory location 2 of 2 > /gpdb/m2
```

Alternatively, you can run the `gpaddmirrors` utility and supply a detailed configuration file using the `--i` option. This is useful if you want your mirror segments on a completely different set of hosts than your primary segments. The format of the mirror configuration file is:

```
filespaceOrder=[filespace1_fsnname[:filespace2_fsnname:...]
mirror[content]=content:address:port:mir_replication_port:pri_replication_port:fselocation[:fselocation:...]
```

For example (if you do not have additional filespaces configured besides the default `pg_system` filespace):

```
filespaceOrder=
mirror0=0:sdw1-l:60000:61000:62000:/gpdata/mir1/gp0
```
The gp_segment_configuration, pg_filespace, and pg_filespace_entry system catalog tables can help you determine your current primary segment configuration so that you can plan your mirror segment configuration. For example, run the following query:

```sql
=# SELECT dbid, content, address as host_address, port, replication_port, fselocation as datadir FROM gp_segment_configuration, pg_filespace_entry WHERE dbid=fseadbid ORDER BY dbid;
```

If creating your mirrors on alternate mirror hosts, the new mirror segment hosts must be pre-installed with the Greenplum Database software and configured exactly the same as the existing primary segment hosts. You must make sure that the user who runs gpaddmirrors (the gpadmin user) has permissions to write to the data directory locations specified. You may want to create these directories on the segment hosts and chown them to the appropriate user before running gpaddmirrors.

**Options**

- `-a` *(do not prompt)*
  
  Run in quiet mode - do not prompt for information. Must supply a configuration file with either `-m` or `-i` if this option is used.

- `-B` *(parallel_processes)*
  
  The number of mirror setup processes to start in parallel. If not specified, the utility will start up to 10 parallel processes depending on how many mirror segment instances it needs to set up.

- `-d` *(master_data_directory)*
  
  The master data directory. If not specified, the value set for `$MASTER_DATA_DIRECTORY` will be used.

- `-i` *(mirror_config_file)*
  
  A configuration file containing one line for each mirror segment you want to create. You must have one mirror segment listed for each primary segment in the system. The format of this file is as follows (as per attributes in the gp_segment_configuration, pg_filespace, and pg_filespace_entry catalog tables):

  ```
  filespaceOrder=[filespace1 fsname[:filespace2 fsname:...]]
  mirror=[content]=content:address:port:mir_replication_port:pri_replication_port:fselocation[:fselocation:...]
  ```

  Note that you only need to specify a name for `filespaceOrder` if your system has multiple filesystems configured. If your system does not have additional filesystems configured besides the default `pg_system` filesystem, this file will only have one location (for the default data directory filesystem, `pg_system`). `pg_system` does not need to be listed in the `filespaceOrder` line. It will always be the first `fselocation` listed after `replication_port`.

- `-l` *(logfile_directory)*
  
  The directory to write the log file. Defaults to `~/gpAdminLogs`.

- `-m` *(datadir_config_file)*
  
  A configuration file containing a list of file system locations where the mirror data directories will be created. If not supplied, the utility prompts you for locations. Each line in the file specifies a mirror data directory location. For example:

  ```
  /gpdata/m1
  ```
If your system has additional filesystems configured in addition to the default `pg_system` filesystem, you must also list file system locations for each filesystem as follows:

```
/filespace1
/gpfs1/m1
/gpfs1/m2
/gpfs1/m3
/gpfs1/m4
```

```
filespace filespace1
/gpfs1/m1
/gpfs1/m2
/gpfs1/m3
```

**If you are not sure how to lay out the mirror configuration file used by the `--i` option, you can run `gpaddmirrors` with this option to generate a sample mirror configuration file based on your primary segment configuration. The utility will prompt you for your mirror segment data directory locations (unless you provide these in a file using `--m`). You can then edit this file to change the host names to alternate mirror hosts if necessary.**

**-o output_sample_mirror_config**

If you are not sure how to lay out the mirror configuration file used by the `--i` option, you can run `gpaddmirrors` with this option to generate a sample mirror configuration file based on your primary segment configuration. The utility will prompt you for your mirror segment data directory locations (unless you provide these in a file using `--m`). You can then edit this file to change the host names to alternate mirror hosts if necessary.

**-p port_offset**

Optional. This number is used to calculate the database ports and replication ports used for mirror segments. The default offset is 1000. Mirror port assignments are calculated as follows:

- primary port + offset = mirror database port
- primary port + (2 * offset) = mirror replication port
- primary port + (3 * offset) = primary replication port

For example, if a primary segment has port 50001, then its mirror will use a database port of 51001, a mirror replication port of 52001, and a primary replication port of 53001 by default.

**-s (spread mirrors)**

Spreads the mirror segments across the available hosts. The default is to group a set of mirror segments together on an alternate host from their primary segment set. Mirror spreading will place each mirror on a different host within the Greenplum Database array. Spreading is only allowed if there is a sufficient number of hosts in the array (number of hosts is greater than the number of segment instances per host).

**-v (verbose)**

Sets logging output to verbose.

**--version (show utility version)**

Displays the version of this utility.

**-? (help)**

Displays the online help.

### Examples

Add mirroring to an existing Greenplum Database system using the same set of hosts as your primary data. Calculate the mirror database and replication ports by adding 100 to the current primary segment port numbers:

```
$ gpaddmirrors -p 100
```
Add mirroring to an existing Greenplum Database system using a different set of hosts from your primary data:

```
$ gpaddmirrors -i mirror_config_file
```

Where `mirror_config_file` looks something like this (if you do not have additional filesystems configured besides the default `pg_system` filesystem):

```
filespaceOrder=
mirror0=0:sdw1-1:52001:53001:54001:/gpdata/mir1/gp0
mirror1=1:sdw1-2:52002:53002:54002:/gpdata/mir2/gp1
mirror2=2:sdw2-1:52001:53001:54001:/gpdata/mir1/gp2
```

Output a sample mirror configuration file to use with `gpaddmirrors -i`:

```
$ gpaddmirrors -o /home/gpadmin/sample_mirror_config
```

See Also

gpinitsystem, gpinitstandby, gpactivatestandby

gpcheck

Verifies and validates Greenplum Database platform settings.

Synopsis

```
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
```

Description

The `gpcheck` utility determines the platform on which you are running Greenplum Database and validates various platform-specific configuration settings. `gpcheck` can use a host file or a file previously created with the `--zipout` option to validate platform settings. At the end of a successful validation process, `GPCHECK_NORMAL` message displays. If `GPCHECK_ERROR` displays, one or more validation checks failed. You can use also `gpcheck` to gather and view platform settings on hosts without running validation checks.

You should run `gpcheck` as `root`. If you do not run `gpcheck` as `root`, the utility displays a warning message and will not be able to validate all configuration settings; Only some of these settings will be validated.

Options

`--config config_file`

The name of a configuration file to use instead of the default file `$GPHOME/etc/gpcheck.cnf` (or `~/gpconfigs/gpcheck_dca_config` on the Dell EMC Greenplum Data Computing Appliance). This file specifies the OS-specific checks to run.
{-f | --file} hostfile_gpcheck
   The name of a file that contains a list of hosts that gpcheck uses to validate platform-specific settings. This file should contain a single host name for all hosts in your Greenplum Database system (master, standby master, and segments). gpcheck uses SSH to connect to the hosts.

{--h | --host} host_ID
   Checks platform-specific settings on the host in your Greenplum Database system specified by host_ID. gpcheck uses SSH to connect to the host.

--local
   Checks platform-specific settings on the segment host where gpcheck is run. This option does not require SSH authentication.

-m master_host
   This option is deprecated and will be removed in a future release.

-s standby_master_host
   This option is deprecated and will be removed in a future release.

--stdout
   Display collected host information from gpcheck. No checks or validations are performed.

--zipout
   Save all collected data to a .zip file in the current working directory. gpcheck automatically creates the .zip file and names it gpcheck_timestamp.tar.gz. No checks or validations are performed.

--zipin gpcheck_zipfile
   Use this option to decompress and check a .zip file created with the --zipout option. gpcheck performs validation tasks against the file you specify in this option.

-? (help)
   Displays the online help.

--version
   Displays the version of this utility.

Examples
Verify and validate the Greenplum Database platform settings by entering a host file:

# gpcheck -f hostfile_gpcheck

Save Greenplum Database platform settings to a zip file:

# gpcheck -f hostfile_gpcheck --zipout

Verify and validate the Greenplum Database platform settings using a zip file created with the --zipout option:

# gpcheck --zipin gpcheck_timestamp.tar.gz

View collected Greenplum Database platform settings:

# gpcheck -f hostfile_gpcheck --stdout
See Also
gpssh, gpscp, gpcheckperf

gpcheckcat

The gpcheckcat utility tests Greenplum Database catalog tables for inconsistencies. The utility is in $GPHOME/bin/lib.

Synopsis

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

Description

The gpcheckcat utility runs multiple tests that check for database catalog inconsistencies. Some of the tests cannot be run concurrently with other workload statements or the results will not be usable. Restart the database in restricted mode when running gpcheckcat, otherwise gpcheckcat might report inconsistencies due to ongoing database operations rather than the actual number of inconsistencies. If you run gpcheckcat without stopping database activity, run it with -O option.

  Note: Any time you run the utility, it checks for and deletes orphaned, temporary database schemas (temporary schemas without a session ID) in the specified databases. The utility displays the results of the orphaned, temporary schema check on the command line and also logs the results.

Catalog inconsistencies are inconsistencies that occur between Greenplum Database system tables. In general, there are three types of inconsistencies:

- Inconsistencies in system tables at the segment level. For example, an inconsistency between a system table that contains table data and a system table that contains column data. As another, a system table that contains duplicates in a column that should to be unique.

- Inconsistencies between same system table across segments. For example, a system table is missing row on one segment, but other segments have this row. As another example, the values of specific row column data are different across segments, such as table owner or table access privileges.

- Persistent Table inconsistencies are inconsistencies in persistence object state and file system objects on a segment. For example, there are no running transactions, all transactions are complete, but there is object that is marked as creation incomplete in persistent tables. As another example, a file exists in database directory, but there is no corresponding object existing in the database system tables.
Options

- **A**
  
  Run `gpcheckcat` on all databases in the Greenplum Database installation.

- **B parallel_processes**
  
  The number of processes to run in parallel.

  The `gpcheckcat` utility attempts to determine the number of simultaneous processes (the batch size) to use. The utility assumes it can use a buffer with a minimum of 20MB for each process. The maximum number of parallel processes is the number of Greenplum Database segment instances. The utility displays the number of parallel processes that it uses when it starts checking the catalog.

  **Note:** The utility might run out of memory if the number of errors returned exceeds the buffer size. If an out of memory error occurs, you can lower the batch size with the `-B` option. For example, if the utility displays a batch size of 936 and runs out of memory, you can specify `-B 468` to run 468 processes in parallel.

- **C catalog_table**
  
  Run cross consistency, foreign key, and ACL tests for the specified catalog table.

- **g data_directory**
  
  Generate SQL scripts to fix catalog inconsistencies. The scripts are placed in `data_directory`.

- **l**
  
  List the `gpcheckcat` tests.

- **O**
  
  Run only the `gpcheckcat` tests that can be run in online (not restricted) mode.

- **p port**
  
  This option specifies the port that is used by the Greenplum Database.

- **P password**
  
  The password of the user connecting to Greenplum Database.

- **R test_name**
  
  Specify a test to run. Some tests can be run only when Greenplum Database is in restricted mode.

  These are the tests that can be performed:

  acl - Cross consistency check for access control privileges

  duplicate - Check for duplicate entries

  duplicate_persistent - Check for duplicate `gp_persistent_relation_node` entries

  foreign_key - Check foreign keys

  inconsistent - Cross consistency check for master segment inconsistency

  mirroring_matching - Checks if mirroring is consistent (either enabled or disabled) for the cluster and segments.

  missing_extraneous - Cross consistency check for missing or extraneous entries

  owner - Check table ownership that is inconsistent with the master database

  part_integrity - Check `pg_partition` branch integrity, partition with OIDs, partition distribution policy

  part_constraint - Check constraints on partitioned tables
unique_index_violation - Check tables that have columns with the unique index constraint for duplicate entries

dependency - Check for dependency on non-existent objects (restricted mode only)

distribution_policy - Check constraints on randomly distributed tables (restricted mode only)

namespace - Check for schemas with a missing schema definition (restricted mode only)

persistent - Check persistent tables (restricted mode only)

pgclass - Check pg_class entry that does not have any corresponding pg_attribute entry (restricted mode only)

-S {none | only}
 Specify this option to control the testing of catalog tables that are shared across all databases in the Greenplum Database installation, such as pg_database.

The value none disables testing of shared catalog tables. The value only tests only the shared catalog tables.

-U user_name
 The user connecting to Greenplum Database.

-? (help)
 Displays the online help.

-v (verbose)
 Displays detailed information about the tests that are performed.

Notes
The utility identifies tables with missing attributes and displays them in various locations in the output and in a non-standardized format. The utility also displays a summary list of tables with missing attributes in the format database.schema.table.segment_id after the output information is displayed.

If gpcheckcat detects inconsistent OID (Object ID) information, it generates one or more verification files that contain an SQL query. You can run the SQL query to see details about the OID inconsistencies and investigate the inconsistencies. The files are generated in the directory where gpcheckcat is invoked.

This is the format of the file:
gpcheckcat.verify.dbname.catalog_table_name.test_name.TIMESTAMP.sql

This is an example verification filename created by gpcheckcat when it detects inconsistent OID (Object ID) information in the catalog table pg_type in the database mydb:
gpcheckcat.verify.mydb.pg_type.missing_extraneous.20150420102715.sql

This is an example query from a verification file:

```sql
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)
) alltyprelids
GROUP BY relname, oid ORDER BY count(*) desc ;
```

gpcheckperf
Verifies the baseline hardware performance of the specified hosts.
Synopsis

**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

Description

The **gpcheckperf** utility starts a session on the specified hosts and runs the following performance tests:

- **Disk I/O Test (dd test)** — To test the sequential throughput performance of a logical disk or file system, the utility uses the *dd* command, which is a standard UNIX utility. It times how long it takes to write and read a large file to and from disk and calculates your disk I/O performance in megabytes (MB) per second. By default, the file size that is used for the test is calculated at two times the total random access memory (RAM) on the host. This ensures that the test is truly testing disk I/O and not using the memory cache.

- **Memory Bandwidth Test (stream)** — To test memory bandwidth, the utility uses the STREAM benchmark program to measure sustainable memory bandwidth (in MB/s). This tests that your system is not limited in performance by the memory bandwidth of the system in relation to the computational performance of the CPU. In applications where the data set is large (as in Greenplum Database), low memory bandwidth is a major performance issue. If memory bandwidth is significantly lower than the theoretical bandwidth of the CPU, then it can cause the CPU to spend significant amounts of time waiting for data to arrive from system memory.

- **Network Performance Test (gpnetbench*)** — To test network performance (and thereby the performance of the Greenplum Database interconnect), the utility runs a network benchmark program that transfers a 5 second stream of data from the current host to each remote host included in the test. The data is transferred in parallel to each remote host and the minimum, maximum, average and median network transfer rates are reported in megabytes (MB) per second. If the summary transfer rate is slower than expected (less than 100 MB/s), you can run the network test serially using the -r n option to obtain per-host results. To run a full-matrix bandwidth test, you can specify -r M which will cause every host to send and receive data from every other host specified. This test is best used to validate if the switch fabric can tolerate a full-matrix workload.

To specify the hosts to test, use the -f option to specify a file containing a list of host names, or use the -h option to name single host names on the command-line. If running the network performance test, all entries in the host file must be for network interfaces within the same subnet. If your segment hosts have multiple network interfaces configured on different subnets, run the network test once for each subnet.

You must also specify at least one test directory (with -d). The user who runs gpcheckperf must have write access to the specified test directories on all remote hosts. For the disk I/O test, the test directories should correspond to your segment data directories (primary and/or mirrors). For the memory bandwidth and network tests, a temporary directory is required for the test program files.

Before using gpcheckperf, you must have a trusted host setup between the hosts involved in the performance test. You can use the utility gpssh-exkeys to update the known host files and exchange public keys between hosts if you have not done so already. Note that gpcheckperf calls to gpssh and gpscp, so these Greenplum utilities must also be in your $PATH.

Options

- **-B block_size**
Specifies the block size (in KB or MB) to use for disk I/O test. The default is 32KB, which is the same as the Greenplum Database page size. The maximum block size is 1 MB.

-d test_directory
For the disk I/O test, specifies the file system directory locations to test. You must have write access to the test directory on all hosts involved in the performance test. You can use the -d option multiple times to specify multiple test directories (for example, to test disk I/O of your primary and mirror data directories).

-d temp_directory
For the network and stream tests, specifies a single directory where the test program files will be copied for the duration of the test. You must have write access to this directory on all hosts involved in the test.

-D (display per-host results)
Reports performance results for each host for the disk I/O tests. The default is to report results for just the hosts with the minimum and maximum performance, as well as the total and average performance of all hosts.

--duration time
Specifies the duration of the network test in seconds (s), minutes (m), hours (h), or days (d). The default is 15 seconds.

-f hostfile_gpcheckperf
For the disk I/O and stream tests, specifies the name of a file that contains one host name per host that will participate in the performance test. The host name is required, and you can optionally specify an alternate user name and/or SSH port number per host. The syntax of the host file is one host per line as follows:

[username@]hostname[:ssh_port]

-f hostfile_gpchecknet
For the network performance test, all entries in the host file must be for host addresses within the same subnet. If your segment hosts have multiple network interfaces configured on different subnets, run the network test once for each subnet. For example (a host file containing segment host address names for interconnect subnet 1):

sdw1-1
sdw2-1
sdw3-1

-h hostname
Specifies a single host name (or host address) that will participate in the performance test. You can use the -h option multiple times to specify multiple host names.

--netperf
Specifies that the netperf binary should be used to perform the network test instead of the Greenplum network test. To use this option, you must download netperf from http://www.netperf.org and install it into $GPHOME/bin/lib on all Greenplum hosts (master and segments).

-r ds{n|N|M}
Specifies which performance tests to run. The default is dsn:
- Disk I/O test (d)
- Stream test (s)
- Network performance test in sequential (n), parallel (N), or full-matrix (M) mode. The optional --duration option specifies how long (in seconds) to run the network test. To use the parallel (N) mode, you must run the test on an even number of hosts.
If you would rather use netperf (http://www.netperf.org) instead of the Greenplum network test, you can download it and install it into $GPHOME/bin/lib on all Greenplum hosts (master and segments). You would then specify the optional --netperf option to use the netperf binary instead of the default gpnetbench* utilities.

- **S file_size**
  Specifies the total file size to be used for the disk I/O test for all directories specified with -d. file_size should equal two times total RAM on the host. If not specified, the default is calculated at two times the total RAM on the host where gpcheckperf is executed. This ensures that the test is truly testing disk I/O and not using the memory cache. You can specify sizing in KB, MB, or GB.

- **-v** (verbose) | **-V** (very verbose)
  Verbose mode shows progress and status messages of the performance tests as they are run. Very verbose mode shows all output messages generated by this utility.

- **--version**
  Displays the version of this utility.

- **-?** (help)
  Displays the online help.

**Examples**

Run the disk I/O and memory bandwidth tests on all the hosts in the file host_file using the test directory of /data1 and /data2:

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

Run only the disk I/O test on the hosts named sdw1 and sdw2 using the test directory of /data1. Show individual host results and run in verbose mode:

```
$ gpcheckperf -h sdw1 -h sdw2 -d /data1 -r d -D -v
```

Run the parallel network test using the test directory of /tmp, where hostfile_gpchecknet_ic* specifies all network interface host address names within the same interconnect subnet:

```
$ gpcheckperf -f hostfile_gpchecknet_ic1 -r N -d /tmp
$ gpcheckperf -f hostfile_gpchecknet_ic2 -r N -d /tmp
```

Run the same test as above, but use netperf instead of the Greenplum network test (note that netperf must be installed in $GPHOME/bin/lib on all Greenplum hosts):

```
$ gpcheckperf -f hostfile_gpchecknet_ic1 -r N --netperf -d /tmp
$ gpcheckperf -f hostfile_gpchecknet_ic2 -r N --netperf -d /tmp
```

**See Also**

gpssh, gscp, gpcheck

gpconfig

Sets server configuration parameters on all segments within a Greenplum Database system.
Synopsis

```
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] [--verbose] [--debug]
gpconfig --help
```

Description

The `gpconfig` utility allows you to set, unset, or view configuration parameters from the `postgresql.conf` files of all instances (master, segments, and mirrors) in your Greenplum Database system. When setting a parameter, you can also specify a different value for the master if necessary. For example, parameters such as `max_connections` require a different setting on the master than what is used for the segments. If you want to set or unset a global or master only parameter, use the `--masteronly` option.

`gpconfig` can only be used to manage certain parameters. For example, you cannot use it to set parameters such as `port`, which is required to be distinct for every segment instance. Use the `-l` (list) option to see a complete list of configuration parameters supported by `gpconfig`.

When `gpconfig` sets a configuration parameter in a segment `postgresql.conf` file, the new parameter setting always displays at the bottom of the file. When you use `gpconfig` to remove a configuration parameter setting, `gpconfig` comments out the parameter in all segment `postgresql.conf` files, thereby restoring the system default setting. For example, if you use `gpconfig` to remove (comment out) a parameter and later add it back (set a new value), there will be two instances of the parameter; one that is commented out, and one that is enabled and inserted at the bottom of the `postgresql.conf` file.

After setting a parameter, you must restart your Greenplum Database system or reload the `postgresql.conf` files in order for the change to take effect. Whether you require a restart or a reload depends on the parameter.

For more information about the server configuration parameters, see the Greenplum Database Reference Guide.

To show the currently set values for a parameter across the system, use the `-s` option.

`gpconfig` uses the following environment variables to connect to the Greenplum Database master instance and obtain system configuration information:

- `PGHOST`
- `PGPORT`
- `PGUSER`
- `PGPASSWORD`
- `PGDATABASE`

Options

`-c | --change param_name`

Changes a configuration parameter setting by adding the new setting to the bottom of the `postgresql.conf` files.

`-v | --value value`

The value to use for the configuration parameter you specified with the `-c` option. By default, this value is applied to all segments, their mirrors, the master, and the standby master.

Parameter values that are not simple identifiers or numbers must be enclosed in single quotes (''). For example, values that contain spaces or special characters must be
enclosed in single quotes. To embed a single quote in a parameter value, enter either two single quotes or backslash and quote (\').

The utility encloses the value in single quotes when adding the setting to the postgresql.conf files.

-m | --mastevalue master_value
   The master value to use for the configuration parameter you specified with the -c option. If specified, this value only applies to the master and standby master. This option can only be used with -v.

--masteronly
   When specified, gpconfig will only edit the master postgresql.conf file.

-r | --remove param_name
   Removes a configuration parameter setting by commenting out the entry in the postgresql.conf files.

-l | --list
   Lists all configuration parameters supported by the gpconfig utility.

-s | --show param_name
   Shows the value for a configuration parameter used on all instances (master and segments) in the Greenplum Database system. If there is a difference in a parameter value among the instances, the utility displays an error message. Running gpconfig with the -s option reads parameter values directly from the database, and not the postgresql.conf file. If you are using gpconfig to set configuration parameters across all segments, then running gpconfig -s to verify the changes, you might still see the previous (old) values. You must reload the configuration files (gpstop -u) or restart the system (gpstop -r) for changes to take effect.

--file
   For a configuration parameter, shows the value from the postgresql.conf file on all instances (master and segments) in the Greenplum Database system. If there is a difference in a parameter value among the instances, the utility displays a message. Must be specified with the -s option.

   For example, the configuration parameter statement_mem is set to 64MB for a user with the ALTER ROLE command, and the value in the postgresql.conf file is 128MB. Running the command gpconfig -s statement_mem --file displays 128MB. The command gpconfig -s statement_mem --file run by the user displays 64MB.

   Not valid with the --file-compare option.

--file-compare
   For a configuration parameter, compares the current Greenplum Database value with the value in the postgresql.conf files on hosts (master and segments). The values in the postgresql.conf files represent the value when Greenplum Database is restarted.

   If the values are not the same, the utility displays the values from all hosts. If all hosts have the same value, the utility displays a summary report.

   Not valid with the --file option.

--skipvalidation
   Overrides the system validation checks of gpconfig and allows you to operate on any server configuration parameter, including hidden parameters and restricted parameters that cannot be changed by gpconfig. When used with the -l option (list), it shows the list of restricted parameters.

   Warning: Use extreme caution when setting configuration parameters with this option.
--verbose
Displays additional log information during gpconfig command execution.

--debug
Sets logging output to debug level.

-? | -h | --help
Displays the online help.

Examples
Set the gp_snmp_community parameter to testenv in the master host file only:

```
gpconfig -c gp_snmp_community -v testenv --masteronly
```

Set the max_connections setting to 100 on all segments and 10 on the master:

```
gpconfig -c max_connections -v 100 -m 10
```

Set the server configuration parameters gp_email_to and gp_email_from. These parameters require single quotes around the values.

```
$ gpconfig -c gp_email_from -v 'gpdb-server@example.com'
$ gpconfig -c gp_email_to -v 'gpdb-admin@example.com'
```

In the postgresql.conf file, the parameters are set correctly, with single quotes around the values:

```
gp_email_from='gpdb-server@example.com'
gp_email_to='gpdb-admin@example.com'
```

These examples shows the syntax required due to bash shell string processing.

```
gpconfig -c search_path -v '"$user",public'
gpconfig -c dynamic_library_path -v '"$libdir'
```

The configuration parameters are added to the postgresql.conf file.

```
search_path='"$user",public'
dynamic_library_path='"$libdir'
```

Comment out all instances of the default_statistics_target configuration parameter, and restore the system default:

```
gpconfig -r default_statistics_target
```

List all configuration parameters supported by gpconfig:

```
gpconfig -l
```

Show the values of a particular configuration parameter across the system:

```
gpconfig -s max_connections
```

See Also

gpstop
**gpcrondump**

Writes out a database to SQL script files. The script files can be used to restore the database using the `gpdbrestore` utility. The `gpcrondump` utility can be called directly or from a `crontab` entry.

**Note:** This utility is deprecated and will not be supported after the end of Greenplum Database 5.x Support Life.

### Synopsis

```
gpcrondump -x database_name
   [-s schema | -S schema | -t schema.table | -T schema.table]
   [ --table-file=filename | --exclude-table-file=filename]
   [ --schema-file=filename | --exclude-schema-file=filename]
   [--dump-stats]
   [-u backup_directory] [-R post_dump_script] [--incremental]
   [-K timestamp [--list-backup-files]]
   [--prefix prefix_string [--list-filter-tables]]
   [-c [ --cleanup-date yyyyymmdd | --cleanup-total n ] ]
   [-z] [-r]
   [-d master_data_directory] [-B parallel_processes] [-a] [-q]
   [-y reportfile] [-l logfile_directory]
   [ --email-file path_to_file ] [-v]
   { [ -E encoding] [ --inserts | --column-inserts ] [ --oids ]
     [ --no-owner | --use-set-session-authorization ] [ --no-privileges ]
     [ --rsyncable ]
     } [ --ddboost [ --replicate --max-streams max_IO_streams]
     [ --ddboost-storage-unit=unit-ID ] ]
   { [ --ddboost-show-config [ --remote ] ]
     } [ --ddboost-config-remove ]
   gpcrondump --ddboost-host ddboost_hostname
     [ --ddboost-host ddboost_hostname ... ]
     [ --ddboost-user ddboost_user ]
   gpcrondump --ddboost-backupdir backup_directory
     [ --ddboost-remote ]
   gpcrondump --ddboost-storage-unit=unit-ID

   gpcrondump --ddboost-show-config [ --remote ]
   gpcrondump --ddboost-config-remove
   gpcrondump -o [ --cleanup-date yyyyymmdd | --cleanup-total n ]
   gpcrondump -?
   gpcrondump --version
```

### Description

The `gpcrondump` utility dumps the contents of a database into SQL script files, which can then be used to restore the database schema and user data at a later time using `gpdbrestore`. During a dump operation, users will still have full access to the database.

By default, dump files are created in their respective master and segment data directories in a directory named `db_dumps/YYYYMMDD`. The data dump files are compressed by default using `gzip`. 
The utility backs up the database-level settings for the server configuration parameters `gp_default_storage_options`, `optimizer`, and `search_path`. The settings are restored when you restore the database with the `gpdbrestore` utility and specify the `-e` option to create an empty target database before performing a restore operation.

If you specify an option to back up schemas, such as `-s`, or `-S`, all procedural languages that are defined in the database are also backed up even though they are not schema specific. The languages are backed up to support any functions that might be backed up. External items such as shared libraries that are used by a language are not backed up. Also, languages are not backed up if you specify an option to back up only tables, such as `-t`, `-T`.

After a backup operation completes, the utility checks the `gpcrondump` status file for SQL execution errors and displays a warning if an error is found. The default location of the backup status files are in the `db_dumps/date/` directory.

If you specify an option that includes or excludes tables or schemas, such as `-t`, `-T`, `-s`, or `-S`, the schema qualified names of the tables that are backed up are listed in the file `gp_dump_timestamp_table`. The file is stored in the backup directory of the master segment.

`gpcrondump` allows you to schedule routine backups of a Greenplum database using `cron` (a scheduling utility for UNIX operating systems). Cron jobs that call `gpcrondump` should be scheduled on the master host.

**Warning:** Backing up a database with `gpcrondump` while simultaneously running `ALTER TABLE` might cause `gpcrondump` to fail.

Back up a database with `gpcrondump` while simultaneously running DDL commands might cause issues with locks. You might see either the DDL command or `gpcrondump` waiting to acquire locks.

### About Database, Schema, and Table Names

You can specify names of databases, schemas, and tables that contain these special characters.

```
" ' ` $ % ^ & * ( ) _ - + ] { } > < \ | ; : / ? and the space character.
```

**Note:** The characters `!`, comma (`,`), and period (`.`) are not supported. Also, the tab (`\t`) and newline (`\n`) characters are not supported.

When the name contains special characters and is specified on the command line, the name must be enclosed in double quotes (`"`). Double quotes are optional for names that do not contain special characters. For example, either use of quotes is valid on the command line "my#1schema".mytable or "my#1schema"."mytable". Within the name, these special characters must be escaped with a backslash (`\`):

```
" ` $ \.
```

When the name is specified in an input file, the name must *not* be enclosed in double quotes. Special characters do not require escaping.

### Backup Filenames

The utility creates backup files with this file name format.

```
prefix_gp_dump_content_dbid_timestamp
```

The `content` and `dbid` are identifiers for the Greenplum Database segment instances that are assigned by Greenplum Database. For information about the identifiers, see the Greenplum Database system catalog table `gp_segment_configuration` in the `Greenplum Database Reference Guide`.

### Using Data Domain Boost

The `gpcrondump` utility is used to schedule Data Domain Boost (DD Boost) backup operations. The utility is also used to set, change, or remove one-time credentials and storage unit ID for DD Boost. The
gpcrondump, gpdbrestore, and gpmfr utilities use the DD Boost credentials to access Data Domain systems. DD Boost information is stored in these files.

- **DDBOOST_CONFIG** is used by gpdbrestore and gpcrondump for backup and restore operations with the Data Domain system. The gpdbrestore utility creates or updates the file when you specify Data Domain information with the --ddboost-host option.
- **DDBOOST_MFR_CONFIG** is used by gpmfr for remote replication operations with the remote Data Domain system. The gpdbrestore utility creates or updates the file when you specify Data Domain information with the --ddboost-host option and the --ddboost-remote option.

The configuration files are created in the current user (gpadmin) home directory on the Greenplum Database master and segment hosts. The path and file name cannot be changed.

When you use DD Boost to perform a backup operation, the operation uses a storage unit on a Data Domain system. You can specify the storage unit ID when you perform these operations:

- When you set the DD Boost credentials with the --ddboost-host option. If you specify the --ddboost-storage-unit option, the storage unit ID is written to the Greenplum Database DD Boost configuration file DDBOOST_CONFIG. If the storage unit ID is not specified, the default value is GPDB.
- When you perform a backup operation with the --ddboost option. When you specify the --ddboost-storage-unit option, the utility uses the specified Data Domain storage unit for the operation. The value in the configuration file is not changed.

When performing a full backup operation (not an incremental backup), the storage unit is created on the Data Domain system if it does not exist.

A storage unit is not created if these gpcrondump options are specified: --incremental, --list-backup-file, --list-filter-tables, -o, or --ddboost-config-remove.

Use the gpcrondump option --ddboost-show-config to display the current DD Boost configuration information from the master configuration file. Specify the --remote option to display the configuration information for the remote Data Domain system.

For information about using DD Boost and Data Domain systems with Greenplum Database, see “Backing Up and Restoring Databases” in the *Greenplum Database Administrator Guide*.

### Using NetBackup

Veritas NetBackup integration is included with Pivotal Greenplum Database. Greenplum Database must be configured to communicate with the Veritas NetBackup master server that is used to backup the database.

When backing up a large amount of data, set the NetBackup CLIENT_READ_TIMEOUT option to a value that is at least twice the expected duration of the operation (in seconds). The CLIENT_READ_TIMEOUT default value is 300 seconds (5 minutes).

See the *Greenplum Database Administrator Guide* for information on configuring Greenplum Database and NetBackup and backing up and restoring with NetBackup.

### About Return Codes

The following is a list of the codes that gpcrondump returns.

- **0** – Dump completed with no problems
- **1** – Dump completed, but one or more warnings were generated
- **2** – Dump failed with a fatal error

### Email Notifications

To have gpcrondump send out status email notifications after a back up operation completes, you must place a file named mail_contacts in the home directory of the Greenplum superuser (gpadmin) or in the same directory as the gpcrondump utility ($GPHOME/bin). This file should contain one email address per line. gpcrondump will issue a warning if it cannot locate a mail_contacts file in either location. If both locations have a mail_contacts file, then the one in $HOME takes precedence.
You can customize the email Subject and From lines of the email notifications that gpcrondump sends after a backup completes for a database. You specify the option --email-file with the location of a YAML file that contains email Subject and From lines that gpcrondump uses. For information about the format of the YAML file, see File Format for Customized Emails.

Note: The UNIX mail utility must be running on Greenplum Database host and must be configured to allow the Greenplum superuser (gpadmin) to send email.

Limitations
Dell EMC DD Boost is integrated with Pivotal Greenplum Database and requires a DD Boost license. Open source Greenplum Database cannot use the DD Boost software, but can back up to a Dell EMC Data Domain system mounted as an NFS share on the Greenplum master and segment hosts.

NetBackup is not compatible with DD Boost. Both NetBackup and DD Boost cannot be used in a single back up operation.

For incremental back up sets, a full backup and associated incremental backups, the backup set must be on a single device. For example, a backup set must all be on a file system. The backup set cannot have some backups on the local file system and others on a Data Domain system or a NetBackup system.

For external tables, the table definition is backed up, however the data is not backed up. For leaf child partition of a partitioned table that is a readable external table, the leaf child partition data is not backed up.

Options
-a (do not prompt)
   Do not prompt the user for confirmation.

-b (bypass disk space check)
   Bypass disk space check. The default is to check for available disk space, unless --ddboost is specified. When using Data Domain Boost, this option is always enabled.

   Note: Bypassing the disk space check generates a warning message. With a warning message, the return code for gpcrondump is 1 if the dump is successful. (If the dump fails, the return code is 2, in all cases.)

-B parallel_processes
   The number of segments to check in parallel for pre/post-dump validation. If not specified, the utility will start up to 60 parallel processes depending on how many segment instances it needs to dump.

-c (clear old dump files first)
   Specify this option to delete old backups before performing a backup. In the db_dumps directory, the directory where the name is the oldest date is deleted. If the directory name is the current date, the directory is not deleted. The default is to not delete old backup files.

   The deleted directory might contain files from one or more backups.

   Warning: Before using this option, ensure that incremental backups required to perform the restore are not deleted. The gpdbrestore utility option --list-backup lists the backup sets required to perform a backup.

   If --ddboost is specified, only the old files on Data Domain Boost are deleted.

   You can specify the option --cleanup-date or --cleanup-total to specify backup sets to delete.

   This option is not supported with the -u option.

-C (clean catalog before restore)
   Clean out the catalog schema prior to restoring database objects. gpcrondump adds the DROP command to the SQL script files when creating the backup files. When the script files
are used by the `gpdbrestore` utility to restore database objects, the `DROP` commands remove existing database objects before restoring them.

If `--incremental` is specified and the files are on NFS storage, the `--c` option is not supported. The database objects are not dropped if the `--c` option is specified.

`--cleanup-date=yyyy-mm-dd`
Remove backup sets for the date `yyyy-mm-dd`. The date format is `yyyymmdd`. If multiple backup sets were created on the date, all the backup sets for that date are deleted. If no backup sets are found, `gpcrondump` returns a warning message and no backup sets are deleted. If the `--c` option is specified, the backup process continues.

Valid only with the `--c` or `--o` option.

**Warning:** Before using this option, ensure that incremental backups required to perform the restore are not deleted. The `gpdbrestore` utility option `--list-backup` lists the backup sets required to perform a backup.

`--cleanup-total=n`
Remove the `n` oldest backup sets based on the backup timestamp.

If there are fewer than `n` backup sets, `gpcrondump` returns a warning message and no backup sets are deleted. If the `--c` option is specified, the backup process continues.

Valid only with the `--c` or `--o` option.

**Warning:** Before using this option, ensure that incremental backups required to perform the restore are not deleted. The `gpdbrestore` utility option `--list-backup` lists the backup sets required to perform a backup.

`--column-inserts`
Dump data as `INSERT` commands with column names.

If `--incremental` is specified, this option is not supported.

`-d master_data_directory`
The master host data directory. If not specified, the value set for `$MASTER_DATA_DIRECTORY` will be used.

`--ddboost [--replicate --max-streams max_IO_streams ] [--ddboost-skip-ping]`
Use Data Domain Boost for this backup. Before using Data Domain Boost, set up the Data Domain Boost credential with the `--ddboost-host` option. Also, see Using Data Domain Boost.

If `--ddboost` is specified, the `--z` option (uncompressed) is recommended.

Backup compression (turned on by default) should be turned off with the `--z` option. Data Domain Boost will deduplicate and compress the backup data before sending it to the Data Domain system.

`--replicate --max-streams max_IO_streams` is optional. If you specify this option, `gpcrondump` replicates the backup on the remote Data Domain server after the backup is complete on the primary Data Domain server. `max_IO_streams` specifies the maximum number of Data Domain I/O streams that can be used when replicating the backup set on the remote Data Domain server from the primary Data Domain server.

You can use `gpmfr` to replicate a backup if replicating a backup with `gpcrondump` takes a long time and prevents other backups from occurring. Only one instance of `gpcrondump` can be running at a time. While `gpcrondump` is being used to replicate a backup, it cannot be used to create a backup.

You can run a mixed backup that writes to both a local disk and Data Domain. If you want to use a backup directory on your local disk other than the default, use the `--u` option.
Mixed backups are not supported with incremental backups. For more information about mixed backups and Data Domain Boost, see "Backing Up and Restoring Databases" in the Greenplum Database Administrator Guide.

**Important:** Never use the Greenplum Database default backup options with Data Domain Boost.

To maximize Data Domain deduplication benefits, retain at least 30 days of backups.

**Note:** The -b, -c, -f, -G, -g, -R, and -u options change if --ddboost is specified. See the options for details.

The DDBoost backup options are not supported if the NetBackup options are specified.

```bash
--ddboost-host ddboost_hostname [--ddboost-host ddboost_hostname ...]
--ddboost-user ddboost_user --ddboost-backupdir backup_directory
[--ddboost-remote] [--ddboost-skip-ping]
```

Sets the Data Domain Boost credentials. Do not combine this options with any other gpcrondump options. Do not enter just one part of this option.

*ddboost_hostname* is the IP address (or hostname associated to the IP) of the host. There is a 30-character limit. If you use two or more network connections to connect to the Data Domain system, specify each connection with the --ddboost-host option.

*ddboost_user* is the Data Domain Boost user name. There is a 30-character limit.

*backup_directory* is the location for the backup files, configuration files, and global objects on the Data Domain system. The location on the system is `GPDB/backup_directory`.

--ddboost-remote is optional. It indicates that the configuration parameters are for the remote Data Domain system used for backup replication and Data Domain Boost managed file replication. Credentials for the remote Data Domain system must be configured to use the --replicate option or the gpmfr management utility.

For example:

```bash
gpcrondump --ddboost-host 192.0.2.230 --ddboost-user ddboostusername --ddboost-backupdir gp_production
```

**Note:** When setting Data Domain Boost credentials, the --ddboost-backupdir option is ignored if the --ddboost-remote option is specified for a Data Domain system that is used for the replication of backups. The --ddboost-backupdir value is for backup operations with a Data Domain system, not for backup replication.

After running gpcrondump with these options, the system verifies the limits on the host and user names and prompts for the Data Domain Boost password. Enter the password when prompted; the password is not echoed on the screen. There is a 40-character limit on the password that can include lowercase letters (a-z), uppercase letters (A-Z), numbers (0-9), and special characters ($, %, #, +, etc.).

The system verifies the password. After the password is verified, the system creates encrypted DDBOOST_CONFIG files in the user's home directory.

In the example, the --ddboost-backupdir option specifies the backup directory `gp_production` in the Data Domain Storage Unit GPDB.

**Note:** If there is more than one operating system user using Data Domain Boost for backup and restore operations, repeat this configuration process for each of those users.

**Important:** Set up the Data Domain Boost credential before running any Data Domain Boost backups with the --ddboost option, described above.

```bash
--ddboost-config-remove
```
Removes all Data Domain Boost credentials from the master and all segments on the system. Do not enter this option with any other gpcrondump option.

--ddboost-show-config [--remote]
Optional. Displays the DD Boost configuration file information for the Data Domain server. Specify this option with the --remote option to display the configuration file information for remote Data Domain server. No backup is performed.

--ddboost-skip-ping
Specify this option to skip the ping of a Data Domain system. When working with a Data Domain system, ping is used to ensure that the Data Domain system is reachable. If the Data Domain system is configured to block ICMP ping probes, specify this option.

--ddboost-storage-unit=unit-ID
Optional. Specify a valid storage unit name for the Data Domain system that is used for backup and restore operations. The default storage unit ID is GPDB. See Using Data Domain Boost.

- Specify this option with the --ddboost-host option to create or update the storage unit ID in the DD Boost credentials file.
- Specify this option with the --ddboost option to override the storage unit ID in the DD Boost credentials file when performing a backup operation.

When performing a full backup operation (not an incremental backup), the storage unit is created on the Data Domain system if it does not exist.

A replication operation uses the same storage unit ID on both local and remote Data Domain systems.

--dump-stats
Specify this option to back up database statistics. The data is written to an SQL file and can be restored manually or with gpdbrestore utility.

The statistics are written in the master data directory to db_dumps/YYYYMMDD/prefix_string_gp_statistics-1_1_timestamp.

If this option is specified with options that include or exclude tables or schemas, the utility backs up only the statistics for the tables that are backed up.

-E encoding
Character set encoding of dumped data. Defaults to the encoding of the database being dumped. See the Greenplum Database Reference Guide for the list of supported character sets.

-email-file path_to_file
Specify the fully-qualified location of the YAML file that contains the customized Subject and From lines that are used when gpcrondump sends notification emails about a database back up.

For information about the format of the YAML file, see File Format for Customized Emails.

-f free_space_percent
When checking that there is enough free disk space to create the dump files, specifies a percentage of free disk space that should remain after the dump completes. The default is 10 percent.

This is option is not supported if --ddboost or --incremental is specified.

-g (copy config files)
Secure a copy of the master and segment configuration files postgresql.conf, pg_ident.conf, and pg_hba.conf. These configuration files are dumped in the master or segment data directory to db_dumps/YYYYMMDD/config_files_timestamp.tar.
If --ddboost is specified, the backup is located on the default storage unit in the directory specified by --ddboost-backupdir when the Data Domain Boost credentials were set.

-G (dump global objects)
Back up database metadata information that is not associated with any particular schema or table such as roles and tablespaces. Global objects are dumped in the master data directory to db_dumps/ YYYYMMDD/prefix_string_gp_global_-l_l_l_timestamp.

If --ddboost is specified, the backup is located on the default storage unit in the directory specified by --ddboost-backupdir when the Data Domain Boost credentials were set.

-h (record dump details)
Record details of the database dump in database table public.gpcrondump_history in the database supplied via -x option. The gpcrondump utility will create the table if it does not currently exist. The public schema must exist in the database so that gpcrondump can create the public.gpcrondump_history table. The default is to record the database dump details.

This option will be deprecated in a future release.

-H (disable recording dump details)
Disable recording details of database dump in database table public.gpcrondump_history in the database supplied via -x option. If not specified, the utility will create/update the history table. The -H option cannot be selected with the -h option.

Note: The gpcrondump utility creates the public.gpcrondump_history table by default. If the public schema has been deleted from the database, you must specify the -H option to prevent gpcrondump from returning an error when it attempts to create the table.

--incremental (backup changes to append-optimized tables)
Adds an incremental backup to a backup set. When performing an incremental backup, the complete backup set created prior to the incremental backup must be available. The complete backup set includes the following backup files:

- The last full backup before the current incremental backup
- All incremental backups created between the time of the full backup and the current incremental backup

An incremental backup is similar to a full backup except for append-optimized tables, including column-oriented tables. An append-optimized table is backed up only if one of the following operations was performed on the table after the last backup:

ALTER TABLE
INSERT
DELETE
UPDATE
TRUNCATE
DROP and then re-create the table

For partitioned append-optimized tables, only the changed table partitions are backed up.

The -u option must be used consistently within a backup set that includes a full and incremental backups. If you use the -u option with a full backup, you must use the -u option when you create incremental backups that are part of the backup set that includes the full backup.

You can create an incremental backup for a full backup of set of database tables. When you create the full backup, specify the --prefix option to identify the backup. To include a set of tables in the full backup, use either the -t option or --table-file option. To exclude a set of tables, use either the -T option or the --exclude-table-file option. See the description of the option for more information on its use.
To create an incremental backup based on the full backup of the set of tables, specify the option --incremental and the --prefix option with the string specified when creating the full backup. The incremental backup is limited to only the tables in the full backup.

**Warning:** gpcrondump does not check for available disk space prior to performing an incremental backup.

**Important:** An incremental back up set, a full backup and associated incremental backups, must be on a single device. For example, a the backups in a backup set must all be on a file system or must all be on a Data Domain system.

--inserts

Dump data as INSERT, rather than COPY commands.

If --incremental is specified, this option is not supported.

-j (vacuum before dump)

Run VACUUM before the dump starts.

-K timestamp [--list-backup-files]

Specify the timestamp that is used when creating a backup. The timestamp is 14-digit string that specifies a date and time in the format yyyymmddhhmms. The date is used for backup directory name. The date and time is used in the backup file names. If -K timestamp is not specified, a timestamp is generated based on the system time.

When adding a backup to set of backups, gpcrondump returns an error if the timestamp does not specify a date and time that is more recent than all other backups in the set.

--list-backup-files is optional. When you specify both this option and the -K timestamp option, gpcrondump does not perform a backup. gpcrondump creates two text files that contain the names of the files that will be created when gpcrondump backs up a Greenplum database. The text files are created in the same location as the backup files.

The file names use the timestamp specified by the -K timestamp option and have the suffix _pipes and _regular_files. For example:

```
gp_dump_20130514093000_pipes
gp_dump_20130514093000_regular_files
```

The _pipes file contains a list of file names that be can be created as named pipes. When gpcrondump performs a backup, the backup files will generate into the named pipes. The _regular_files file contains a list of backup files that must remain regular files. gpcrondump and gpdbrestore use the information in the regular files during backup and restore operations. To backup a complete set of Greenplum Database backup files, the files listed in the _regular_files file must also be backed up after the completion of the backup job.

To use named pipes for a backup, you need to create the named pipes on all the Greenplum Database hosts and make them writable before running gpcrondump.

If --ddboost is specified, -K timestamp [--list-backup-files] is not supported.

-k (vacuum after dump)

Run VACUUM after the dump has completed successfully.

-l logfile_directory

The directory to write the log file. Defaults to ~/gpAdminLogs.

--netbackup-block-size size

Specify the block size, in bytes, of data being transferred to the Veritas NetBackup server. The default is 512 bytes.
NetBackup options are not supported if DDBoost backup options are specified.

--netbackup-keyword keyword
    Specify a keyword for the backup that is transferred to the Veritas NetBackup server.
    NetBackup adds the keyword property and the specified keyword value to the
    NetBackup .img files that are created for the backup.
    The maximum length of this parameter is 127 characters.
    NetBackup options are not supported if DDBoost backup options are specified.

--netbackup-policy netbackup_policy
    The name of the NetBackup policy created for backing up Greenplum Database.
    NetBackup options are not supported if DDBoost backup options are specified.
    The maximum length of this parameter is 127 characters.

--netbackup-service-host netbackup_server
    The NetBackup master server that Greenplum Database connects to when backing up to
    NetBackup.
    NetBackup options are not supported if DDBoost backup options are specified.
    The maximum length of this parameter is 127 characters.

--netbackup-schedule netbackup_schedule
    The name of the NetBackup schedule created for backing up Greenplum Database.
    NetBackup options are not supported if DDBoost backup options are specified
    The maximum length of this parameter is 127 characters.

--no-owner
    Do not output commands to set object ownership.

--no-privileges
    Do not output commands to set object privileges (GRANT/REVOKE commands).

-o (clear old dump files only)
    Clear out old dump files only, but do not run a dump. This will remove the oldest dump
directory except the current date’s dump directory. All dump sets within that directory will
be removed.

    Warning: Before using this option, ensure that incremental backups required
to perform the restore are not deleted. The gpdbrestore utility option --
    list-backup lists the backup sets required to perform a backup.

    If --ddboost is specified, only the old files on Data Domain Boost are deleted.
    You can specify the option --cleanup-date or --cleanup-total to specify backup
    sets to delete.
    If --incremental is specified, this option is not supported.

--oids
    Include object identifiers (oid) in dump data.
    If --incremental is specified, this option is not supported.

--prefix prefix_string [--list-filter-tables]
    Prepends prefix_string followed by an underscore character (_) to the names of all
    the backup files created during a backup.

    --list-filter-tables is optional. When you specify both options, gpcrondump does
    not perform a backup. For the full backup created by gpcrondump that is identified by
    the prefix-string, the tables that were included or excluded for the backup are listed.
You must also specify the \texttt{--incremental} option if you specify the \texttt{--list-filter-tables} option.

If \texttt{--ddboost} is specified, \texttt{--prefix prefix_string [--list-filter-tables]} is not supported.

\textbf{-q (no screen output)}

Run in quiet mode. Command output is not displayed on the screen, but is still written to the log file.

\textbf{-r (rollback on failure)}

Rollback the dump files (delete a partial dump) if a failure is detected. The default is to not rollback.

\textit{Note:} This option is not supported if \texttt{--ddboost} is specified.

\textbf{-R post_dump_script}

The absolute path of a script to run after a successful dump operation. For example, you might want a script that moves completed dump files to a backup host. This script must reside in the same location on the master and all segment hosts.

\textbf{--rsyncable}

Passes the \texttt{--rsyncable} flag to the \texttt{gzip} utility to synchronize the output occasionally, based on the input during compression. This synchronization increases the file size by less than 1% in most cases. When this flag is passed, the \texttt{rsync(l)} program can synchronize compressed files much more efficiently. The \texttt{gunzip} utility cannot differentiate between a compressed file created with this option, and one created without it.

\textbf{-s schema_name}

Dump all the tables that are qualified by the specified schema in the database. The \texttt{-s} option can be specified multiple times. System catalog schemas are not supported. If you want to specify multiple schemas, you can also use the \texttt{--schema-file=filename} option in order not to exceed the maximum token limit.

Only a set of tables or set of schemas can be specified. For example, the \texttt{-s} option cannot be specified with the \texttt{-t} option.

If \texttt{--incremental} is specified, this option is not supported.

\textbf{-S schema_name}

A schema name to \texttt{exclude} from the database dump. The \texttt{-S} option can be specified multiple times. If you want to specify multiple schemas, you can also use the \texttt{--exclude-schema-file=filename} option in order not to exceed the maximum token limit.

Only a set of tables or set of schemas can be specified. For example, this option cannot be specified with the \texttt{-t} option.

If \texttt{--incremental} is specified, this option is not supported.

\textbf{-t schema.table_name}

Dump only the named table in this database. The \texttt{-t} option can be specified multiple times. If you want to specify multiple tables, you can also use the \texttt{--table-file=filename} option in order not to exceed the maximum token limit.

Only a set of tables or set of schemas can be specified. For example, this option cannot be specified with the \texttt{-s} option.

If \texttt{--incremental} is specified, this option is not supported.

\textbf{-T schema.table_name}

A table name to \texttt{exclude} from the database dump. The \texttt{-T} option can be specified multiple times. If you want to specify multiple tables, you can also use the \texttt{--exclude-table-file=filename} option in order not to exceed the maximum token limit.
Only a set of tables or set of schemas can be specified. For example, this option cannot be specified with the -s option.

If --incremental is specified, this option is not supported.

--exclude-schema-file=filename
Excludes all the tables that are qualified by the specified schemas listed in the filename from the database dump. The file filename contains any number of schemas, listed one per line.

Only a set of tables or set of schemas can be specified. For example, this option cannot be specified with the -t option.

If --incremental is specified, this option is not supported.

--exclude-table-file=filename
Excludes all tables listed in the filename from the database dump. The file filename contains any number of tables, listed one per line.

Only a set of tables or set of schemas can be specified. For example, this cannot be specified with the -s option.

If --incremental is specified, this option is not supported.

--schema-file=filename
Dumps only the tables that are qualified by the schemas listed in the filename. The file filename contains any number of schemas, listed one per line.

Only a set of tables or set of schemas can be specified. For example, this option cannot be specified with the -t option.

If --incremental is specified, this option is not supported.

--table-file=filename
Dumps only the tables listed in the filename. The file filename contains any number of tables, listed one per line.

Only a set of tables or set of schemas can be specified. For example, this cannot be specified with the -s option.

If --incremental is specified, this option is not supported.

-u backup_directory
Specifies the absolute path where the backup files will be placed on each host. If the path does not exist, it will be created, if possible. If not specified, defaults to the data directory of each instance to be backed up. Using this option may be desirable if each segment host has multiple segment instances as it will create the dump files in a centralized location rather than the segment data directories.

Note: This option is not supported if --ddboost is specified.

--use-set-session-authorization
Use SET SESSION AUTHORIZATION commands instead of ALTER OWNER commands to set object ownership.

-v | --verbose
Specifies verbose mode.

--version (show utility version)
Displays the version of this utility.

-x database_name
Required. The name of the Greenplum database to dump.

-y reportfile
This option is deprecated and will be removed in a future release. If specified, a warning message is returned stating that the \(-y\) option is deprecated.

Specifies the full path name where a copy of the backup job log file is placed on the master host. The job log file is created in the master data directory or if running remotely, the current working directory.

\(-z\) (no compression)

Do not use compression. Default is to compress the dump files using gzip.

Use this option \((-z\) for NFS and Data Domain Boost backups.

\(-?\) (help)

Displays the online help.

File Format for Customized Emails

You can configure \texttt{gpcrondump} to send an email notification after a back up operation completes for a database. To customize the From and Subject lines of the email that are sent for a database, you create a YAML file and specify the location of the file with the option \texttt{--email-file}. In the YAML file, you can specify a different From and Subject line for each database that \texttt{gpcrondump} backs up. This is the format of the YAML file to specify a custom From and Subject line for a database:

\begin{verbatim}
EMAILDETAILS:
  - DBNAME: database_name
    FROM: from_user
    SUBJECT: subject_text
\end{verbatim}

When email notification is configured for \texttt{gpcrondump}, the \texttt{from_user} and the \texttt{subject_text} are the strings that \texttt{gpcrondump} uses in the email notification after completing the back up for \texttt{database_name}.

This example YAML file specifies different From and Subject lines for the databases \texttt{testdb100} and \texttt{testdb200}.

\begin{verbatim}
EMAILDETAILS:
  - DBNAME: testdb100
    FROM: RRP_MPE2_1
    SUBJECT: backup completed for Database 'testdb100'
  - DBNAME: testdb200
    FROM: Report_from_DCDDEV_host
    SUBJECT: Completed backup for database 'testdb200'
\end{verbatim}

Examples

Call \texttt{gpcrondump} directly and dump \texttt{mydatabase} (and global objects):

\begin{verbatim}
gpcrondump -x mydatabase -c -g -G
\end{verbatim}

A \texttt{crontab} entry that runs a backup of the \texttt{sales} database (and global objects) nightly at one past midnight:

\begin{verbatim}
01 0 * * * /home/gpadmin/gpdump.sh >> gpdump.log
\end{verbatim}

The content of dump script \texttt{gpdump.sh} is:

\begin{verbatim}
#!/bin/bash
\end{verbatim}
export GPHOME=/usr/local/greenplum-db
export MASTER_DATA_DIRECTORY=/data/gpdb_p1/gp-1
. $GPHOME/greenplum_path.sh
gpcrondump -x sales -c -g -G -a -q

This example creates two text files, one with the suffix _pipes and the other with _regular_files. The _pipes file contain the file names that can be named pipes when you backup the Greenplum database mytestdb.

```
gpcrondump -x mytestdb -K 20131030140000 --list-backup-files
```

To use incremental backup with a set of database tables, you must create a full backup of the set of tables and specify the --prefix option to identify the backup set. The following example uses the --table-file option to create a full backup of the set of files listed in the file user-tables. The prefix user_backup identifies the backup set.

```
gpcrondump -x mydatabase --table-file=user-tables
    --prefix user_backup
```

To create an incremental backup for the full backup created in the previous example, specify the --incremental option and the option --prefix user_backup to identify backup set. This example creates an incremental backup.

```
gpcrondump -x mydatabase --incremental --prefix user_backup
```

This command lists the tables that were included or excluded for the full backup.

```
gpcrondump -x mydatabase --incremental --prefix user_backup
    --list-filter-tables
```

This command backs up the database customer and specifies a NetBackup policy and schedule that are defined on the NetBackup master server nbu_server1. A block size of 1024 bytes is used to transfer data to the NetBackup server.

```
gpcrondump -x customer --netbackup-service-host=nbu_server1
    --netbackup-policy=gpdb_cust --netbackup-schedule=gpdb_backup
    --netbackup-block-size=1024
```

See Also

gpdbrestore

gpdbrestore

Restores a database from a set of dump files generated by gpcrondump.

**Note:** This utility is deprecated and will not be supported after the end of Greenplum Database 5.x Support Life.

Synopsis

```
```
gpdbrestore -?

Description
The `gpdbrestore` utility recreates the data definitions (schema) and user data in a Greenplum database using the script files created by `gpcrondump` operations.

When you restore from an incremental backup, the `gpdbrestore` utility assumes the complete backup set is available. The complete backup set includes the following backup files:

- The last full backup before the specified incremental backup
- All incremental backups created between the time of the full backup and the specified incremental backup

The `gpdbrestore` utility provides the following functionality:

- Automatically reconfigures for compression.
- Validates the number of dump files are correct (for primary only, mirror only, primary and mirror, or a subset consisting of some mirror and primary segment dump files).
- If a failed segment is detected, restores to active segment instances.
- Except when restoring data from a NetBackup server, you do not need to know the complete timestamp key (`-t`) of the backup set to restore. Additional options are provided to instead give just a date (`-b`), backup set directory location (`-R`), or database name (`-s`) to restore.
- The `-R` option allows the ability to restore from a backup set located on a host outside of the Greenplum Database array (archive host). Ensures that the correct dump file goes to the correct segment instance.
- Identifies the database name automatically from the backup set.
- Allows you to restore particular tables only (`-T` option) instead of the entire database. Note that single tables are not automatically dropped or truncated prior to restore.
- Performs an `ANALYZE` operation on the tables that are restored. You can disable the `ANALYZE` operation by specifying the option `--noanalyze`.
- Can restore global objects such as roles and tablespaces (`-G` option).
- Detects if the backup set is primary segments only or primary and mirror segments and performs the appropriate restore operation.
- Allows you to drop the target database before a restore in a single operation.

The backups contain the database-level settings for the server configuration parameters `gp_default_storage_options`, `optimizer`, and `search_path`, the settings are restored when you perform a restore operation and specify the `-e` option to create an empty target database before performing a restore operation.

**Important:** When restoring table data to an existing table, the utility assumes that the database table definition is the same as the table that was backed up. The utility does not check the table definitions.

Database, Schema, and Table Names
You can specify names of databases, schemas, and tables that contain these special characters.

"' ` - # $ % & * ( ) _ + [ ] { } > < \ | ; : / ? and the space character.
Note: The characters !, comma (,), and period (.) are not supported. Also, the tab (\t) and newline (\n) characters are not supported.

When the name contains special characters and is specified on the command line, the name must be enclosed in double quotes (""). Double quotes are optional for names that do not contain special characters. For example, either use of quotes is valid on the command line "my#1schema".mytable or "my#1schema"."mytable". Within the name, these special characters must be escaped with a backslash (\): " ` $ \.

When the name is specified in an input file, the name must not be enclosed in double quotes. Special characters do not require escaping.

Restoring from a Data Domain System with DD Boost

When you create a backup with gpcrondump using DD Boost, the backup is stored on a Data Domain system storage unit. When restore the backup, you must use the same storage unit ID that was used when you backed up the data. You can use the gpcrondump option --ddboost-show-config to display the current DD Boost configuration information that includes the storage unit ID.

For information about using DD Boost and Data Domain systems with Greenplum Database, see "Backing Up and Restoring Databases" in the Greenplum Database Administrator Guide.

NetBackup is not compatible with DD Boost. Both NetBackup and DD Boost cannot be used in a single backup operation.

Restoring a Database from NetBackup

Greenplum Database must be configured to communicate with the Veritas NetBackup master server that is used to restore database data. See the Greenplum Database System Administrator Guide for information about configuring Greenplum Database and NetBackup.

When restoring from NetBackup server, you must specify the timestamp of the backup with the -t option.

When restoring a large amount of data, set the NetBackup CLIENT_READ_TIMEOUT option to a value that is at least twice the expected duration of the operation (in seconds). The CLIENT_READ_TIMEOUT default value is 300 seconds (5 minutes).

NetBackup is not compatible with DD Boost. Both NetBackup and DD Boost cannot be used in a single backup operation.

Restoring a Database with Named Pipes

If you used named pipes when you backed up a database with gpcrondump, named pipes with the backup data must be available when restoring the database from the backup.

Error Reporting

After a restore operation completes, the utility checks the gpdbrestore status file for SQL execution errors and displays a warning if an error is found. The default location of the restore status files are in the db_dumps/date/ directory.

Options
-a (do not prompt)
Do not prompt the user for confirmation.

-b YYYYMMDD
Looks for dump files in the segment data directories on the Greenplum Database array of hosts in db_dumps/YYYYMMDD. If --ddboost is specified, the systems looks for dump files on the Data Domain Boost host.

-B parallel_processes
The number of segments to check in parallel for pre/post-restore validation. If not specified, the utility will start up to 60 parallel processes depending on how many segment instances it needs to restore.
--change-schema=\texttt{schema\_name}

Optional. Restores tables from a backup created with \texttt{gpcrondump} to a different schema. The \texttt{schema\_name} must exist in the database. If the schema does not exist, the utility returns an error. System catalog schemas are not supported.

You must specify tables to restore with the \texttt{-T} and \texttt{--table-file} options. If a table that is being restored exists in \texttt{schema-name}, the utility returns a warning and attempts to append the data to the table from the backup. You can specify the \texttt{--truncate} option to truncate table data before restoring data to the table from the backup.

This option is not supported if \texttt{-S} is specified.

\texttt{-d \texttt{master\_data\_directory}}

Optional. The master host data directory. If not specified, the value set for \texttt{$\text{MASTER\_DATA\_DIRECTORY}$} will be used.

\texttt{--ddboost [--ddboost-storage-unit=\texttt{unit-ID} ]}

Use Data Domain Boost for this restore operation, if the \texttt{--ddboost} option was specified when the data was backed up with the \texttt{gpcrondump} utility. Before using Data Domain Boost, make sure the one-time Data Domain Boost credential setup is complete.

The \texttt{--ddboost-storage-unit} option is optional. When restoring a backup from a Data Domain server, you must use the same storage unit that was used when you backed up the data.

See \textit{Restoring from a Data Domain System with DD Boost}.

If you backed up Greenplum Database configuration files with the \texttt{gpcrondump} option \texttt{-g} and specified the \texttt{--ddboost} option, you must manually restore the backup from the Data Domain system. The configuration files must be restored for the Greenplum Database master and all the hosts and segments. The backup location on the Data Domain system is the directory \texttt{unit_ID/backup_directory/date}. The \texttt{backup_directory} is set when you specify the Data Domain credentials with \texttt{gpcrondump}. The \texttt{unit_ID} is the Data Domain system storage unit that was used when you backed up the database.

This option is not supported if \texttt{--netbackup-service-host} is specified.

\texttt{-e (create target database before restore)}

Creates the target database and then performs a restore operation. If the target database exists, drops the existing database before creating the database and performing a restore operation.

The utility restores database-level settings for the server configuration parameters \texttt{gp\_default\_storage\_options}, \texttt{optimizer}, and \texttt{search\_path}.

\texttt{-G [include | only] (restore global objects)}

Restores database metadata information that is not associated with a specific schema or table, such as roles and tablespaces, if the global object dump file \texttt{db\_dumps/date/prefix\_string gp\_global\_1\_1\_timestamp} is found in the master data directory. The global object file is created with the \texttt{gpcrondump} option \texttt{-G}.

- The keyword \texttt{include} restores global objects in addition to performing a restore. This is the default if no keyword is specified.
- The keyword \texttt{only} restores only global objects. No other database objects or database table data are restored.

The \texttt{-m} option restores metadata associated with schemas or tables.

\texttt{-l \texttt{logfile\_directory}}

The directory to write the log file. Defaults to \texttt{~/gpAdminLogs}.

\texttt{--list-backup}
Lists the set of full and incremental backup sets required to perform a restore based on the timestamp_key specified with the -t option and the location of the backup set.

This option is supported only if the timestamp_key is for an incremental backup.

-L (list tablenames in backup set)
When used with the -t option, lists the table names that exist in the named backup set and exits. Does not perform a restore.

-m (restore metadata only)
Restores database metadata information such schema and table definitions and information created by SET statements. This option does not restore database table data. All table and schema metadata is restored unless options are specified that include or exclude tables or schemas. If table or schema filters are specified, the utility restores the schema and table metadata only for the schemas and tables that are specified to be restored.

Database information that is not associated with a specific schema or table, such as roles and tablespaces, is not restored. You can specify the -G option with this option to restore global metadata that was backed up with the gpcrondump utility.

Database statistics are not restored. You can specify the --restore-stats option to restore statistics that were backed up with the gpcrondump utility.

Not supported with the --noplan or --noanalyze options.

--netbackup-block-size size
Specify the block size, in bytes, of data being transferred from the Veritas NetBackup server. The default is 512 bytes.

NetBackup options are not supported if DDBoost backup options are specified.

--netbackup-service-host netbackup_server
The NetBackup master server that Greenplum Database connects to when backing up to NetBackup. If you specify this option, you must specify the timestamp of the backup with the -t option.

The maximum length for the this parameter is 127 characters.

This option is not supported with any of the these options: -R, -s, -b, -L, or --ddboost.

NetBackup options are not supported if DDBoost backup options are specified.

--noanalyze
The ANALYZE command is not run after a successful restore. The default is to run the ANALYZE command on restored tables. This option is useful if running ANALYZE on tables in your database requires a significant amount of time.

If this option is specified, you should run ANALYZE manually on restored tables. Failure to run ANALYZE following a restore might result in poor database performance.

Not supported with the -m option.

--noplan
Restores only the data backed up during the incremental backup specified by the timestamp_key. No other data from the complete backup set are restored. The full backup set containing the incremental backup must be available.

If the timestamp_key specified with the -t option does not reference an incremental backup, an error is returned.

Not supported with the -m option.

--prefix prefix_string
If you specified the `gpcrondump` option `--prefix prefix_string` to create the backup, you must specify this option with the `prefix_string` when restoring the backup.

If you created a full backup of a set of tables with `gpcrondump` and specified a prefix, you can use `gpcrondump` with the options `--list-filter-tables` and `--prefix prefix_string` to list the tables that were included or excluded for the backup.

- `-q` (no screen output)
  Run in quiet mode. Command output is not displayed on the screen, but is still written to the log file.

- `-R hostname:path_to_dumpset`
  Allows you to provide a hostname and full path to a set of dump files. The host does not have to be in the Greenplum Database array of hosts, but must be accessible from the Greenplum master.

- `--redirect database_name`
  Specify the name of the database where the data is restored. Specify this option to restore data to a database that is different than the database specified during back up. If `database_name` does not exist, it is created.

- `--report-status-dir report_directory`
  Specifies the absolute path to the directory on the each Greenplum Database host (master and segment hosts) where `gpdbrestore` writes report status files for a restore operation. If `report_directory` does not exist or is not writable, `gpdbrestore` returns an error and stops.

  If this option is not specified and the `-u` option is specified, report status files are written to the location specified by the `-u` option if the `-u` location is writable. If the location specified by `-u` option is not writable, the report status files are written to segment data directories.

- `--restore-stats [include | only]`
  Specify this option to restore database statistics that were backed up with the `gpcrondump` utility option `--dump-stats`.

  - The keyword `include` restores the statistics that were backed up in addition to performing a restore. This is the default if no keyword is specified.
  - The keyword `only` restores only the statistics that were backed up. No other database objects or database data are restored.

  If this option is specified with other options that include or exclude tables or schemas to restore, the utility restores statistics only for the tables specified to be restored.

  If statistics would be restored for a table that does not exist in the database, the utility displays a warning. The statistics are not restored.

- `-s database_name`
  Looks for latest set of dump files for the given database name in the segment data directories `db_dumps` directory on the Greenplum Database array of hosts.

- `-S schema_name`
  Restore all the tables, views, indexes, functions, and sequences qualified by the specified schema from the backup. The `-S` option can be specified multiple times. System catalog schemas are not supported. The schema name must exist in the backup set of the database being restored. To replace the data in the schema tables with the data from backup, you can specify the `--truncate` option. The schema tables are truncated before the data is restored.

  To support functions that might be restored, all procedural languages that are in the backup are also restored even though they are not schema specific. External items such as shared libraries that are used by a language are not included in a backup.

  The `-S` option cannot be specified with the `--change-schema` option.
-t timestamp_key
   The 14 digit timestamp key that uniquely identifies a backup set of data to restore. It is of the form YYYYMMDDHHMMSS. Looks for dump files matching this timestamp key in the segment data directories db_dumps directory on the Greenplum Database array of hosts.

-T schema.table_name
   The name of a table to restore. The -T option can be specified multiple times. The named table(s) must exist in the backup set of the database being restored. Existing tables are not automatically truncated before data is restored from backup. To replace existing data in the table from backup, you can specify the --truncate option.
   Wildcard characters are not supported.

--table-file file_name
   Specify a file file_name that contains a list of table names to restore. The file contains any number of table names, listed one per line. See the -T option for information about restoring specific tables.

--truncate
   Truncate table data before restoring data to the table from the backup. If this option is not specified, existing table data is not removed before data is restored to the table.
   This option is supported only when restoring a set of tables with the option -S, -T or --table-file. If a table to be restored does not exist in the database, the table is restored and the utility returns a warning message stating that the table did not exist in the database.
   This option is not supported with the -e option.

-u backup_directory
   Specifies the absolute path to the directory containing the db_dumps directory on each host. If not specified, defaults to the data directory of each instance to be backed up.
   Specify this option if you specified a backup directory with the gpcrondump option -u when creating a backup set.
   If backup_directory is not writable, backup operation report status files are written to segment data directories. You can specify a different location where report status files are written with the --report-status-dir option.
   Note: This option is not supported if --ddboost is specified.

-v | --verbose
   Specifies verbose mode.

--version (show utility version)
   Displays the version of this utility.

-? (help)
   Displays the online help.

Examples

Restore the sales database from the latest backup files generated by gpcrondump (assumes backup files are in the segment data directories in db_dumps):

    gpdbrestore -s sales

Restore a database from backup files that reside on an archive host outside the Greenplum Database array (command issued on the Greenplum master host):

    gpdbrestore -R archivehostname:/data_pl/db_dumps/20160214
Restore global objects only (roles and tablespaces):

```
gpdbrestore -G
```

**Note:** The `--R` option is not supported when restoring a backup set that includes incremental backups.

If you restore from a backup set that contains an incremental backup, all the files in the backup set must be available to `gpdbrestore`. For example, the following timestamp keys specify a backup set. 20160514054532 is the full backup and the others are incremental.

```
20160514054532
20160714095512
20150914081205
20161114064330
20170114051246
```

The following `gpdbrestore` command specifies the timestamp key 20161114064330. The incremental backup with the timestamps 20160714095512 and 20160914081205 and the full backup must be available to perform a restore.

```
gpdbrestore -t 20161114064330
```

The following `gpdbrestore` command uses the `--noplan` option to restore only the data that was backed up during the incremental backup with the timestamp key 20161114064330. Data in the previous incremental backups and the data in the full backup are not restored.

```
gpdbrestore -t 20161114064330 --noplan
```

This `gpdbrestore` command restores Greenplum Database data from the data managed by NetBackup master server `nbu_server1`. The option `--netbackup-service-host=nbu_server1` specifies the timestamp generated by `gpcrondump` when the backup was created. The `--e` option specifies that the target database is dropped before it is restored.

```
gpdbrestore -t 20170530090000 -e --netbackup-service-host=nbu_server1
```

See Also

`gpcrondump`

**gpdeletesystem**

Deletes a Greenplum Database system that was initialized using `gpinitdb`.

**Synopsis**

```
gpdeletesystem [-d master_data_directory] [-B parallel_processes]
   [-f] [-l logfile_directory] [-D]
```

```
gpdeletesystem -?
```

```
gpdeletesystem -v
```

**Description**

The `gpdeletesystem` utility performs the following actions:

- Stop all `postgres` processes (the segment instances and master instance).
- Deletes all data directories.
Before running `gpdeletesystem`:

- Move any backup files out of the master and segment data directories.
- Make sure that Greenplum Database is running.
- If you are currently in a segment data directory, change directory to another location. The utility fails with an error when run from within a segment data directory.

This utility will not uninstall the Greenplum Database software.

**Options**

- **-d master_data_directory**
  Specifies the master host data directory. If this option is not specified, the setting for the environment variable `MASTER_DATA_DIRECTORY` is used. If this option is specified, it overrides any setting of `MASTER_DATA_DIRECTORY`. If `master_data_directory` cannot be determined, the utility returns an error.

- **-B parallel_processes**
  The number of segments to delete in parallel. If not specified, the utility will start up to 60 parallel processes depending on how many segment instances it needs to delete.

- **-f (force)**
  Force a delete even if backup files are found in the data directories. The default is to not delete Greenplum Database instances if backup files are present.

- **-l logfile_directory**
  The directory to write the log file. Defaults to `~/gpAdminLogs`.

- **-D (debug)**
  Sets logging level to debug.

- **-? (help)**
  Displays the online help.

- **-v (show utility version)**
  Displays the version, status, last updated date, and check sum of this utility.

**Examples**

Delete a Greenplum Database system:

```
gpdeletesystem -d /gpdata/gp-1
```

Delete a Greenplum Database system even if backup files are present:

```
gpdeletesystem -d /gpdata/gp-1 -f
```

**See Also**

`gpinitsystem`  
`gpexpand`

`gpexpand`  
Expands an existing Greenplum Database across new hosts in the array.

**Synopsis**

```
gpexpand [-f|--hosts-file] hosts_file
   | [-i|--input] input_file [-B batch_size] [-V|--novacuum]
```
Prerequisites

- You are logged in as the Greenplum Database superuser (gpadmin).
- The new segment hosts have been installed and configured as per the existing segment hosts. This involves:
  - Configuring the hardware and OS
  - Installing the Greenplum software
  - Creating the gpadmin user account
  - Exchanging SSH keys.
- Enough disk space on your segment hosts to temporarily hold a copy of your largest table.
- When redistributing data, Greenplum Database must be running in production mode. Greenplum Database cannot be restricted mode or in master mode. The gpstart options -R or -m cannot be specified to start Greenplum Database.

Description

The gpexpand utility performs system expansion in two phases: segment initialization and then table redistribution.

In the initialization phase, gpexpand runs with an input file that specifies data directories, dbid values, and other characteristics of the new segments. You can create the input file manually, or by following the prompts in an interactive interview.

If you choose to create the input file using the interactive interview, you can optionally specify a file containing a list of expansion hosts. If your platform or command shell limits the length of the list of hostnames that you can type when prompted in the interview, specifying the hosts with -f may be mandatory.

In addition to initializing the segments, the initialization phase performs these actions:
- Creates an expansion schema to store the status of the expansion operation, including detailed status for tables.
- Changes the distribution policy for all tables to DISTRIBUTED RANDOMLY. The original distribution policies are later restored in the redistribution phase.

To begin the redistribution phase, you must run gpexpand with either the -d (duration) or -e (end time) options. Until the specified end time or duration is reached, the utility will redistribute tables in the expansion schema. Each table is reorganized using ALTER TABLE commands to rebalance the tables across new segments, and to set tables to their original distribution policy. If gpexpand completes the reorganization of all tables before the specified duration, it displays a success message and ends.

Note: Data redistribution should be performed during low-use hours. Redistribution can divided into batches over an extended period.
Options
- **a | --analyze**
  Run **ANALYZE** to update the table statistics after expansion. The default is to not run **ANALYZE**.

- **B batch_size**
  Batch size of remote commands to send to a given host before making a one-second pause. Default is 16. Valid values are 1-128.

  The **gpexpand** utility issues a number of setup commands that may exceed the host’s maximum threshold for authenticated connections as defined by **MaxStartups** in the SSH daemon configuration. The one-second pause allows authentications to be completed before **gpexpand** issues any more commands.

  The default value does not normally need to be changed. However, it may be necessary to reduce the maximum number of commands if **gpexpand** fails with connection errors such as 'ssh_exchange_identification: Connection closed by remote host.'

- **c | --clean**
  Remove the expansion schema.

- **d | --duration hh:mm:ss**
  Duration of the expansion session from beginning to end.

- **D database_name**
  Specifies the database in which to create the expansion schema and tables. If this option is not given, the setting for the environment variable **PGDATABASE** is used. The database templates **template1** and **template0** cannot be used.

- **e | --end 'YYYY-MM-DD hh:mm:ss'**
  Ending date and time for the expansion session.

- **f | --hosts-file filename**
  Specifies the name of a file that contains a list of new hosts for system expansion. Each line of the file must contain a single host name.

  This file can contain hostnames with or without network interfaces specified. The **gpexpand** utility handles either case, adding interface numbers to end of the hostname if the original nodes are configured with multiple network interfaces.

  **Note:** The Greenplum Database segment host naming convention is **sdwN** where **sdw** is a prefix and **N** is an integer. For example, **sdw1, sdw2** and so on. For hosts with multiple interfaces, the convention is to append a dash (-) and number to the host name. For example, **sdw1-1 and sdw1-2** are the two interface names for host **sdw1**.

- **i | --input input_file**
  Specifies the name of the expansion configuration file, which contains one line for each segment to be added in the format of:

  `hostname:address:port:fselocation:dbid:content:preferred_role:replication_port`

  If your system has filesystems, **gpexpand** will expect a filesystem configuration file (`input_file_name.fs`) to exist in the same directory as your expansion configuration file. The filesystem configuration file is in the format of:

  `filesystemOrder=filesystem1_name:filesystem2_name: ...`  
  `dbid:/path/for/filespace1:/path/for/filespace2: ...`  
  `dbid:/path/for/filespace1:/path/for/filespace2: ...`  
  ` ...`

- **n parallel_processes**
The number of tables to redistribute simultaneously. Valid values are 1 - 96.

Each table redistribution process requires two database connections: one to alter the table, and another to update the table's status in the expansion schema. Before increasing \(-n\), check the current value of the server configuration parameter `max_connections` and make sure the maximum connection limit is not exceeded.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-r)</td>
<td>(--rollback) Roll back a failed expansion setup operation. If the rollback command fails, attempt again using the (-D) option to specify the database that contains the expansion schema for the operation that you want to roll back.</td>
</tr>
<tr>
<td>(-s)</td>
<td>(--silent) Runs in silent mode. Does not prompt for confirmation to proceed on warnings.</td>
</tr>
<tr>
<td>(-S)</td>
<td>(--simple-progress) If specified, the <code>gpexpand</code> utility records only the minimum progress information in the Greenplum Database table <code>gpexpand.expansion_progress</code>. The utility does not record the relation size information and status information in the table <code>gpexpand.status_detail</code>. Specifying this option can improve performance by reducing the amount of progress information written to the <code>gpexpand</code> tables.</td>
</tr>
<tr>
<td>([-t)</td>
<td>(--tardir) directory] directory The fully qualified path to a directory on segment hosts where the <code>gpexpand</code> utility copies a temporary tar file. The file contains Greenplum Database files that are used to create segment instances. The default directory is the user home directory.</td>
</tr>
<tr>
<td>(-v)</td>
<td>(--verbose) Verbose debugging output. With this option, the utility will output all DDL and DML used to expand the database.</td>
</tr>
<tr>
<td>(--version)</td>
<td>Display the utility's version number and exit.</td>
</tr>
<tr>
<td>(-V)</td>
<td>(--novacuum) Do not vacuum catalog tables before creating schema copy.</td>
</tr>
<tr>
<td>(-?)</td>
<td>(-h)</td>
</tr>
</tbody>
</table>

**Examples**

Run `gpexpand` with an input file to initialize new segments and create the expansion schema in the default database:

```
$ gpexpand -i input_file
```

Run `gpexpand` for sixty hours maximum duration to redistribute tables to new segments:

```
$ gpexpand -d 60:00:00
```

**See Also**

`gpfsh-exkeys`

`gpfdist`

Serves data files to or writes data files out from Greenplum Database segments.
Synopsis

```
[-S] [-w time] [-v] [-V] [-s] [-m max_length] 
[--ssl certificate_path] [--sslclean wait_time] [-c config.yml]
```

```
gpfdist -? | --help
```

```
gpfdist --version
```

Description

gpfdist is Greenplum Database parallel file distribution program. It is used by readable external tables and gpload to serve external table files to all Greenplum Database segments in parallel. It is used by writable external tables to accept output streams from Greenplum Database segments in parallel and write them out to a file.

In order for gpfdist to be used by an external table, the LOCATION clause of the external table definition must specify the external table data using the gpfdist:// protocol (see the Greenplum Database command CREATE EXTERNAL TABLE).

**Note:** If the --ssl option is specified to enable SSL security, create the external table with the gpfdists:// protocol.

The benefit of using gpfdist is that you are guaranteed maximum parallelism while reading from or writing to external tables, thereby offering the best performance as well as easier administration of external tables.

For readable external tables, gpfdist parses and serves data files evenly to all the segment instances in the Greenplum Database system when users SELECT from the external table. For writable external tables, gpfdist accepts parallel output streams from the segments when users INSERT into the external table, and writes to an output file.

For readable external tables, if load files are compressed using gzip or bzip2 (have a .gz or .bz2 file extension), gpfdist uncompresses the files automatically before loading provided that gunzip or bunzip2 is in your path.

**Note:** Currently, readable external tables do not support compression on Windows platforms, and writable external tables do not support compression on any platforms.

When reading or writing data with the gpfdist or gpfdists protocol, Greenplum Database includes X-GP-PROTO in the HTTP request header to indicate that the request is from Greenplum Database. The utility rejects HTTP requests that do not include X-GP-PROTO in the request header.

Most likely, you will want to run gpfdist on your ETL machines rather than the hosts where Greenplum Database is installed. To install gpfdist on another host, simply copy the utility over to that host and add gpfdist to your $PATH.

**Note:** When using IPv6, always enclose the numeric IP address in brackets.

Options

**-d directory**

The directory from which gpfdist will serve files for readable external tables or create output files for writable external tables. If not specified, defaults to the current directory.

**-l log_file**

The fully qualified path and log file name where standard output messages are to be logged.

**-p http_port**
The HTTP port on which `gpfdist` will serve files. Defaults to 8080.

- **t timeout**
  Sets the time allowed for Greenplum Database to establish a connection to a `gpfdist` process. Default is 5 seconds. Allowed values are 2 to 7200 seconds (2 hours). May need to be increased on systems with a lot of network traffic.

- **m max_length**
  Sets the maximum allowed data row length in bytes. Default is 32768. Should be used when user data includes very wide rows (or when `line too long` error message occurs). Should not be used otherwise as it increases resource allocation. Valid range is 32K to 256MB. (The upper limit is 1MB on Windows systems.)

  **Note:** Memory issues might occur if you specify a large maximum row length and run a large number of `gpfdist` concurrent connections. For example, setting this value to the maximum of 256MB with 96 concurrent `gpfdist` processes requires approximately 24GB of memory (`((96 + 1) \times 246MB)`).

- **-s**
  Enables simplified logging. When this option is specified, only messages with `WARN` level and higher are written to the `gpfdist` log file. `INFO` level messages are not written to the log file. If this option is not specified, all `gpfdist` messages are written to the log file.

  You can specify this option to reduce the information written to the log file.

- **-S (use O_SYNC)**
  Opens the file for synchronous I/O with the `O_SYNC` flag. Any writes to the resulting file descriptor block `gpfdist` until the data is physically written to the underlying hardware.

- **-w time**
  Sets the number of seconds that Greenplum Database delays before closing a target file such as a named pipe. The default value is 0, no delay. The maximum value is 7200 seconds (2 hours).

  For a Greenplum Database with multiple segments, there might be a delay between segments when writing data from different segments to the file. You can specify a time to wait before Greenplum Database closes the file to ensure all the data is written to the file.

- **--ssl certificate_path**
  Adds SSL encryption to data transferred with `gpfdist`. After executing `gpfdist` with the `--ssl` `certificate_path` option, the only way to load data from this file server is with the `gpfdist://` protocol. For information on the `gpfdist://` protocol, see “Loading and Unloading Data” in the Greenplum Database Administrator Guide.

  The location specified in `certificate_path` must contain the following files:
  - The server certificate file, `server.crt`
  - The server private key file, `server.key`
  - The trusted certificate authorities, `root.crt`

  The root directory (`/`) cannot be specified as `certificate_path`.

- **--sslclean wait_time**
  When the utility is run with the `--ssl` option, sets the number of seconds that the utility delays before closing an SSL session and cleaning up the SSL resources after it completes writing data to or from a Greenplum Database segment. The default value is 0, no delay. The maximum value is 500 seconds. If the delay is increased, the transfer speed decreases.
In some cases, this error might occur when copying large amounts of data: gpfdist server closed connection. To avoid the error, you can add a delay, for example --sslclean 5.

-c config.yaml
   Specifies rules that gpfdist uses to select a transform to apply when loading or extracting data. The gpfdist configuration file is a YAML 1.1 document.

For information about the file format, see Configuration File Format in the Greenplum Database Administrator Guide. For information about configuring data transformation with gpfdist, see Transforming XML Data in the Greenplum Database Administrator Guide.

This option is not available on Windows platforms.

-v (verbose)
   Verbose mode shows progress and status messages.

-V (very verbose)
   Verbose mode shows all output messages generated by this utility.

-? (help)
   Displays the online help.

--version
   Displays the version of this utility.

Examples
To serve files from a specified directory using port 8081 (and start gpfdist in the background):

```
gpfdist -d /var/load_files -p 8081 &
```

To start gpfdist in the background and redirect output and errors to a log file:

```
gpfdist -d /var/load_files -p 8081 -l /home/gpadmin/log &
```

To stop gpfdist when it is running in the background:
--First find its process id:

```
ps ax | grep gpfdist
```

--Then kill the process, for example:

```
kill 3456
```

See Also
gpload, CREATE EXTERNAL TABLE in the Greenplum Database Reference Guide

gpfilespace

Creates a filespace using a configuration file that defines per-segment file system locations. Filespaces describe the physical file system resources to be used by a tablespace.

Synopsis

```
gpfilespace [connection_option ...] [-l logfile_directory] [-o output_file_name]
```
**Description**

A tablespace requires a file system location to store its database files. In Greenplum Database, the master and each segment (primary and mirror) needs its own distinct storage location. This collection of file system locations for all components in a Greenplum system is referred to as a *filespace*. Once a filespace is defined, it can be used by one or more tablespaces.

When used with the `-o` option, the `gpfilespace` utility looks up your system configuration information in the Greenplum Database catalog tables and prompts you for the appropriate file system locations needed to create the filespace. It then outputs a configuration file that can be used to create a filespace. If a file name is not specified, a `gpfilespace_config_#` file will be created in the current directory by default.

Once you have a configuration file, you can run `gpfilespace` with the `-c` option to create the filespace in Greenplum Database.

You will need to create a filespace before you can use the `gpfilespace --movetempfilespace` or `--movetransfilespace` option to move your temporary or transaction files to the new location.

Use either `gpfilespace --showtempfilespace` or `--showtransfilespace` options to show the name of the filespace currently associated with temporary or transaction files.

**Note:** If segments are down due to a power or nic failure, you may see inconsistencies during filespace creation. You may not be able to bring up the Greenplum Database.

**Options**

- `-c` | `--config fs_config_file`
  A configuration file containing:
  - An initial line denoting the new filespace name. For example:
    
    ```
    filespace:myfs
    ```
  - One line each for the master, the primary segments, and the mirror segments. A line describes a file system location that a particular segment database instance should use as its data directory location to store database files associated with a tablespace. Each line is in the format of:
    
    ```
    hostname:dbid:/filesystem_dir/seg_datadir_name
    ```

- `-l` | `--logdir logfile_directory`
  The directory to write the log file. Defaults to `~/gpAdminLogs`.

- `-o` | `--output output_file_name`
  The directory location and file name to output the generated filespace configuration file. You will be prompted to enter a name for the filespace, a master file system location, the primary segment file system locations, and the mirror segment file system locations.
For example, if your configuration has 2 primary and 2 mirror segments per host, you will be prompted for a total of 5 locations (including the master). The file system locations must exist on all hosts in your system prior to running the `gpfilespace` utility. The utility will designate segment-specific data directories within the location(s) you specify, so it is possible to use the same location for multiple segments. However, primaries and mirrors cannot use the same location. After the utility creates the configuration file, you can manually edit the file to make any required changes to the filespace layout before creating the filespace in Greenplum Database.

```
--movetempfilespace {filespace_name | default}
    Moves temporary files to a new filespace or to the default location.

--movetransfilespace {filespace_name | default}
    Moves transaction files to a new filespace or to the default location.

--showtempfilespace
    Show the name of the filespace currently associated with temporary files. This option checks that all primary and mirror segments, master and master standby are using the same filespace or temporary files. You will receive a warning message and an email if any inconsistencies exist.

--showtransfilespace
    Show the name of the filespace currently associated with transaction files. This option checks that all primary and mirror segments, master and master standby are using the same filespace or transaction files. You will receive a warning message and an email if any inconsistencies exist.

-v | --version (show utility version)
    Displays the version of this utility.

-? | --help (help)
    Displays the utility usage and syntax.
```

**Connection Options**

```
-h host | --host host
    The host name of the machine on which the Greenplum master database server is running. If not specified, reads from the environment variable `PGHOST` or defaults to localhost.

-p port | --port port
    The TCP port on which the Greenplum master database server is listening for connections. If not specified, reads from the environment variable `PGPORT` or defaults to 5432.

-U username | --username superuser_name
    The database superuser role name to connect as. If not specified, reads from the environment variable `PGUSER` or defaults to the current system user name. Only database superusers are allowed to create filespaces.

-W | --password
    Force a password prompt.
```

**Examples**

Create a filespace configuration file. You will be prompted to enter a name for the filespace, a master file system location, the primary segment file system locations, and the mirror segment file system locations. For example, if your configuration has 2 primary and 2 mirror segments per host, you will be prompted for a total of 5 locations (including the master). The file system locations must exist on all hosts in your system prior to running the `gpfilespace` utility:

```
$ gpfilespace -o .
```
Enter a name for this filespace
> fastdisk

Checking your configuration:

Your system has 2 hosts with 2 primary and 2 mirror segments per host.

Configuring hosts: [sdw1, sdw2]

Please specify 2 locations for the primary segments, one per line:
primary location 1> /gp_pri_filespc
primary location 2> /gp_pri_filespc

Please specify 2 locations for the mirror segments, one per line:
mirror location 1> /gp_mir_filespc
mirror location 2> /gp_mir_filespc

Enter a file system location for the master:
master location> /gp_master_filespc

Example filespace configuration file:

```
filespace:fastdisk
mdw:1:/gp_master_filespc/gp-1
sdw1:2:/gp_pri_filespc/gp0
sdw1:3:/gp_mir_filespc/gp1
sdw2:4:/gp_mir_filespc/gp0
sdw2:5:/gp_pri_filespc/gp1
```

Execute the configuration file to create the filespace in Greenplum Database:

$ gpfilespace -c gpfilespace_config_1

See Also

CREATE TABLESPACE in the Greenplum Database Reference Guide

gpinitstandby

Adds and/or initializes a standby master host for a Greenplum Database system.

Synopsis

```
gpinitstandby { -s standby_hostname [-P port]
    [-F list_of_filespaces] | -r | -n }
    [-a] [-q] [-D] [-l logfile_directory]

   gpinitstandby -v
   gpinitstandby -?
```

Description

The *gpinitstandby* utility adds a backup, standby master host to your Greenplum Database system. If your system has an existing standby master host configured, use the `-r` option to remove it before adding the new standby master host.
Before running this utility, make sure that the Greenplum Database software is installed on the standby master host and that you have exchanged SSH keys between the hosts. It is recommended that the master port is set to the same port number on the master host and the backup master host.

This utility should be run on the currently active primary master host. See the Greenplum Database Installation Guide for instructions.

The utility performs the following steps:

- Updates the Greenplum Database system catalog to remove the existing standby master host information (if the `-r` option is supplied)
- Updates the Greenplum Database system catalog to add the new standby master host information
- Edits the `pg_hba.conf` file of the Greenplum Database master to allow access from the newly added standby master.
- Sets up the standby master instance on the alternate master host
- Starts the synchronization process

A backup, standby master host serves as a ‘warm standby’ in the event of the primary master host becoming non-operational. The standby master is kept up to date by transaction log replication processes (the `walsender` and `walreceiver`), which run on the primary master and standby master hosts and keep the data between the primary and standby master hosts synchronized. If the primary master fails, the log replication process is shut down, and the standby master can be activated in its place by using the `gpactivatestandby` utility. Upon activation of the standby master, the replicated logs are used to reconstruct the state of the master host at the time of the last successfully committed transaction.

The activated standby master effectively becomes the Greenplum Database master, accepting client connections on the master port and performing normal master operations such as SQL command processing and resource management.

**Important:** If the `gpinitstandby` utility previously failed to initialize the standby master, you must delete the files in the standby master data directory before running `gpinitstandby` again. The standby master data directory is not cleaned up after an initialization failure because it contains log files that can help in determining the reason for the failure.

If an initialization failure occurs, a summary report file is generated in the standby host directory `/tmp`. The report file lists the directories on standby host that require clean up.

**Options**

- `-a` *(do not prompt)*
  Do not prompt the user for confirmation.

- `-D` *(debug)*
  Sets logging level to debug.

- `-F list_of_filespaces`
  A list of filespace names and the associated locations. Each filespace name and its location is separated by a colon. If there is more than one file space name, each pair (name and location) is separated by a comma. For example:

  ```
  filespace1_name:fs1_location,filespace2_name:fs2_location
  ```

  If this option is not specified, `gpinitstandby` prompts the user for the filespace names and locations.

  If the list is not formatted correctly or number of filespaces do not match the number of filespaces already created in the system, `gpinitstandby` returns an error.

- `-l logfile_directory`
  The directory to write the log file. Defaults to `~/gpAdminLogs`.

- `-n` *(restart standby master)*
Specify this option to start a Greenplum Database standby master that has been configured but has stopped for some reason.

```
-P port
```

This option specifies the port that is used by the Greenplum Database standby master. The default is the same port used by the active Greenplum Database master.

If the Greenplum Database standby master is on the same host as the active master, the ports must be different. If the ports are the same for the active and standby master and the host is the same, the utility returns an error.

```
-q (no screen output)
```

Run in quiet mode. Command output is not displayed on the screen, but is still written to the log file.

```
-r (remove standby master)
```

Removes the currently configured standby master host from your Greenplum Database system.

```
-s standby_hostname
```

The host name of the standby master host.

```
-v (show utility version)
```

Displays the version, status, last updated date, and check sum of this utility.

```
-? (help)
```

Displays the online help.

**Examples**

Add a standby master host to your Greenplum Database system and start the synchronization process:

```
gpinitstandby -s host09
```

Start an existing standby master host and synchronize the data with the current primary master host:

```
gpinitstandby -n
```

**Note:** Do not specify the -n and -s options in the same command.

Add a standby master host to your Greenplum Database system specifying a different port:

```
gpinitstandby -s myhost -P 2222
```

If you specify the same host name as the active Greenplum Database master, the installed Greenplum Database software that is used as a standby master must be in a separate location from the active Greenplum Database master. Also, filesystem locations that are used by the standby master must be different than the filesystem locations used by the active Greenplum Database master.

Remove the existing standby master from your Greenplum system configuration:

```
gpinitstandby -r
```

**See Also**

`gpinitssystem, gpaddmirrors, gpactivatestandby`
gpinsистем

Initializes a Greenplum Database system using configuration parameters specified in the gpinsистем_config file.

Synopsis

gpinsистем -c  cluster_configuration_file
[-h  hostfile_gpinsистем] [-B  parallel_processes]
[-p  postgresql_conf_param_file]
[-s  standby_master_host  [-P  standby_master_port]
    [-F  standby_master_filespaces]]
[-m  number | --max_connections(number]
[-b  size | --shared_buffers=size]
[-n  locale | --locale=locale] [--lc-collate=locale]
[--lc-type=locale] [--lc-messages=locale]
[--lc-monetary=locale] [--lc-numeric=locale]
[--lc-time=locale] [-e  password | --su_password=password]
[-I  input_configuration_file]
[-O  output_configuration_file]

Description

The gpinsистем utility creates a Greenplum Database instance or writes an input configuration file using the values defined in a cluster configuration file and any command-line options that you provide. See Initialization Configuration File Format for more information about the configuration file. Before running this utility, make sure that you have installed the Greenplum Database software on all the hosts in the array.

With the -O  output_configuration_file option, gpinsистем does not create a new database instance but instead writes all provided configuration information to the specified output file. This file uses the QD_PRIMARY_ARRAY and PRIMARY_ARRAY parameters to define each member using its hostname, port, data directory, segment prefix, segment ID, and content ID. Details of the array configuration can be modified as necessary to match values available in a Greenplum Database backup, or can simply be used to recreate the same cluster configuration at a later time. Configuration files that use QD_PRIMARY_ARRAY and PRIMARY_ARRAY must be passed into gpinsистем using the -I  input_configuration_file option. See Initialization Configuration File Format for more information.

In a Greenplum Database DBMS, each database instance (the master and all segments) must be initialized across all of the hosts in the system in such a way that they can all work together as a unified DBMS. The gpinsистем utility takes care of initializing the Greenplum master and each segment instance, and configuring the system as a whole.

Before running gpinsистем, you must set the $GPHOME environment variable to point to the location of your Greenplum Database installation on the master host and exchange SSH keys between all host addresses in the array using gpssh-exkeys.

This utility performs the following tasks:

- Verifies that the parameters in the configuration file are correct.
- Ensures that a connection can be established to each host address. If a host address cannot be reached, the utility will exit.
- Verifies the locale settings.
- Displays the configuration that will be used and prompts the user for confirmation.
- Initializes the master instance.
- Initializes the standby master instance (if specified).
- Initializes the primary segment instances.
- Initializes the mirror segment instances (if mirroring is configured).
- Configures the Greenplum Database system and checks for errors.
- Starts the Greenplum Database system.

**Options**

- **-a**
  Do not prompt the user for confirmation.

- **-B parallel_processes**
  The number of segments to create in parallel. If not specified, the utility will start up to 4 parallel processes at a time.

- **-c cluster_configuration_file**
  Required. The full path and filename of the configuration file, which contains all of the defined parameters to configure and initialize a new Greenplum Database system. See *Initialization Configuration File Format* for a description of this file. You must provide either the **-c cluster_configuration_file** option or the **-I input_configuration_file** option to **gpinitsystem**.

- **-D**
  Sets log output level to debug.

- **-h hostfile_gpinitsystem**
  Optional. The full path and filename of a file that contains the host addresses of your segment hosts. If not specified on the command line, you can specify the host file using the MACHINE_LIST_FILE parameter in the **gpinitsystem_config** file.

- **-I input_configuration_file**
  The full path and filename of an input configuration file, which defines the Greenplum Database members and segments using the **QD_PRIMARY_ARRAY** and **PRIMARY_ARRAY** parameters. The input configuration file is typically created by using **gpinitsystem** with the **-O output_configuration_file** option. You must provide either the **-c cluster_configuration_file** option or the **-I input_configuration_file** option to **gpinitsystem**.

- **--locale=locale | -n locale**
  Sets the default locale used by Greenplum Database. If not specified, the LC_ALL, LC_COLLATE, or LANG environment variable of the master host determines the locale. If these are not set, the default locale is C (POSIX). A locale identifier consists of a language identifier and a region identifier, and optionally a character set encoding. For example, sv_SE is Swedish as spoken in Sweden, en_US is U.S. English, and fr_CA is French Canadian. If more than one character set can be useful for a locale, then the specifications look like this: en_US.UTF-8 (locale specification and character set encoding). On most systems, the command **locale** will show the locale environment settings and **locale -a** will show a list of all available locales.

- **--lc-collate=locale**
  Similar to **--locale**, but sets the locale used for collation (sorting data). The sort order cannot be changed after Greenplum Database is initialized, so it is important to choose a collation locale that is compatible with the character set encodings that you plan to use for your data. There is a special collation name of C or POSIX (byte-order sorting as opposed to dictionary-order sorting). The C collation can be used with any character encoding.

- **--lc-ctype=locale**
  Similar to **--locale**, but sets the locale used for character classification (what character sequences are valid and how they are interpreted). This cannot be changed after
Greenplum Database is initialized, so it is important to choose a character classification locale that is compatible with the data you plan to store in Greenplum Database.

--lc-messages=locale
Similar to --locale, but sets the locale used for messages output by Greenplum Database. The current version of Greenplum Database does not support multiple locales for output messages (all messages are in English), so changing this setting will not have any effect.

--lc-monetary=locale
Similar to --locale, but sets the locale used for formatting currency amounts.

--lc-numeric=locale
Similar to --locale, but sets the locale used for formatting numbers.

--lc-time=locale
Similar to --locale, but sets the locale used for formatting dates and times.

-l logfile_directory
The directory to write the log file. Defaults to ~/gpAdminLogs.

--max_connections=number | -m number
Sets the maximum number of client connections allowed to the master. The default is 250.

-O output_configuration_file
When used with the -O option, gpinitsystem does not create a new Greenplum Database cluster but instead writes the supplied cluster configuration information to the specified output_configuration_file. This file defines Greenplum Database members and segments using the QD_PRIMARY_ARRAY, PRIMARY_ARRAY, and MIRROR_ARRAY parameters, and can be later used with -I input_configuration_file to initialize a new cluster.

-p postgresql_conf_param_file
Optional. The name of a file that contains postgresql.conf parameter settings that you want to set for Greenplum Database. These settings will be used when the individual master and segment instances are initialized. You can also set parameters after initialization using the gpconfig utility.

-q
Run in quiet mode. Command output is not displayed on the screen, but is still written to the log file.

--shared_buffers=size | -b size
Sets the amount of memory a Greenplum server instance uses for shared memory buffers. You can specify sizing in kilobytes (kB), megabytes (MB) or gigabytes (GB). The default is 125MB.

-s standby_master_host
Optional. If you wish to configure a backup master host, specify the host name using this option. The Greenplum Database software must already be installed and configured on this host.

-P standby_master_port
Optional. If you configure a standby master host, specify its port number using this option. The Greenplum Database software must already be installed and configured on this host.

-F standby_master_filespaces
Optional. If you configure a standby master host, specify a list of filespace names and the associated locations using this option. Each filespace name and its location is separated
by a colon. If there is more than one file space name, each pair (name and location) is separated by a comma. For example:

```
filespace1_name:fs1_location,filespace2_name:fs2_location
```

If this option is not specified, `gpinitstandby` prompts the user for the file space names and locations.

If the list is not formatted correctly or the number of file spaces does not match the number of file spaces already created in the system, `gpinitstandby` returns an error.

```
--su_password=superuser_password | -e superuser_password
```

Use this option to specify the password to set for the Greenplum Database superuser account (such as `gpadmin`). If this option is not specified, the default password is assigned to the superuser account. You can use the `ALTER ROLE` command to change the password at a later time.

Recommended security best practices:

- Do not use the default password option for production environments.
- Change the password immediately after installation.

```
-S
```

If mirroring parameters are specified, spreads the mirror segments across the available hosts. The default is to group the set of mirror segments together on an alternate host from their primary segment set. Mirror spreading places each mirror on a different host within the Greenplum Database array. Spreading is only allowed if the number of hosts is greater than the number of segment instances.

```
-v
```

Displays the version of this utility.

```
-h
```

Displays the online help.

**Initialization Configuration File Format**

`gpinitsystem` requires a cluster configuration file with the following parameters defined. An example initialization configuration file can be found in `$GPHOME/docs/cli_help/gpconfigs/gpinitsystem_config`.

To avoid port conflicts between Greenplum Database and other applications, the Greenplum Database port numbers should not be in the range specified by the operating system parameter `net.ipv4.ip_local_port_range`. For example, if `net.ipv4.ip_local_port_range = 10000 65535`, you could set Greenplum Database base port numbers to these values.

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

**ARRAY_NAME**

Required. A name for the array you are configuring. You can use any name you like. Enclose the name in quotes if the name contains spaces.

**MACHINE_LIST_FILE**

Optional. Can be used in place of the `-h` option. This specifies the file that contains the list of segment host address names that comprise the Greenplum Database system. The master host is assumed to be the host from which you are running the utility and should
not be included in this file. If your segment hosts have multiple network interfaces, then this file would include all addresses for the host. Give the absolute path to the file.

**SEG_PREFIX**

*Required.* This specifies a prefix that will be used to name the data directories on the master and segment instances. The naming convention for data directories in a Greenplum Database system is `SEG_PREFIXnumber` where `number` starts with 0 for segment instances (the master is always -1). So for example, if you choose the prefix `gpseg`, your master instance data directory would be named `gpseg-1`, and the segment instances would be named `gpseg0`, `gpseg1`, `gpseg2`, `gpseg3`, and so on.

**PORT_BASE**

*Required.* This specifies the base number by which primary segment port numbers are calculated. The first primary segment port on a host is set as `PORT_BASE`, and then incremented by one for each additional primary segment on that host. Valid values range from 1 through 65535.

**DATA DIRECTORY**

*Required.* This specifies the data storage location(s) where the utility will create the primary segment data directories. The number of locations in the list dictate the number of primary segments that will get created per physical host (if multiple addresses for a host are listed in the host file, the number of segments will be spread evenly across the specified interface addresses). It is OK to list the same data storage area multiple times if you want your data directories created in the same location. The user who runs `gpinitsystem` (for example, the `gpadmin` user) must have permission to write to these directories. For example, this will create six primary segments per host:

```bash
declare -a DATA_DIRECTORY=(/data1/primary /data1/primary /data1/primary /data2/primary /data2/primary /data2/primary)
```

**MASTER HOSTNAME**

*Required.* The host name of the master instance. This host name must exactly match the configured host name of the machine (run the `hostname` command to determine the correct hostname).

**MASTER DIRECTORY**

*Required.* This specifies the location where the data directory will be created on the master host. You must make sure that the user who runs `gpinitsystem` (for example, the `gpadmin` user) has permissions to write to this directory.

**MASTER PORT**

*Required.* The port number for the master instance. This is the port number that users and client connections will use when accessing the Greenplum Database system.

**TRUSTED SHELL**

*Required.* The shell the `gpinitsystem` utility uses to execute commands on remote hosts. Allowed values are `ssh`. You must set up your trusted host environment before running the `gpinitsystem` utility (you can use `gpssh-exkeys` to do this).

**CHECK_POINT_SEGMENTS**

*Required.* Maximum distance between automatic write ahead log (WAL) checkpoints, in log file segments (each segment is normally 16 megabytes). This will set the `checkpoint_segments` parameter in the `postgresql.conf` file for each segment instance in the Greenplum Database system.

**ENCODING**

*Required.* The character set encoding to use. This character set must be compatible with the `--locale` settings used, especially `--lc-collate` and `--lc-ctype`. Greenplum Database supports the same character sets as PostgreSQL.
DATABASE_NAME

Optional. The name of a Greenplum Database database to create after the system is initialized. You can always create a database later using the `CREATE DATABASE` command or the `createdb` utility.

MIRROR_PORT_BASE

Optional. This specifies the base number by which mirror segment port numbers are calculated. The first mirror segment port on a host is set as MIRROR_PORT_BASE, and then incremented by one for each additional mirror segment on that host. Valid values range from 1 through 65535 and cannot conflict with the ports calculated by PORT_BASE.

REPLICATION_PORT_BASE

Optional. This specifies the base number by which the port numbers for the primary file replication process are calculated. The first replication port on a host is set as REPLICATION_PORT_BASE, and then incremented by one for each additional primary segment on that host. Valid values range from 1 through 65535 and cannot conflict with the ports calculated by PORT_BASE or MIRROR_PORT_BASE.

MIRROR_REPLICATION_PORT_BASE

Optional. This specifies the base number by which the port numbers for the mirror file replication process are calculated. The first mirror replication port on a host is set as MIRROR_REPLICATION_PORT_BASE, and then incremented by one for each additional mirror segment on that host. Valid values range from 1 through 65535 and cannot conflict with the ports calculated by PORT_BASE, MIRROR_PORT_BASE, or REPLICATION_PORT_BASE.

MIRROR_DATA_DIRECTORY

Optional. This specifies the data storage location(s) where the utility will create the mirror segment data directories. There must be the same number of data directories declared for mirror segment instances as for primary segment instances (see the DATA_DIRECTORY parameter). The user who runs gpinitsystem (for example, the gpadmin user) must have permission to write to these directories. For example:

```bash
declare -a MIRROR_DATA_DIRECTORY=(/data1/mirror /data1/mirror /data1/mirror /data2/mirror /data2/mirror /data2/mirror)
```

QD_PRIMARY_ARRAY, PRIMARY_ARRAY, MIRROR_ARRAY

These parameters can only be provided using an input configuration file, with the gpinitsystem -I input_configuration_file option. QD_PRIMARY_ARRAY, PRIMARY_ARRAY, and MIRROR_ARRAY define the Greenplum Database master host and the primary and mirror instances on the segment hosts, respectively, using the format:

```
host~port~data_directory/seg_prefix<segment_id>~dbid~content_id~replication_port
```

The Greenplum Database master always uses the value -1 for the segment ID and content ID. For example:

```bash
QD_PRIMARY_ARRAY=127.0.0.1~5432~/gpmaster/gpsne-1~1~-1~0
declare -a PRIMARY_ARRAY=
( 127.0.0.1~40000~/gpdata1/gpsne0~2~0~6000
127.0.0.1~40001~/gpdata2/gpsne1~3~1~6001
)
declare -a MIRROR_ARRAY=
( 127.0.0.1~50000~/gpmirror1/gpsne0~4~0~51000
127.0.0.1~50001~/gpmirror2/gpsne1~5~1~51001
)
You can use the `gpinitsystem -O output_configuration_file` to populate `QD_PRIMARY_ARRAY, PRIMARY_ARRAY, MIRROR_ARRAY` using the hosts, data directories, segment prefix, and base port values that you provide to the command. For recovery purposes, you can edit the segment and content IDs to match the values of an existing Greenplum Database backup.

**HEAP_CHECKSUM**

Optional. This parameter specifies if checksums are enabled for heap data. When enabled, checksums are calculated for heap storage in all databases, enabling Greenplum Database to detect corruption in the I/O system. This option is set when the system is initialized and cannot be changed later.

The `HEAP_CHECKSUM` option is on by default and turning it off is strongly discouraged. If you set this option to off, data corruption in storage can go undetected and make recovery much more difficult.

To determine if heap checksums are enabled in a Greenplum Database system, you can query the `data_checkums` server configuration parameter with the `gpconfig` management utility:

```
$ gpconfig -s data_checkums
```

**Examples**

Initialize a Greenplum Database array by supplying a cluster configuration file and a segment host address file, and set up a spread mirroring (`-S`) configuration:

```
$ gpinitsystem -c gpinitsystem_config -h hostfile_gpinitsystem -S
```

Initialize a Greenplum Database array and set the superuser remote password:

```
$ gpinitsystem -c gpinitsystem_config -h hostfile_gpinitsystem --su-password=mypassword
```

Initialize a Greenplum Database array with an optional standby master host:

```
$ gpinitsystem -c gpinitsystem_config -h hostfile_gpinitsystem -s host09
```

Instead of initializing a Greenplum Database array, write the provided configuration to an output file. The output file uses the `QD_PRIMARY_ARRAY` and `PRIMARY_ARRAY` parameters to define master and segment hosts:

```
$ gpinitsystem -c gpinitsystem_config -h hostfile_gpinitsystem -S -O cluster_init.config
```

Initialize a Greenplum Database using an input configuration file (a file that defines the Greenplum Database array using `QD_PRIMARY_ARRAY` and `PRIMARY_ARRAY` parameters:

```
$ gpinitsystem -I cluster_init.config
```

**See Also**

`gpssh-exkeys, gpdeletesystem`
**gpload**

Runs a load job as defined in a YAML formatted control file.

**Synopsis**

```
   [---no_auto_trans] [([v | --V] [-q]) [-D]]
gpload -?
gpload --version
```

**Prerequisites**

The client machine where `gpload` is executed must have the following:

- Python 2.6.2 or later, `pygresql` (the Python interface to PostgreSQL), and `pyyaml`. Note that Python and the required Python libraries are included with the Greenplum Database server installation, so if you have Greenplum Database installed on the machine where `gpload` is running, you do not need a separate Python installation.

  **Note:** Greenplum Database Loaders for Windows supports only Python 2.5 (available from [https://www.python.org](https://www.python.org)).

- The `gpfdist` parallel file distribution program installed and in your `$PATH`. This program is located in `$GPHOME/bin` of your Greenplum Database server installation.

- Network access to and from all hosts in your Greenplum Database array (master and segments).

- Network access to and from the hosts where the data to be loaded resides (ETL servers).

**Description**

`gpload` is a data loading utility that acts as an interface to the Greenplum Database external table parallel loading feature. Using a load specification defined in a YAML formatted control file, `gpload` executes a load by invoking the Greenplum Database parallel file server (`gpfdist`), creating an external table definition based on the source data defined, and executing an INSERT, UPDATE or MERGE operation to load the source data into the target table in the database.

The operation, including any SQL commands specified in the SQL collection of the YAML control file (see Control File Format), are performed as a single transaction to prevent inconsistent data when performing multiple, simultaneous load operations on a target table.

**Options**

- `-f control_file`
  
  Required. A YAML file that contains the load specification details. See Control File Format.

- `--gpfdist_timeout seconds`
  
  Sets the timeout for the `gpfdist` parallel file distribution program to send a response. Enter a value from 0 to 30 seconds (entering "0" to disables timeouts). Note that you might need to increase this value when operating on high-traffic networks.

- `-l log_file`
  
  Specifies where to write the log file. Defaults to `~/gpAdminLogs/gpload_YYYYMMDD`. For more information about the log file, see Log File Format.

- `--no_auto_trans`
Specify `--no_auto_trans` to disable processing the load operation as a single transaction if you are performing a single load operation on the target table.

By default, `gpload` processes each load operation as a single transaction to prevent inconsistent data when performing multiple, simultaneous operations on a target table.

- `-q` *(no screen output)*
  Run in quiet mode. Command output is not displayed on the screen, but is still written to the log file.

- `-D` *(debug mode)*
  Check for error conditions, but do not execute the load.

- `-v` *(verbose mode)*
  Show verbose output of the load steps as they are executed.

- `-V` *(very verbose mode)*
  Shows very verbose output.

- `--version`
  Show the version of this utility, then exit.

**Connection Options**

- `-d` *database*
  The database to load into. If not specified, reads from the load control file, the environment variable `$PGDATABASE` or defaults to the current system user name.

- `-h` *hostname*
  Specifies the host name of the machine on which the Greenplum Database master database server is running. If not specified, reads from the load control file, the environment variable `$PGHOST` or defaults to localhost.

- `-p` *port*
  Specifies the TCP port on which the Greenplum Database master database server is listening for connections. If not specified, reads from the load control file, the environment variable `$PGPORT` or defaults to 5432.

- `-U` *username*
  The database role name to connect as. If not specified, reads from the load control file, the environment variable `$PGUSER` or defaults to the current system user name.

- `-W` *(force password prompt)*
  Force a password prompt. If not specified, reads the password from the environment variable `$PGPASSWORD` or from a password file specified by `$PGPASSFILE` or in ~/.pgpass. If these are not set, then `gpload` will prompt for a password even if `-W` is not supplied.

**Control File Format**

The `gpload` control file uses the YAML 1.1 document format and then implements its own schema for defining the various steps of a Greenplum Database load operation. The control file must be a valid YAML document.

The `gpload` program processes the control file document in order and uses indentation (spaces) to determine the document hierarchy and the relationships of the sections to one another. The use of white space is significant. White space should not be used simply for formatting purposes, and tabs should not be used at all.
The basic structure of a load control file is:

```plaintext
---
VERSION: 1.0.0.1
DATABASE: db_name
USER: db_username
HOST: master_hostname
PORT: master_port
GPLOAD:
  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
SQL:
  - BEFORE: "sql_command"
  - AFTER: "sql_command"
```

**VERSION**

Optional. The version of the `gpload` control file schema. The current version is 1.0.0.1.

**DATABASE**

Optional. Specifies which database in the Greenplum Database system to connect to. If not specified, defaults to `$PGDATABASE` if set or the current system user name. You can also specify the database on the command line using the `-d` option.

**USER**
Optional. Specifies which database role to use to connect. If not specified, defaults to the current user or $PGUSER if set. You can also specify the database role on the command line using the \-U option.

If the user running \texttt{gpload} is not a Greenplum Database superuser, then the appropriate rights must be granted to the user for the load to be processed. See the \textit{Greenplum Database Reference Guide} for more information.

\textbf{HOST}

Optional. Specifies Greenplum Database master host name. If not specified, defaults to localhost or $PGHOST if set. You can also specify the master host name on the command line using the \-h option.

\textbf{PORT}

Optional. Specifies Greenplum Database master port. If not specified, defaults to 5432 or $PGPORT if set. You can also specify the master port on the command line using the \-p option.

\textbf{GPLOAD}

Required. Begins the load specification section. A \texttt{GPLOAD} specification must have an \texttt{INPUT} and an \texttt{OUTPUT} section defined.

\textbf{INPUT}

Required. Defines the location and the format of the input data to be loaded. \texttt{gpload} will start one or more instances of the \texttt{gpfdist} file distribution program on the current host and create the required external table definition(s) in Greenplum Database that point to the source data. Note that the host from which you run \texttt{gpload} must be accessible over the network by all Greenplum Database hosts (master and segments).

\textbf{SOURCE}

Required. The \texttt{SOURCE} block of an \texttt{INPUT} specification defines the location of a source file. An \texttt{INPUT} section can have more than one \texttt{SOURCE} block defined. Each \texttt{SOURCE} block defined corresponds to one instance of the \texttt{gpfdist} file distribution program that will be started on the local machine. Each \texttt{SOURCE} block defined must have a \texttt{FILE} specification.

For more information about using the \texttt{gpfdist} parallel file server and single and multiple \texttt{gpfdist} instances, see "Loading and Unloading Data" in the \textit{Greenplum Database Administrator Guide}.

\textbf{LOCAL\_HOSTNAME}

Optional. Specifies the host name or IP address of the local machine on which \texttt{gpload} is running. If this machine is configured with multiple network interface cards (NICs), you can specify the host name or IP of each individual NIC to allow network traffic to use all NICs simultaneously. The default is to use the local machine’s primary host name or IP only.

\textbf{PORT}

Optional. Specifies the specific port number that the \texttt{gpfdist} file distribution program should use. You can also supply a \texttt{PORT\_RANGE} to select an available port from the specified range. If both \texttt{PORT} and \texttt{PORT\_RANGE} are defined, then \texttt{PORT} takes precedence. If neither \texttt{PORT} or \texttt{PORT\_RANGE} are defined, the default is to select an available port between 8000 and 9000.

If multiple host names are declared in \texttt{LOCAL\_HOSTNAME}, this port number is used for all hosts. This configuration is desired if you want to use all NICs to load the same file or set of files in a given directory location.

\textbf{PORT\_RANGE}

Optional. Can be used instead of \texttt{PORT} to supply a range of port numbers from which \texttt{gpload} can choose an available port for this instance of the \texttt{gpfdist} file distribution program.

\textbf{FILE}
Required. Specifies the location of a file, named pipe, or directory location on the local file system that contains data to be loaded. You can declare more than one file so long as the data is of the same format in all files specified.

If the files are compressed using gzip or bzip2 (have a .gz or .bz2 file extension), the files will be uncompressed automatically (provided that gunzip or bunzip2 is in your path).

When specifying which source files to load, you can use the wildcard character (*) or other C-style pattern matching to denote multiple files. The files specified are assumed to be relative to the current directory from which gpload is executed (or you can declare an absolute path).

SSL
Optional. Specifies usage of SSL encryption. If SSL is set to true, gpload starts the gpfdist server with the --ssl option and uses the gpfdists:// protocol.

CERTIFICATES_PATH
Required when SSL is true; cannot be specified when SSL is false or unspecified. The location specified in CERTIFICATES_PATH must contain the following files:

- The server certificate file, server.crt
- The server private key file, server.key
- The trusted certificate authorities, root.crt

The root directory (/) cannot be specified as CERTIFICATES_PATH.

FULLY_QUALIFIED_DOMAIN_NAME
Optional. Specifies whether gpload resolve hostnames to the fully qualified domain name (FQDN) or the local hostname. If the value is set to true, names are resolved to the FQDN. If the value is set to false, resolution is to the local hostname. The default is false.

A fully qualified domain name might be required in some situations. For example, if the Greenplum Database system is in a different domain than an ETL application that is being accessed by gpload.

COLUMNS
Optional. Specifies the schema of the source data file(s) in the format of

field_name: data_type

The DELIMITER character in the source file is what separates two data value fields (columns). A row is determined by a line feed character (0x0a).

If the input COLUMNS are not specified, then the schema of the output TABLE is implied, meaning that the source data must have the same column order, number of columns, and data format as the target table.

The default source-to-target mapping is based on a match of column names as defined in this section and the column names in the target TABLE. This default mapping can be overridden using the MAPPING section.

TRANSFORM
Optional. Specifies the name of the input transformation passed to gpload. For information about XML transformations, see "Loading and Unloading Data" in the Greenplum Database Administrator Guide.

TRANSFORM_CONFIG
Required when TRANSFORM is specified. Specifies the location of the transformation configuration file that is specified in the TRANSFORM parameter, above.

MAX_LINE_LENGTH
Optional. An integer that specifies the maximum length of a line in the XML transformation data passed to `gpload`.

**FORMAT**

Optional. Specifies the format of the source data file(s) - either plain text (`TEXT`) or comma separated values (`CSV`) format. Defaults to `TEXT` if not specified. For more information about the format of the source data, see "Loading and Unloading Data" in the *Greenplum Database Administrator Guide*.

**DELIMITER**

Optional. Specifies a single ASCII character that separates columns within each row (line) of data. The default is a tab character in TEXT mode, a comma in CSV mode. You can also specify a non-printable ASCII character or a non-printable unicode character, for example: `\x1B` or `\u001B`. The escape string syntax, `E'character-code'`, is also supported for non-printable characters. The ASCII or unicode character must be enclosed in single quotes. For example: `E'\x1B'` or `E'\u001B'`.

**ESCAPE**

Specifies the single character that is used for C escape sequences (such as `\n`, `\t`, `\100`, and so on) and for escaping data characters that might otherwise be taken as row or column delimiters. Make sure to choose an escape character that is not used anywhere in your actual column data. The default escape character is a `\` (backslash) for text-formatted files and a `"` (double quote) for csv-formatted files, however it is possible to specify another character to represent an escape. It is also possible to disable escaping in text-formatted files by specifying the value 'OFF' as the escape value. This is very useful for data such as text-formatted web log data that has many embedded backslashes that are not intended to be escapes.

**NULL_AS**

Optional. Specifies the string that represents a null value. The default is `\N` (backslash-N) in TEXT mode, and an empty value with no quotations in CSV mode. You might prefer an empty string even in TEXT mode for cases where you do not want to distinguish nulls from empty strings. Any source data item that matches this string will be considered a null value.

**FORCE_NOT_NULL**

Optional. In CSV mode, processes each specified column as though it were quoted and hence not a NULL value. For the default null string in CSV mode (nothing between two delimiters), this causes missing values to be evaluated as zero-length strings.

**QUOTE**

Required when `FORMAT` is CSV. Specifies the quotation character for CSV mode. The default is double-quote (`"`).

**HEADER**

Optional. Specifies that the first line in the data file(s) is a header row (contains the names of the columns) and should not be included as data to be loaded. If using multiple data source files, all files must have a header row. The default is to assume that the input files do not have a header row.

**ENCODING**

Optional. Character set encoding of the source data. Specify a string constant (such as `'SQL_ASCII'`), an integer encoding number, or `DEFAULT` to use the default client encoding. If not specified, the default client encoding is used. For information about supported character sets, see the *Greenplum Database Reference Guide*.

**ERROR_LIMIT**

Optional. Enables single row error isolation mode for this load operation. When enabled, input rows that have format errors will be discarded provided that the error limit count is
not reached on any Greenplum Database segment instance during input processing. If the error limit is not reached, all good rows will be loaded and any error rows will either be discarded or captured as part of error log information. The default is to abort the load operation on the first error encountered. Note that single row error isolation only applies to data rows with format errors; for example, extra or missing attributes, attributes of a wrong data type, or invalid client encoding sequences. Constraint errors, such as primary key violations, will still cause the load operation to abort if encountered. For information about handling load errors, see "Loading and Unloading Data" in the Greenplum Database Administrator Guide.

**LOG_ERRORS**

Optional when ERROR_LIMIT is declared. Value is either true or false. The default value is false. If the value is true, rows with formatting errors are logged internally when running in single row error isolation mode. You can examine formatting errors with the Greenplum Database built-in SQL function `gp_read_error_log('table_name')`. If formatting errors are detected when loading data, `gpload` generates a warning message with the name of the table that contains the error information similar to this message.

```
timestamp|WARN|1 bad row, please use GPDB built-in function gp_read_error_log('table-name') to access the detailed error row
```

If LOG_ERRORS: true is specified, REUSE_TABLES: true must be specified to retain the formatting errors in Greenplum Database error logs. If REUSE_TABLES: true is not specified, the error information is deleted after the `gpload` operation. Only summary information about formatting errors is returned. You can delete the formatting errors from the error logs with the Greenplum Database function `gp_truncate_error_log()`.

For more information about handling load errors, see "Loading and Unloading Data" in the Greenplum Database Administrator Guide. For information about the `gp_read_error_log()` function, see the CREATE EXTERNAL TABLE command in the Greenplum Database Reference Guide.

**EXTERNAL**

Optional. Defines the schema of the external table database objects created by `gpload`. The default is to use the Greenplum Database search_path.

**SCHEMA**

Required when EXTERNAL is declared. The name of the schema of the external table. If the schema does not exist, an error is returned.

If % (percent character) is specified, the schema of the table name specified by TABLE in the OUTPUT section is used. If the table name does not specify a schema, the default schema is used.

**OUTPUT**

Required. Defines the target table and final data column values that are to be loaded into the database.

**TABLE**

Required. The name of the target table to load into.

**MODE**

Optional. Defaults to INSERT if not specified. There are three available load modes:

- INSERT - Loads data into the target table using the following method:

  ```sql
  INSERT INTO target_table SELECT * FROM input_data;
  ```
**UPDATE** - Updates the **UPDATE_COLUMNS** of the target table where the rows have **MATCH_COLUMNS** attribute values equal to those of the input data, and the optional **UPDATE_CONDITION** is true.

**MERGE** - Inserts new rows and updates the **UPDATE_COLUMNS** of existing rows where **FOOBAR** attribute values are equal to those of the input data, and the optional **MATCH_COLUMNS** is true. New rows are identified when the **MATCH_COLUMNS** value in the source data does not have a corresponding value in the existing data of the target table. In those cases, the entire row from the source file is inserted, not only the **MATCH** and **UPDATE columns**. If there are multiple new **MATCH_COLUMNS** values that are the same, only one new row for that value will be inserted. Use **UPDATE_CONDITION** to filter out the rows to discard.

**MATCH_COLUMNS**
Required if **MODE** is **UPDATE** or **MERGE**. Specifies the column(s) to use as the join condition for the update. The attribute value in the specified target column(s) must be equal to that of the corresponding source data column(s) in order for the row to be updated in the target table.

**UPDATE_COLUMNS**
Required if **MODE** is **UPDATE** or **MERGE**. Specifies the column(s) to update for the rows that meet the **MATCH_COLUMNS** criteria and the optional **UPDATE_CONDITION**.

**UPDATE_CONDITION**
Optional. Specifies a Boolean condition (similar to what you would declare in a **WHERE** clause) that must be met in order for a row in the target table to be updated (or inserted in the case of a **MERGE**).

**MAPPING**
Optional. If a mapping is specified, it overrides the default source-to-target column mapping. The default source-to-target mapping is based on a match of column names as defined in the source **COLUMNS** section and the column names of the target **TABLE**. A mapping is specified as either:

```
target_column_name: source_column_name
```

or
```
target_column_name: 'expression'
```

Where **expression** is any expression that you would specify in the **SELECT** list of a query, such as a constant value, a column reference, an operator invocation, a function call, and so on.

**PRELOAD**
Optional. Specifies operations to run prior to the load operation. Right now the only preload operation is **TRUNCATE**.

**TRUNCATE**
Optional. If set to true, **gpload** will remove all rows in the target table prior to loading it.

**REUSE_TABLES**
Optional. If set to true, **gpload** will not drop the external table objects and staging table objects it creates. These objects will be reused for future load operations that use the same load specifications. This improves performance of trickle loads (ongoing small loads to the same target table).

If **LOG_ERRORS: true** is specified, **REUSE_TABLES: true** must be specified to retain the formatting errors in Greenplum Database error logs. If **REUSE_TABLES: true** is not specified, formatting error information is deleted after the **gpload** operation.

**SQL**
Optional. Defines SQL commands to run before and/or after the load operation. You can specify multiple BEFORE and/or AFTER commands. List commands in the order of desired execution.

**BEFORE**

Optional. An SQL command to run before the load operation starts. Enclose commands in quotes.

**AFTER**

Optional. An SQL command to run after the load operation completes. Enclose commands in quotes.

### Log File Format

Log files output by `gpload` have the following format:

```
timestamp|level|message
```

Where `timestamp` takes the form: `YYYY-MM-DD HH:MM:SS`, `level` is one of `DEBUG`, `LOG`, `INFO`, `ERROR`, and `message` is a normal text message.

Some `INFO` messages that may be of interest in the log files are (where `#` corresponds to the actual number of seconds, units of data, or failed rows):

```
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
```

### Notes

If your database object names were created using a double-quoted identifier (delimited identifier), you must specify the delimited name within single quotes in the `gpload` control file. For example, if you create a table as follows:

```
CREATE TABLE "MyTable" ("MyColumn" text);
```

Your YAML-formatted `gpload` control file would refer to the above table and column names as follows:

```
- COLUMNS:
  - "MyColumn": text
OUTPUT:
  - TABLE: public."MyTable"
```

### Examples

Run a load job as defined in `my_load.yml`:

```
gpload -f my_load.yml
```

Example load control file:

```
---
VERSION: 1.0.0.1
DATABASE: ops
USER: gpadmin
HOST: mdw-1
```
PORT: 5432
GPLOAD:
  INPUT:
    - SOURCE:
      LOCAL_HOSTNAME:
        - etl1-1
        - etl1-2
        - etl1-3
        - etl1-4
    PORT: 8081
    FILE:
      - /var/load/data/*
    COLUMNS:
      - name: text
      - amount: float4
      - category: text
      - desc: text
      - date: date
    FORMAT: text
    DELIMITER: '|'
    ERROR_LIMIT: 25
    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)"

See Also

gpfdist, CREATE EXTERNAL TABLE in the Greenplum Database Reference Guide

gplogfilter

Searches through Greenplum Database log files for specified entries.

Synopsis

gplogfilter [timestamp_options] [pattern_options]
 [output_options] [input_options] [input_file]
gplogfilter --help
 gplogfilter --version

Description

The gplogfilter utility can be used to search through a Greenplum Database log file for entries matching the specified criteria. If an input file is not supplied, then gplogfilter will use the $MASTER_DATA_DIRECTORY environment variable to locate the Greenplum master log file in the standard logging location. To read from standard input, use a dash (-) as the input file name. Input files may be compressed using gzip. In an input file, a log entry is identified by its timestamp in YYYY-MM-DD [hh:mm[:ss]] format.
You can also use `gplogfilter` to search through all segment log files at once by running it through the `gpssh` utility. For example, to display the last three lines of each segment log file:

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

By default, the output of `gplogfilter` is sent to standard output. Use the `-o` option to send the output to a file or a directory. If you supply an output file name ending in `.gz`, the output file will be compressed by default using maximum compression. If the output destination is a directory, the output file is given the same name as the input file.

**Options**

**Timestamp Options**

- `-b` | `--begin=datetime`
  Specifies a starting date and time to begin searching for log entries in the format of `YYYY-MM-DD [hh:mm[:ss]]`.
  If a time is specified, the date and time must be enclosed in either single or double quotes.
  This example encloses the date and time in single quotes:

  ```
gplogfilter -b '2013-05-23 14:33'
```

- `-e` | `--end=datetime`
  Specifies an ending date and time to stop searching for log entries in the format of `YYYY-MM-DD [hh:mm[:ss]]`.
  If a time is specified, the date and time must be enclosed in either single or double quotes.
  This example encloses the date and time in single quotes:

  ```
gplogfilter -e '2013-05-23 14:33'
```

- `-d` | `--duration=time`
  Specifies a time duration to search for log entries in the format of `[hh][[:mm[:ss]]].` If used without either the `-b` or `-e` option, will use the current time as a basis.

**Pattern Matching Options**

- `-c` | `--case=i [gnore] | r [espect]`
  Matching of alphabetic characters is case sensitive by default unless proceeded by the `--case=ignore` option.

- `-C` | `--columns='string'`
  Selects specific columns from the log file. Specify the desired columns as a comma-delimited string of column numbers beginning with 1, where the second column from left is 2, the third is 3, and so on. See "Viewing the Database Server Log Files" in the Greenplum Database Administrator Guide for details about the log file format and for a list of the available columns and their associated number.

- `-f` | `--find='string'`
  Finds the log entries containing the specified string.

- `-F` | `--nofind='string'`
  Rejects the log entries containing the specified string.

- `-m` | `--match=regex`
  Finds log entries that match the specified Python regular expression. See `https://docs.python.org/library/re.html` for Python regular expression syntax.

- `-M` | `--nomatch=regex`
Rejects log entries that match the specified Python regular expression. See https://docs.python.org/library/re.html for Python regular expression syntax.

- **t | --trouble**
  Finds only the log entries that have ERROR:, FATAL:, or PANIC: in the first line.

**Output Options**

- **n integer | --tail=integer**
  Limits the output to the last integer of qualifying log entries found.

- **s offset [limit] | --slice=offset [limit]**
  From the list of qualifying log entries, returns the limit number of entries starting at the offset entry number, where an offset of zero (0) denotes the first entry in the result set and an offset of any number greater than zero counts back from the end of the result set.

- **o output_file | --out=output_file**
  Writes the output to the specified file or directory location instead of STDOUT.

- **z 0-9 | --zip=0-9**
  Compresses the output file to the specified compression level using gzip, where 0 is no compression and 9 is maximum compression. If you supply an output file name ending in .gz, the output file will be compressed by default using maximum compression.

- **a | --append**
  If the output file already exists, appends to the file instead of overwriting it.

**Input Options**

input_file

- The name of the input log file(s) to search through. If an input file is not supplied, gplogfilter will use the $MASTER_DATA_DIRECTORY environment variable to locate the Greenplum Database master log file. To read from standard input, use a dash (-) as the input file name.

- **u | --unzip**
  Uncompress the input file using gunzip. If the input file name ends in .gz, it will be uncompressed by default.

- **--help**
  Displays the online help.

- **--version**
  Displays the version of this utility.

**Examples**

Display the last three error messages in the master log file:

```
 gplogfilter -t -n 3
```

Display all log messages in the master log file timestamped in the last 10 minutes:

```
 gplogfilter -d :10
```

Display log messages in the master log file containing the string |con6 cmd11|:

```
 gplogfilter -f '|con6 cmd11|'
```
Using *gpssh*, run *gplogfilter* on the segment hosts and search for log messages in the segment log files containing the string *con6* and save output to a file.

```
gpssh -f seg_hosts_file -e 'source /usr/local/greenplum-db/greenplum_path.sh ; gplogfilter -f con6 /gpdata/*/pg_log/gpdb*.csv' > seglog.out
```

**See Also**

*gpssh*, *gpscp*

### gpmapreduce

Runs Greenplum MapReduce jobs as defined in a YAML specification document.

**Synopsis**

```
gpmapreduce -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]

gpmapreduce -x | --explain

gpmapreduce -X | --explain-analyze

gpmapreduce -V | --version

gpmapreduce -h | --help
```

**Prerequisites**

The following are required prior to running this program:

- You must have your MapReduce job defined in a YAML file. For information about the Greenplum MapReduce specification, see the *Greenplum Database Reference Guide*.
- You must be a Greenplum Database superuser to run MapReduce jobs written in untrusted Perl or Python.
- You must be a Greenplum Database superuser to run MapReduce jobs with *EXEC* and *FILE* inputs.
- You must be a Greenplum Database superuser to run MapReduce jobs with *GPFDIST* input unless the user has the appropriate rights granted. See the *Greenplum Database Reference Guide* for more information.

**Description**

*MapReduce* is a programming model developed by Google for processing and generating large data sets on an array of commodity servers. Greenplum MapReduce allows programmers who are familiar with the MapReduce paradigm to write map and reduce functions and submit them to the Greenplum Database parallel engine for processing.

In order for Greenplum to be able to process MapReduce functions, the functions need to be defined in a YAML document, which is then passed to the Greenplum MapReduce program, *gpmapreduce*, for execution by the Greenplum Database parallel engine. The Greenplum system takes care of the details of distributing the input data, executing the program across a set of machines, handling machine failures, and managing the required inter-machine communication.
Options
- **f** *yaml_file*
  Required. The YAML file that contains the Greenplum MapReduce job definitions. See the Greenplum Database Reference Guide.

- **-?** | --help
  Show help, then exit.

- **-V** | --version
  Show version information, then exit.

- **-v** | --verbose
  Show verbose output.

- **-x** | --explain
  Do not run MapReduce jobs, but produce explain plans.

- **-X** | --explain-analyze
  Run MapReduce jobs and produce explain-analyze plans.

- **-k** | --keyname=value
  Sets a YAML variable. A value is required. Defaults to "key" if no variable name is specified.

Connection Options
- **-h** *host* | --host *host*
  Specifies the host name of the machine on which the Greenplum master database server is running. If not specified, reads from the environment variable **PGHOST** or defaults to localhost.

- **-p** *port* | --port *port*
  Specifies the TCP port on which the Greenplum master database server is listening for connections. If not specified, reads from the environment variable **PGPORT** or defaults to 5432.

- **-U** *username* | --username *username*
  The database role name to connect as. If not specified, reads from the environment variable **PGUSER** or defaults to the current system user name.

- **-W** | --password
  Force a password prompt.

Examples
Run a MapReduce job as defined in *my_yaml.txt* and connect to the database *mydatabase*:

```
gpmapreduce -f my_yaml.txt mydatabase
```

See Also
Greenplum MapReduce specification in the Greenplum Database Reference Guide

gpmfr
Manages the Greenplum Database backup images that are stored on a local Data Domain system and a remote Data Domain system that is used for disaster recovery. Managed file replication is used for disaster recovery by the Data Domain Boost software option to transfer a backup image from one Data Domain system to another.
Synopsis

```
gpmfr --delete {LATEST | OLDEST | timestamp}[--remote]
    [--master-port=master_port] [--skip-ping]
    [--ddboost-storage-unit=unit-ID]
    [-a] [-v | --verbose]

gpmfr {--replicate | --recover} {LATEST | OLDEST | timestamp}
    --max-streams max_IO_streams [--master-port=master_port] [--skip-ping]
    [--ddboost-storage-unit=unit-ID]
    [-a] [-q | --quiet] [-v | --verbose]

gpmfr {--list {LATEST | OLDEST | timestamp}}
    [--ddboost-storage-unit=unit-ID]
    [--master-port=master_port] [--remote] [--skip-ping]
    [-v | --verbose]

gpmfr --list-files {LATEST | OLDEST | timestamp}
    [--ddboost-storage-unit=unit-ID]
    [--master-port=master_port] [--remote] [--skip-ping]
    [-v | --verbose]

gpmfr --show-streams [--skip-ping] [-v | --verbose]

gpmfr -h | --help

gpmfr --version
```

Prerequisites

The Data Domain systems that are used as local and remote backup systems for managed file replication must have Data Domain Boost and Replicator enabled.

The Greenplum Database master host segment hosts must be able to connect to both the local Data Domain system and the remote Data Domain system.

The login credentials for the local and remote Data Domain systems must be configured on the Greenplum master host with the `gpcrondump` utility. See "Backing Up and Restoring Databases" in the Greenplum Database Administrator Guide for information about setting up Data Domain systems for use with Greenplum Database.

See the Greenplum Database Release Notes for information about the supported version of Data Domain Boost.

Description

The `gpmfr` utility provides these capabilities:

- Lists the backup data sets that are on the local or the remote Data Domain system.
- Replicates a backup data set that is on the local Data Domain system to the remote system.
- Recovers a backup data set that is on the remote Data Domain system to the local system.
- Deletes a backup data set that is on the local or the remote Data Domain system.

The Greenplum Database backup sets are identified by timestamps (`yyyyymmddhhmmss`).

`gpmfr` attempts to schedule the replication task for the files in backup data set. It ensures that the limit on the maximum number of I/O streams used for replication is never exceeded. The I/O streams limit is set with the `--max-streams` option that accompanies the `--replicate` or `--recover` option.

When cancelling a replication operation, `gpmfr` kills all active replication processes and cleans up all the files on replication Data Domain system.
Options

-a
Do not prompt the user for confirmation. Progress information is displayed on the output. Specify the option -q or --quiet to write progress information to the log file.

--ddboost-storage-unit=unit-ID
Optional. Specify a valid storage unit ID for the Data Domain system that is used for the gpmfr operation. A replicate or recover operation uses the same storage unit ID on both local and remote Data Domain systems. If the storage unit on the destination Data Domain system (where the backup is being copied) is created if it does not exist.

If this option is not specified, the utility uses the storage unit specified when configuring the DD Boost credentials or the default ID GPDB.

--delete {LATEST | OLDEST | timestamp}
Deletes a Greenplum Database backup set from the local Data Domain system.

LATEST specifies deleting the latest backup set (first in chronological order).

OLDEST specifies deleting the backup set that is oldest in chronological order.

timestamp specifies deleting the Greenplum Database backup set identified by the timestamp.

--list
Lists the Greenplum Database backup sets that are on the local Data Domain system. The backup sets are identified by timestamps (yyyyymmddhhmms).

Specify the option --remote to list the Greenplum Database backup sets that are on the remote Data Domain system.

--list-files {LATEST | OLDEST | timestamp}
Lists the files in a Greenplum Database backup that is on the local Data Domain system.

Specify the option --remote to list the files in the backup set that is on the remote Data Domain system.

LATEST specifies listing the files in the latest backup set (first in chronological order).

OLDEST specifies listing the files in the backup set that is oldest in chronological order.

timestamp specifies listing the file in the backup set identified by the timestamp.

--master-port=master_port
Specifies the Greenplum Database master port number. To validate backup sets, the utility retrieves information from the Greenplum Database instance that uses the port number. If the option is not specified, the default value is 5432.

If gpmfr does not find a Greenplum Database, validation is skipped and a warning is displayed.

--max-streams max_IO_streams
Specifies the maximum number of Data Domain I/O streams that can be used when copying the backup set between the local and remote Data Domain systems.

-q | --quiet
Runs in quiet mode. File transfer progress information is not displayed on the output, it is written to the log file. If this option is not specified, progress information is only displayed on screen, it is not written to the log file.

--recover {LATEST | OLDEST | timestamp}
Recovers a Greenplum Database backup set that is available on the remote Data Domain system to the local system.
LATEST specifies recovering the most recent backup set (first in chronological order).
OLDEST specifies recovering the backup set that is oldest in chronological order.
timestamp specifies recovering the backup set identified by the timestamp.
If a backup set with the same timestamp exists on local Data Domain system, the utility prompts you to confirm replacing the backup.
A progress bar indicating transfer status of the backup set is shown on shown at the output.

--replicate {LATEST | OLDEST | timestamp}
Replicates a Greenplum Database backup set that is on the local Data Domain system to the remote system.
LATEST specifies replicating the most recent backup set (first in chronological order).
OLDEST specifies replicating the backup set that is oldest in chronological order.
timestamp specifies replicating the backup set identified by the timestamp.
If a backup set with the same timestamp exists on remote Data Domain system, the utility prompts you to confirm replacing the backup.
A progress bar indicating transfer status of the backup set is shown at the output.
A backup set must be completely backed up to the local Domain system before it can be replicated to the remote Data Domain system.

--remote
Perform the operation on the remote Data Domain system that is used for disaster recovery.
For example, gpmfr --list lists the backup sets that are on the local Data Domain system that is used to back up Greenplum Database. gpmfr --list --remote lists the backup sets that are on the remote system.

--show-streams
Displays the replication I/O stream soft limit and the number of I/O streams that are in use.

--skip-ping
Specify this option to skip the ping of a Data Domain system. gpmfr uses ping to ensure that the Data Domain system is reachable. If the Data Domain host is configured to block ICMP ping probes, specify this option to skip the ping of the Data Domain system.

-h | --help
Displays the online help.

-v | --verbose
Specifies verbose logging mode. Additional log information is written to the log file during command execution.

--version
Displays the version of this utility.

Example
The following example replicates the latest backup set on the local Data Domain server to the remote server. The maximum number of I/O streams that can be used for the replication is 30.

```bash
gpmfr --replicate LATEST --max-streams 30
```
**See Also**
gpcrondump, gpdbrestore

**gpperfmon_install**
Installs the gpperfmon database, which is used by Greenplum Command Center, and optionally enables the data collection agents.

**Synopsis**
```
gpperfmon_install --port gpdb_port
    [--enable --password gpmon_password [--pgpass path_to_file]]
    [--verbose]

gpperfmon_install --help | -h | -?
```

**Description**
The gpperfmon_install utility automates the steps required to enable the data collection agents. You must be the Greenplum Database system user (gpadmin) to run this utility. The --port option is required. When using the --enable option, the --password option is also required. Use the --port option to supply the port of the Greenplum Database master instance. If using the --enable option, Greenplum Database must be restarted after the utility completes.

When run without the --enable option, the utility just creates the gpperfmon database (the database used to store system metrics collected by the data collection agents). When run with the --enable option, the utility also runs the following additional tasks necessary to enable the performance monitor data collection agents:

1. Creates the gpmon superuser role in Greenplum Database. The data collection agents require this role to connect to the database and write their data. The gpmon superuser role uses MD5-encrypted password authentication by default. Use the --password option to set the gpmon superuser's password.
2. Updates the $MASTER_DATA_DIRECTORY/pg_hba.conf file. The utility adds these lines to the host-based authentication file (pg_hba.conf):

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>gpperfmon</td>
<td>gpmon</td>
<td>md5</td>
</tr>
<tr>
<td>host</td>
<td>all</td>
<td>gpmon</td>
<td>127.0.0.1/28</td>
</tr>
<tr>
<td>host</td>
<td>all</td>
<td>gpmon</td>
<td>::1/128</td>
</tr>
</tbody>
</table>

The second and third lines, the host entries, give gpmon access to all Greenplum Database databases.

**Note:** It might be necessary to edit the lines in the pg_hba.conf file after running the gpperfmon_install utility to limit the gpmon role's access to databases or to change the authentication method. After you edit the file, run gpstop -u to reload the file in Greenplum Database.

- To limit gpmon access to just the gpperfmon database, edit the host entries in the pg_hba.conf file. For the gpmon user change the second field from all to gpperfmon:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>gpperfmon</td>
<td>gpmon</td>
<td>md5</td>
</tr>
<tr>
<td>host</td>
<td>gpperfmon</td>
<td>gpmon</td>
<td>127.0.0.1/28</td>
</tr>
<tr>
<td>host</td>
<td>gpperfmon</td>
<td>gpmon</td>
<td>::1/128</td>
</tr>
</tbody>
</table>

- The gpperfmon_install utility assumes the default MD5 authentication method. Greenplum Database can optionally be configured to use the SHA-256 hash algorithm to compute the password hashes saved in the system catalog. This is incompatible with the
Greenplum Database Utility Guide

MD5 authentication method, which expects an MD5 hash or clear text password in the system catalog. Because of this, if you have enabled the SHA-256 hash algorithm in the database, you must edit the `pg_hba.conf` file after running the `gpperfmon_install` utility. For the host entries, change the authentication method for the `gpmon` role from `md5` to `password`:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>127.0.0.1/28</th>
<th>password</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>gpperfmon</td>
<td>gpmon</td>
<td>md5</td>
</tr>
<tr>
<td>host</td>
<td>all</td>
<td>gpmon</td>
<td>password</td>
</tr>
<tr>
<td>host</td>
<td>all</td>
<td>gpmon</td>
<td>::1/128</td>
</tr>
</tbody>
</table>

The `password` authentication method submits the user's clear text password for authentication and should not be used on an untrusted network. See "Protecting Passwords in Greenplum Database" in the Greenplum Database Administrator Guide for more information about configuring password hashing.

3. Updates the password file (`.pgpass`). In order to allow the data collection agents to connect as the `gpmon` role without a password prompt, you must have a password file that has an entry for the `gpmon` user. The utility adds the following entry to your password file (if the file does not exist, the utility will create it):

```
*:5432:gppperfmon:gpmon:gpmon_password
```

If your password file is not located in the default location (`~/.pgpass`), use the `--pgpass` option to specify the file location.

4. Sets the server configuration parameters for Greenplum Command Center. The following parameters must be enabled for the data collection agents to begin collecting data. The utility sets the following parameters in the Greenplum Database `postgresql.conf` configuration files:

- `gp_enable_gpperfmon=on` (in all `postgresql.conf` files)
- `gpperfmon_port=8888` (in all `postgresql.conf` files)
- `gp_external_enable_exec=on` (in the master `postgresql.conf` file)

Data collection agents can be configured by setting parameters in the `gpperfmon.conf` configuration file. See Data Collection Agent Configuration for details.

For information about Greenplum Command Center, see the Greenplum Command Center Documentation.

Options

--enable
In addition to creating the `gpperfmon` database, performs the additional steps required to enable the data collection agents. When `--enable` is specified the utility will also create and configure the `gpmon` superuser account and set the Command Center server configuration parameters in the `postgresql.conf` files.

--password gpmon_password
Required if `--enable` is specified. Sets the password of the `gpmon` superuser. Disallowed if `--enable` is not specified.

--port gpdb_port
Required. Specifies the connection port of the Greenplum Database master.

--pgpass path_to_file
Optional if `--enable` is specified. If the password file is not in the default location of `~/.pgpass`, specifies the location of the password file.

--verbose
Sets the logging level to verbose.

--help | -h | -?

Data Collection Agent Configuration

The `$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf` file stores configuration parameters for the data collection agents. For configuration changes to these options to take effect, you must save `gpperfmon.conf` and then restart Greenplum Database server (gpstop -r).

The `gpperfmon.conf` file contains the following configuration parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_location</td>
<td>Specifies a directory location for gpperfmon log files. Default is <code>$MASTER_DATA_DIRECTORY/gpperfmon/logs</code>.</td>
</tr>
<tr>
<td>min_query_time</td>
<td>Specifies the minimum query run time in seconds for statistics collection. All queries that run longer than this value are logged in the <code>queries_history</code> table. For queries with shorter run times, no historical data is collected. Defaults to 20 seconds. If you know that you want to collect data for all queries, you can set this parameter to a low value. Setting the minimum query run time to zero, however, collects data even for the numerous queries run by Greenplum Command Center, creating a large amount of data that may not be useful.</td>
</tr>
</tbody>
</table>
| max_log_size    | This parameter is not included in `gpperfmon.conf`, but it may be added to this file. To prevent the log files from growing to excessive size, you can add the `max_log_size` parameter to `gpperfmon.conf`. The value of this parameter is measured in bytes. For example:  

```
max_log_size = 10485760
```

With this setting, the log files will grow to 10MB before the system rolls over to a new log file. |
<p>| partition_age   | The number of months that gpperfmon statistics data will be retained. The default value is 0, which means that partitions are never dropped automatically. |
| quantum         | Specifies the time in seconds between updates from data collection agents on all segments. Valid values are 10, 15, 20, 30, and 60. Defaults to 15 seconds. If you prefer a less granular view of performance, or want to collect and analyze minimal amounts of data for system metrics, choose a higher quantum. To collect data more frequently, choose a lower value. |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>harvest_interval</td>
<td>The time, in seconds, between data harvests. A data harvest moves recent data from the gpperfmon external (_tail) tables to their corresponding history files. The default is 120. The minimum value is 30.</td>
</tr>
<tr>
<td>ignore_gexec_packet</td>
<td>(Deprecated) When set to true, data collection agents do not collect performance data in the gpperfmon database queries_* tables: rows_out, cpu_elapsed, cpu currpct, skew_cpu, and skew_rows. The default setting, true, reduces the amount of memory consumed by the gpmmon process. Set this parameter to false if you require this additional performance data.</td>
</tr>
<tr>
<td>smdw_aliases</td>
<td>This parameter allows you to specify additional host names for the standby master. For example, if the standby master has two NICs, you can enter: smdw_aliases=smdw-1,smdw-2. This optional fault tolerance parameter is useful if the Greenplum Command Center loses connectivity with the standby master. Instead of continuously retrying to connect to host smdw, it will try to connect to the NIC-based aliases of smdw-1 and/or smdw-2. This ensures that the Command Center Console can continuously poll and monitor the standby master.</td>
</tr>
</tbody>
</table>

**Notes**

The gpperfmon database and Greenplum Command Center require the gpmon role. After the gpperfmon database and gpmon role have been created, you can change the password for the gpmon role and update the information that Greenplum Command Center uses to connect to the gpperfmon database:

1. Log in to Greenplum Database as a superuser and change the gpmon password with the ALTER ROLE command.

   ```
   # ALTER ROLE gpmon WITH PASSWORD 'new_password' ;
   ```

2. Update the password in .pgpass file that is used by Greenplum Command Center. The default file location is the gpadmin home directory (~/.pgpass). The .pgpass file contains a line with the gpmon password.

   ```
   *:5432:gpperfmon:gpmon:new_password
   ```

3. Restart the Greenplum Command Center with the Command Center gpcmdr utility.

   ```
   $ gpcmdr --restart
   ```

The gpperfmon monitoring system requires some initialization after startup. Monitoring information appears after a few minutes have passed, and not immediately after installation and startup of the gpperfmon system.
Examples
Create the `gpperfmon` database only:

```
$ su - gpadmin
$ gpperfmon_install --port 5432
```

Create the `gpperfmon` database, create the `gpmon` superuser, and enable the data collection agents:

```
$ su - gpadmin
$ gpperfmon_install --enable --password changeme --port 5432
$ gpstop -r
```

See Also
`gpstop`

`gppkg`
Installs Greenplum Database extensions in `.gppkg` format, such as PL/Java, PL/R, PostGIS, and MADlib, along with their dependencies, across an entire cluster.

Synopsis
```
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
```

Description
The Greenplum Package Manager (`gppkg`) utility installs Greenplum Database extensions, along with any dependencies, on all hosts across a cluster. It will also automatically install extensions on new hosts in the case of system expansion and segment recovery.

   **Note:** After a major upgrade to Greenplum Database, you must download and install all `gppkg` extensions again.

Examples of database extensions and packages software that are delivered using the Greenplum Package Manager are:

- PL/Java
- PL/R
- PostGIS
- MADlib

Options
- `-a` (do not prompt)
  
  Do not prompt the user for confirmation.
- `-c` | `--clean`
Reconciles the package state of the cluster to match the state of the master host. Running this option after a failed or partial install/uninstall ensures that the package installation state is consistent across the cluster.

- \texttt{-d master\_data\_directory}
  
  The master data directory. If not specified, the value set for \$\text{MASTER\_DATA\_DIRECTORY} will be used.

- \texttt{-i package | --install=package}

  Installs the given package. This includes any pre/post installation steps and installation of any dependencies.

- \texttt{--migrate GPHOME\_old GPHOME\_new}

  Migrates packages from a separate \$\text{GPHOME}. Carries over packages from one version of Greenplum Database to another.

  For example: \texttt{gppkg --migrate /usr/local/greenplum-db-<old-version> /usr/local/greenplum-db-<new-version>}

  When migrating packages, these requirements must be met.

  \begin{itemize}
  \item At least the master instance of the destination Greenplum Database must be started (the instance installed in \textit{GPHOME\_new}). Before running the \texttt{gppkg} command start the Greenplum Database master with the command \texttt{gpstart -m}.
  \item Run the \texttt{gppkg} utility from the \textit{GPHOME\_new} installation. The migration destination installation directory.
  \end{itemize}

- \texttt{-q | --query query\_option}

  Provides information specified by \texttt{query\_option} about the installed packages. Only one \texttt{query\_option} can be specified at a time. The following table lists the possible values for \texttt{query\_option}. \texttt{<package\_file>} is the name of a package.

  \begin{table}[h]
  \centering
  \begin{tabular}{|l|l|}
  \hline
  \textbf{query\_option} & \textbf{Returns} \\
  \hline
  \texttt{<package\_file>} & Whether the specified package is installed. \\
  \hline
  \texttt{--info <package\_file>} & The name, version, and other information about the specified package. \\
  \hline
  \texttt{--list <package\_file>} & The file contents of the specified package. \\
  \hline
  \texttt{--all} & List of all installed packages. \\
  \hline
  \end{tabular}
  \caption{Query Options for gppkg}
  \end{table}

- \texttt{-r name-version | --remove=name-version}

  Removes the specified package.

- \texttt{-u package | --update=package}

  Updates the given package.

  \textbf{Warning}: The process of updating a package includes removing all previous versions of the system objects related to the package. For example, previous versions of shared libraries are removed. After the update process, a database function will fail when it is called if the function references a package file that has been removed.

- \texttt{--version (show utility version)}

  Displays the version of this utility.

- \texttt{-v | --verbose}
Sets the logging level to verbose.

-? | -h | --help
   Displays the online help.

**gprecoverseg**

Recovers a primary or mirror segment instance that has been marked as down (if mirroring is enabled).

**Synopsis**

```
gprecoverseg [-p new_recover_host[,...]] | -i recover_config_file | 
            -s filesystem_config_file] [-d master_data_directory] 
gprecoverseg -r 
gprecoverseg -o output_recover_config_file | -S output_filespace_config_file 
            [-p new_recover_host[,...]] 
gprecoverseg -? 
gprecoverseg --version
```

**Description**

In a system with mirrors enabled, the **gprecoverseg** utility reactivates a failed segment instance and identifies the changed database files that require resynchronization. Once **gprecoverseg** completes this process, the system goes into **resyncronizing** mode until the recovered segment is brought up to date. The system is online and fully operational during resyncronization.

During an incremental recovery (the **-F** option is not specified), if **gprecoverseg** detects a segment instance with mirroring disabled in a system with mirrors enabled, the utility reports that mirroring is disabled for the segment, does not attempt to recover that segment instance, and continues the recovery process.

A segment instance can fail for several reasons, such as a host failure, network failure, or disk failure. When a segment instance fails, its status is marked as **down** in the Greenplum Database system catalog, and its mirror is activated in **change tracking** mode. In order to bring the failed segment instance back into operation again, you must first correct the problem that made it fail in the first place, and then recover the segment instance in Greenplum Database using **gprecoverseg**.

Segment recovery using **gprecoverseg** requires that you have an active mirror to recover from. For systems that do not have mirroring enabled, or in the event of a double fault (a primary and mirror pair both down at the same time) — you must take manual steps to recover the failed segment instances and then perform a system restart to bring the segments back online. For example, this command restarts a system.

```
gpstop -r
```

By default, a failed segment is recovered in place, meaning that the system brings the segment back online on the same host and data directory location on which it was originally configured. In this case, use the following format for the recovery configuration file (using **-i**).

```
filespaceOrder=[filesystem1_fsnme[, filesystem2_fsnme[, ...]]]
<failed_host_address>:<port>:<data_directory>
```

In some cases, this may not be possible (for example, if a host was physically damaged and cannot be recovered). In this situation, **gprecoverseg** allows you to recover failed segments to a completely new host (using **-p**), on an alternative data directory location on your remaining live segment hosts (using **-s**),
or by supplying a recovery configuration file (using \texttt{-i}) in the following format. The word \texttt{SPACE} indicates the location of a required space. Do not add additional spaces.

\begin{verbatim}
filesystemOrder=[filesystem1_fnname[, filesystem2_fnname[, ...]]]
<failed_host_address>:<port>:<data_directory>SPACE
<recovery_host_address>:<port>:<replication_port>:<data_directory>
[:<fselocation>:...]
\end{verbatim}

See the \texttt{-i} option below for details and examples of a recovery configuration file.

The \texttt{gp_segment_configuration}, \texttt{pg_filespace}, and \texttt{pg_filespace_entry} system catalog tables can help you determine your current segment configuration so that you can plan your mirror recovery configuration. For example, run the following query:

\begin{verbatim}
=# SELECT dbid, content, address, port,
    replication_port, fselocation as datadir
 FROM gp_segment_configuration, pg_filespace_entry
 WHERE dbid=fseDbid
 ORDER BY dbid;
\end{verbatim}

The new recovery segment host must be pre-installed with the Greenplum Database software and configured exactly the same as the existing segment hosts. A spare data directory location must exist on all currently configured segment hosts and have enough disk space to accommodate the failed segments.

The recovery process marks the segment as up again in the Greenplum Database system catalog, and then initiates the resynchronization process to bring the transactional state of the segment up-to-date with the latest changes. The system is online and available during resynchronization. To check the status of the resynchronization process run:

\begin{verbatim}
gpstate -m
\end{verbatim}

\section*{Options}
\begin{itemize}
\item \texttt{-a (do not prompt)}
  Do not prompt the user for confirmation.
\item \texttt{-B parallel\_processes}
  The number of segments to recover in parallel. If not specified, the utility will start up to four parallel processes depending on how many segment instances it needs to recover.
\item \texttt{-d master\_data\_directory}
  Optional. The master host data directory. If not specified, the value set for \texttt{$MASTER\_DATA\_DIRECTORY} will be used.
\item \texttt{-F (full recovery)}
  Optional. Perform a full copy of the active segment instance in order to recover the failed segment. The default is to only copy over the incremental changes that occurred while the segment was down.
\item \texttt{-i recover\_config\_file}
  Specifies the name of a file with the details about failed segments to recover. Each line in the file is in the following format. The word \texttt{SPACE} indicates the location of a required space. Do not add additional spaces.

\begin{verbatim}
filesystemOrder=[filesystem1_fnname[, filesystem2_fnname[, ...]]]
<failed_host_address>:<port>:<data_directory>SPACE
<recovery_host_address>:<port>:<replication_port>:<data_directory>
[:<fselocation>:...]
\end{verbatim}
\end{itemize}
Lines beginning with # are treated as comments and ignored.

**Filespace Order**

The first comment line that is not a comment specifies filespace ordering. This line starts with `filespaceOrder=` and is followed by list of filespace names delimited by a colon. For example:

```
filespaceOrder=raid1:raid2
```

The default `pg_system` filespace should not appear in this list. The list should be left empty on a system with no filespaces other than the default `pg_system` filespace. For example:

```
filespaceOrder=
```

**Segments to Recover**

Each line after the first specifies a segment to recover. This line can have one of two formats. In the event of in-place recovery, enter one group of colon delimited fields in the line. For example:

```
failedAddress:failedPort:failedDataDirectory
```

For recovery to a new location, enter two groups of fields separated by a space in the line. The required space is indicated by `SPACE`. Do not add additional spaces.

```
failedAddress:failedPort:failedDataDirectory SPACE newAddress:
```

On a system with additional filespaces, the second group of fields is expected to be followed with a list of the corresponding filespace locations separated by additional colons. For example, on a system with two additional filespaces, enter two additional directories in the second group, as follows. The required space is indicated by `SPACE`. Do not add additional spaces.

```
failedAddress:failedPort:failedDataDirectory SPACE newAddress:
```

**Examples**

**In-place recovery of a single mirror**

```
filespaceOrder=
sdw1-1:50001:/data1/mirror/gpseg16
```

**Recovery of a single mirror to a new host**

```
filespaceOrder=
sdw1-1:50001:/data1/mirror/gpseg16 SPACE sdw4-1:51001:/data1/recover1/gpseg16
```

**Recovery of a single mirror to a new host on a system with an extra filespace**

```
filespaceOrder=
fs1sdw1-1:50001:/data1/mirror/gpseg16 SPACE sdw4-1:50001:51001:/data1/recover1/gpseg16:/data1/fs1/gpseg16
```

**Obtaining a Sample File**
You can use the `-o` option to output a sample recovery configuration file to use as a starting point.

`-l logfile_directory`

The directory to write the log file. Defaults to `~/gpAdminLogs`.

`-o output_recover_config_file`

Specifies a file name and location to output a sample recovery configuration file. The output file lists the currently invalid segments and their default recovery location in the format that is required by the `-i` option. Use together with the `-p` option to output a sample file for recovering on a different host. This file can be edited to supply alternate recovery locations if needed.

`-p new_recover_host[,....]`

Specifies a spare host outside of the currently configured Greenplum Database array on which to recover invalid segments. In the case of multiple failed segment hosts, you can specify a comma-separated list. The spare host must have the Greenplum Database software installed and configured, and have the same hardware and OS configuration as the current segment hosts (same OS version, locales, gpadmin user account, data directory locations created, ssh keys exchanged, number of network interfaces, network interface naming convention, and so on.).

`-q (no screen output)`

Run in quiet mode. Command output is not displayed on the screen, but is still written to the log file.

`-r (rebalance segments)`

After a segment recovery, segment instances may not be returned to the preferred role that they were given at system initialization time. This can leave the system in a potentially unbalanced state, as some segment hosts may have more active segments than is optimal for top system performance. This option rebalances primary and mirror segments by returning them to their preferred roles. All segments must be valid and synchronized before running `gprecoverseg -r`. If there are any in progress queries, they will be cancelled and rolled back.

`-s filesystem_config_file`

Specifies the name of a configuration file that contains file system locations on the currently configured segment hosts where you can recover failed segment instances. The filesystem configuration file is in the format of:

```
pag_system=default_fselocation
filesystem1_name=filesystem1_fselocation
filesystem2_name=filesystem2_fselocation
...
```

If your system does not have additional filesystems configured, this file will only have one location (for the default filesystem, `pg_system`). These file system locations must exist on all segment hosts in the array and have sufficient disk space to accommodate recovered segments.

Note: The `-s` and `-S` options are only used when you recover to existing hosts in the cluster. You cannot use these options when you recover to a new host. To recover to a new host, use the `-i` and `-o` options.

`-S output_filespace_config_file`

Specifies a file name and location to output a sample filesystem configuration file in the format that is required by the `-s` option. This file should be edited to supply the correct alternate filesystem locations.

`-v (verbose)`
Sets logging output to verbose.

```
--version  (version)
```
Displays the version of this utility.

```
-?  (help)
```
Displays the online help.

### Examples

**Recover any failed segment instances in place:**

```
$ gprecoverseg
```

Rebalance your Greenplum Database system after a recovery by resetting all segments to their preferred role. First check that all segments are up and synchronized.

```
$ gpstate -m
$ gprecoverseg -r
```

**Recover any failed segment instances to a newly configured spare segment host:**

```
$ gprecoverseg -i recover_config_file
```

**Output the default recovery configuration file:**

```
$ gprecoverseg -o /home/gpadmin/recover_config_file
```

### See Also

`gpstart`, `gpstop`

---

**gpreload**

Reloads Greenplum Database table data sorting the data based on specified columns.

### Synopsis

```
gpreload  -d  database  [-p  port]  [-t  |  --table-file]  path_to_file  [-a]
gpreload  -h
```

### Description

The `gpreload` utility reloads table data with column data sorted. For tables that were created with the table storage option `APPENDONLY=TRUE` and compression enabled, reloading the data with sorted data can improve table compression. You specify a list of tables to be reloaded the table column to be sorted in a text file.

Compression is improved by sorting data when the data in the column has a relatively low number of distinct values when compared to the total number of rows.
For a table being reloaded, the order of the columns to be sorted might affect compression. The columns with fewest distinct values should be listed first. For example, listing state then city would generally result in better compression than listing city then state.

```
public.cust_table: state, city
public.cust_table: city, state
```

For information about the format of the file used with `gpreload`, see the `--table-file` option.

**Notes**

To improve reload performance, indexes on tables being reloaded should be removed before reloading the data.

Running the `ANALYZE` command after reloading table data might query performance because of a change in the data distribution of the reloaded data.

**Options**

- `-a` *(do not prompt)*
  Optional. If specified, the `gpreload` utility does not prompt the user for confirmation.

- `-d` *database*
  The database that contains the tables to be reloaded. The `gpreload` utility connects to the database as the user running the utility.

- `-p` *port*
  The Greenplum Database master port. If not specified, the value of the `PGPORT` environment variable is used. If the value is not available, an error is returned.

- `{ -t | --table-file }` *path_to_file*
  The location and name of file containing list of schema qualified table names to reload and the column names to reorder from the Greenplum Database. Only user defined tables are supported. Views or system catalog tables are not supported.
  If indexes are defined on table listed in the file, `gpreload` prompts to continue.
  Each line specifies a table name and the list of columns to sort. This is the format of each line in the file:

```
schema.table_name: column [desc] [, column2 [desc] ... ]
```

The table name is followed by a colon (`:`) and then at least one column name. If you specify more than one column, separate the column names with a comma. The columns are sorted in ascending order. Specify the keyword `desc` after the column name to sort the column in descending order.

Wildcard characters are not supported.

If there are errors in the file, `gpreload` reports the first error and exits. No data is reloaded.

The following example reloads three tables:

```
public.clients: region, state, rep_id desc
public.merchants: region, state
test.lineitem: group, assy, whse
```

In the first table `public.clients`, the data in the `rep_id` column is sorted in descending order. The data in the other columns are sorted in ascending order.

```
--version (show utility version)
```
Displays the version of this utility.

-? (help)
Displays the online help.

Example
This example command reloads the tables in the database mytest that are listed in the file data-tables.txt.

gpreload -d mytest --table-file data-tables.txt

See Also
CREATE TABLE in the Greenplum Database Reference Guide

gpscp
Copies files between multiple hosts at once.

Synopsis

gpscp { -f hostfile_gpssh | -h hostname [-h hostname ...] } 
[--J character] [-v] [[user@]hostname:]file_to_copy [...] 
[[user@]hostname:]copy_to_path

gpscp -?
gpscp --version

Description
The gpscp utility allows you to copy one or more files from the specified hosts to other specified hosts in one command using SCP (secure copy). For example, you can copy a file from the Greenplum Database master host to all of the segment hosts at the same time.

To specify the hosts involved in the SCP session, use the -f option to specify a file containing a list of host names, or use the -h option to name single host names on the command-line. At least one host name (-h) or a host file (-f) is required. The -J option allows you to specify a single character to substitute for the hostname in the copy from and copy to destination strings. If -J is not specified, the default substitution character is an equal sign (=). For example, the following command will copy .bashrc from the local host to /home/gpadmin on all hosts named in hostfile_gpssh:

    gpscp -f hostfile_gpssh .bashrc =:/home/gpadmin

If a user name is not specified in the host list or with user@ in the file path, gpscp will copy files as the currently logged in user. To determine the currently logged in user, do a whoami command. By default, gpscp goes to $HOME of the session user on the remote hosts after login. To ensure the file is copied to the correct location on the remote hosts, it is recommended that you use absolute paths.

Before using gpscp, you must have a trusted host setup between the hosts involved in the SCP session. You can use the utility gpssh-exkeys to update the known host files and exchange public keys between hosts if you have not done so already.

Options

- f hostfile_gpssh
Specifies the name of a file that contains a list of hosts that will participate in this SCP session. The syntax of the host file is one host per line as follows:

\[<\text{hostname}>\]

-\( h \)  \( \text{hostname} \)

Specifies a single host name that will participate in this SCP session. You can use the -\( h \) option multiple times to specify multiple host names.

-\( J \)  \( \text{character} \)

The -\( J \) option allows you to specify a single character to substitute for the \( \text{hostname} \) in the copy from and copy to destination strings. If -\( J \) is not specified, the default substitution character is an equal sign (\( = \)).

-\( v \) (verbose mode)

Optional. Reports additional messages in addition to the SCP command output.

\( \text{file\_to\_copy} \)

Required. The file name (or absolute path) of a file that you want to copy to other hosts (or file locations). This can be either a file on the local host or on another named host.

\( \text{copy\_to\_path} \)

Required. The path where you want the file(s) to be copied on the named hosts. If an absolute path is not used, the file will be copied relative to \( $\text{HOME} \) of the session user. You can also use the equal sign ‘\( = \)' (or another character that you specify with the -\( J \) option) in place of a \( \text{hostname} \). This will then substitute in each host name as specified in the supplied host file (-\( f \)) or with the -\( h \) option.

-\( ? \) (help)

Displays the online help.

--version

Displays the version of this utility.

Examples

Copy the file named \text{installer.tar} to / on all the hosts in the file hostfile_gpssh.

\[\text{gpscp -f hostfile_gpssh installer.tar =:/}\]

Copy the file named \text{myfuncs.so} to the specified location on the hosts named \text{sdw1} and \text{sdw2}:

\[\text{gpscp -h sdw1 -h sdw2 myfuncs.so =:/usr/local/greenplum-db/lib}\]

See Also

\text{gpssh, gpssh-exkeys}

\text{gpseginstall}

Installs Greenplum Database on segment hosts.

Synopsis

\text{gpseginstall -f hostfile [-u gpdb_admin_user] [-p password]}\[\[-c u|p|c|s|E|e|l|v]\]

\text{gpseginstall --help}
Description

The gpseginstall utility provides a simple way to quickly install Greenplum Database on segment hosts that you specify in a host list file. The utility does not install or update Greenplum Database on the master host. You can run gpseginstall as root or as a non-root user. gpseginstall does not perform database initialization. See gpinitsystem for more information about initializing Greenplum Database.

When run as root, gpseginstall default actions are to add a system user (default is gpadmin), create a password (default is changeme), and deploy and install Greenplum Database on segment hosts. To do this, gpseginstall locates the current Greenplum Database binaries on the master from the installation path in the current user's environment variables ($GPHOME). It compresses Greenplum Database software into a tar.gz file and performs an MD5 checksum to verify file integrity.

Then, it copies Greenplum Database to the segment hosts, installs (decompresses) Greenplum Database, and changes the ownership of the Greenplum Database installation to the system user you specify with the -u option. Lastly, it exchanges keys between all Greenplum Database hosts as both root and as the system user you specify with the -u option. gpseginstall also perform a user limit check and verifies the version number of Greenplum Database on all the segments.

If you run gpseginstall as a non-root user, gpseginstall only compresses, copies, and installs Greenplum Database on segment hosts. It can also exchanges keys between Greenplum Database hosts for the current system user, and verifies the version number of Greenplum Database on all the segments.

Options

-c | --commands option_list

Optional. This allows you to customize gpseginstall actions. Note that these command options are executed by default if you do not specify the -c option in the gpseginstall syntax.

- u: Adds a system user. (root only)
- p: Changes the password for a system user. (root only)
- s: Compresses, copies, decompresses (installs) Greenplum Database on all segments.
- c: Changes the ownership of the Greenplum Database installation directory on the segment hosts. (root only)
- E: Exchange keys between Greenplum Database master and segment hosts for the root user. (root only)
- e: Exchange keys between Greenplum Database master and segment hosts for the non-root system user.
- l: (Linux only) Checks and modifies the user limits configuration file (/etc/security/limits.conf file) when adding a new user to segment hosts. (root only)
- v: Verifies the version of Greenplum Database running on all segments.

-f | --file hostfile

Required. This specifies the file that lists the segment hosts onto which you want to install Greenplum Database.

The host list file must have one host name per line and includes a host name for each segment host in your Greenplum system. Make sure there are no blank lines or extra spaces. If a host has multiple configured host names, use only one host name per host. For example:

```
sdw1-1
sdw2-1
sdw3-1
```
If available, you can use the same `gpssh-exkeys` host list file you used to exchange keys between Greenplum Database hosts.

**-p | --password password**
Optional. Sets the password for the user you specify with the `-u` option. The default password is `changeme`. This option is only available when you run `gpsetinstall` as root.

Recommended security best practices:
- Always use passwords.
- Do not use default passwords.
- Change default passwords immediately after installation.

**-u | --user user**
Optional. This specifies the system user. This user is also the Greenplum Database administrative user. This user owns Greenplum Database installation and administers the database. This is also the user under which Greenplum Database is started/initialized. This option is only available when you run `gpseginstall` as root. The default is `gpadmin`.

**--help (help)**
Displays the online help.

### Examples

As root, install a Greenplum Database on all segments, leave the system user as the default (`gpadmin`) and set the `gpadmin` password to `secret123`:

```
# gpsetinstall -f my_host_list_file -p secret123
```

As a non-root user, compress and copy Greenplum Database binaries to all segments (as `gpadmin`):

```
$ gpseginstall -f host_file
```

As root, add a user (`gpadmin2`), set the password for the user (`secret1234`), exchange keys between hosts as the new user, check user limits, and verify version numbers, but do not change ownership of Greenplum binaries, compress/copy/install Greenplum Database on segments, or exchange keys as root.

```
$ gpseginstall -f host_file -u gpadmin2 -p secret1234 -c upelv
```

### See Also

`gpinitsystem`, `gpssh-exkeys`

### gpssh

Provides SSH access to multiple hosts at once.

### Synopsis

```
gpssh { -f hostfile_gpssh | -h hostname [-h hostname ...] } [-s] [-e] [-d seconds] [-t multiplier] [-v] [bash_command]
gpssh -?
```
gpssh --version

Description
The gpssh utility allows you to run bash shell commands on multiple hosts at once using SSH (secure shell). You can execute a single command by specifying it on the command-line, or omit the command to enter into an interactive command-line session.

To specify the hosts involved in the SSH session, use the -f option to specify a file containing a list of host names, or use the -h option to name single host names on the command-line. At least one host name (-h) or a host file (-f) is required. Note that the current host is not included in the session by default — to include the local host, you must explicitly declare it in the list of hosts involved in the session.

Before using gpssh, you must have a trusted host setup between the hosts involved in the SSH session. You can use the utility gpssh-exkeys to update the known host files and exchange public keys between hosts if you have not done so already.

If you do not specify a command on the command-line, gpssh will go into interactive mode. At the gpssh command prompt (=>), you can enter a command as you would in a regular bash terminal command-line, and the command will be executed on all hosts involved in the session. To end an interactive session, press CTRL+D on the keyboard or type exit or quit.

If a user name is not specified in the host file, gpssh will execute commands as the currently logged in user. To determine the currently logged in user, do a whoami command. By default, gpssh goes to $HOME of the session user on the remote hosts after login. To ensure commands are executed correctly on all remote hosts, you should always enter absolute paths.

If you encounter network timeout problems when using gpssh, you can use -d and -t options or set parameters in the gpssh.conf file to control the timing that gpssh uses when validating the initial ssh connection. For information about the configuration file, see gpssh Configuration File.

Options
bash_command
A bash shell command to execute on all hosts involved in this session (optionally enclosed in quotes). If not specified, gpssh starts an interactive session.

-d (delay) seconds
Optional. Specifies the time, in seconds, to wait at the start of a gpssh interaction with ssh. Default is 0.05. This option overrides the delaybeforesend value that is specified in the gpssh.conf configuration file.
Increasing this value can cause a long wait time during gpssh startup.

-e (echo)
Optional. Echoes the commands passed to each host and their resulting output while running in non-interactive mode.

-f hostfile_gpssh
Specifies the name of a file that contains a list of hosts that will participate in this SSH session. The host name is required, and you can optionally specify an alternate user name and/or SSH port number per host. The syntax of the host file is one host per line as follows:

[username@]hostname[:ssh_port]

-h hostname
Specifies a single host name that will participate in this SSH session. You can use the -h option multiple times to specify multiple host names.

-s
Optional. If specified, before executing any commands on the target host, gpssh sources the file greenplum_path.sh in the directory specified by the $GPHOME environment variable.

This option is valid for both interactive mode and single command mode.

-t multiplier
Optional. A decimal number greater than 0 (zero) that is the multiplier for the timeout that gpssh uses when validating the ssh prompt. Default is 1. This option overrides the prompt_validation_timeout value that is specified in the gpssh.conf configuration file.

Increasing this value has a small impact during gpssh startup.

-v (verbose mode)
Optional. Reports additional messages in addition to the command output when running in non-interactive mode.

--version
Displays the version of this utility.

-? (help)
Displays the online help.

gpssh Configuration File
The gpssh.conf file contains parameters that let you adjust the timing that gpssh uses when validating the initial ssh connection. These parameters affect the network connection before the gpssh session executes commands with ssh. The location of the file is specified by the environment variable MASTER_DATA_DIRECTORY. If the environment variable is not defined or the gpssh.conf file does not exist, gpssh uses the default values or the values set with the -d and -t options. For information about the environment variable, see the Greenplum Database Reference Guide.

The gpssh.conf file is a text file that consists of a [gpssh] section and parameters. On a line, the # (pound sign) indicates the start of a comment. This is an example gpssh.conf file.

```
[gpssh]
delaybeforesend = 0.05
prompt_validation_timeout = 1.0
sync_retries = 5
```

These are the gpssh.conf parameters.

delaybeforesend = seconds
Specifies the time, in seconds, to wait at the start of a gpssh interaction with ssh. Default is 0.05. Increasing this value can cause a long wait time during gpssh startup. The -d option overrides this parameter.

prompt_validation_timeout = multiplier
A decimal number greater than 0 (zero) that is the multiplier for the timeout that gpssh uses when validating the ssh prompt. Increasing this value has a small impact during gpssh startup. Default is 1. The -t option overrides this parameter.

sync_retries = attempts
A non-negative integer that specifies the maximum number of times that gpssh attempts to connect to a remote Greenplum Database host. The default is 3. If the value is 0, gpssh returns an error if the initial connection attempt fails. Increasing the number of attempts also increases the time between retry attempts. This parameter cannot be configured with a command-line option.

The -t option also affects the time between retry attempts.
Increasing this value can compensate for slow network performance or segment host performance issues such as heavy CPU or I/O load. However, when a connection cannot be established, an increased value also increases the delay when an error is returned.

Examples

Start an interactive group SSH session with all hosts listed in the file hostfile_gpssh:

```
$ gpssh -f hostfile_gpssh
```

At the `gpssh` interactive command prompt, run a shell command on all the hosts involved in this session.

```
=> ls -a /data/primary/*
```

Exit an interactive session:

```
=> exit
=> quit
```

Start a non-interactive group SSH session with the hosts named `sdw1` and `sdw2` and pass a file containing several commands named `command_file` to `gpssh`:

```
$ gpssh -h sdw1 -h sdw2 -v -e < command_file
```

Execute single commands in non-interactive mode on hosts `sdw2` and `localhost`:

```
$ 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'
```

See Also

`gpssh-exkeys`, `gpscp`

**gpssh-exkeys**

Exchanges SSH public keys between hosts.

**Synopsis**

```
gpssh-exkeys -f hostfile_exkeys | - h hostname [-h hostname ...]
gpssh-exkeys -e hostfile_exkeys -x hostfile_gpexpand
gpssh-exkeys -?
gpssh-exkeys --version
```

**Description**

The `gpssh-exkeys` utility exchanges SSH keys between the specified host names (or host addresses). This allows SSH connections between Greenplum hosts and network interfaces without a password prompt. The utility is used to initially prepare a Greenplum Database system for password-free SSH access, and also to add additional ssh keys when expanding a Greenplum Database system.

To specify the hosts involved in an initial SSH key exchange, use the `-f` option to specify a file containing a list of host names (recommended), or use the `-h` option to name single host names on the command-
line. At least one host name (-h) or a host file is required. Note that the local host is included in the key exchange by default.

To specify new expansion hosts to be added to an existing Greenplum Database system, use the -e and -x options. The -e option specifies a file containing a list of existing hosts in the system that already have SSH keys. The -x option specifies a file containing a list of new hosts that need to participate in the SSH key exchange.

Keys are exchanged as the currently logged in user. You should perform the key exchange process twice: once as root and once as the gpadmin user (the user designated to own your Greenplum Database installation). The Greenplum Database management utilities require that the same non-root user be created on all hosts in the Greenplum Database system, and the utilities must be able to connect as that user to all hosts without a password prompt.

The gpssh-exkeys utility performs key exchange using the following steps:

- Creates an RSA identification key pair for the current user if one does not already exist. The public key of this pair is added to the authorized_keys file of the current user.
- Updates the known_hosts file of the current user with the host key of each host specified using the -h, -f, -e, and -x options.
- Connects to each host using ssh and obtains the authorized_keys, known_hosts, and id_rsa.pub files to set up password-free access.
- Adds keys from the id_rsa.pub files obtained from each host to the authorized_keys file of the current user.
- Updates the authorized_keys, known_hosts, and id_rsa.pub files on all hosts with new host information (if any).

**Options**

- **-e hostfile_exkeys**
  When doing a system expansion, this is the name and location of a file containing all configured host names and host addresses (interface names) for each host in your current Greenplum system (master, standby master and segments), one name per line without blank lines or extra spaces. Hosts specified in this file cannot be specified in the host file used with -x.

- **-f hostfile_exkeys**
  Specifies the name and location of a file containing all configured host names and host addresses (interface names) for each host in your Greenplum system (master, standby master and segments), one name per line without blank lines or extra spaces.

- **-h hostname**
  Specifies a single host name (or host address) that will participate in the SSH key exchange. You can use the -h option multiple times to specify multiple host names and host addresses.

- **--version**
  Displays the version of this utility.

- **-x hostfile_gexpand**
  When doing a system expansion, this is the name and location of a file containing all configured host names and host addresses (interface names) for each new segment host you are adding to your Greenplum system, one name per line without blank lines or extra spaces. Hosts specified in this file cannot be specified in the host file used with -e.

- **-? (help)**
  Displays the online help.
Examples

Exchange SSH keys between all host names and addresses listed in the file `hostfile_exkeys`:

```bash
$ gpssh-exkeys -f hostfile_exkeys
```

Exchange SSH keys between the hosts `sdw1`, `sdw2`, and `sdw3`:

```bash
$ gpssh-exkeys -h sdw1 -h sdw2 -h sdw3
```

Exchange SSH keys between existing hosts `sdw1`, `sdw2`, and `sdw3`, and new hosts `sdw4` and `sdw5` as part of a system expansion operation:

```bash
$ 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
```

See Also

`gpssh`, `gpscp`

**gpstart**

Starts a Greenplum Database system.

**Synopsis**

```bash
gpstart -? | -h | --help
gpstart --version
```
Description

The gpstart utility is used to start the Greenplum Database server processes. When you start a Greenplum Database system, you are actually starting several postgres database server listener processes at once (the master and all of the segment instances). The gpstart utility handles the startup of the individual instances. Each instance is started in parallel.

The first time an administrator runs gpstart, the utility creates a hosts cache file named .gphostcache in the user's home directory. Subsequently, the utility uses this list of hosts to start the system more efficiently. If new hosts are added to the system, you must manually remove this file from the gpadmin user's home directory. The utility will create a new hosts cache file at the next startup.

As part of the startup process, the utility checks the consistency of heap checksum setting among the Greenplum Database master and segment instances, either enabled or disabled on all instances. If the heap checksum setting is different among the instances, an error is returned and Greenplum Database does not start. The validation can be disabled by specifying the option --skip-heap-checksum-validation. For more information about heap checksums, see Enabling High Availability and Data Consistency Features in the Greenplum Database Administrator Guide.

Note: Before you can start a Greenplum Database system, you must have initialized the system using gpinitsystem. Enabling or disabling heap checksums is set when you initialize the system and cannot be changed after initialization.

Options

-a
Do not prompt the user for confirmation.

-B parallel_processes
The number of segments to start in parallel. If not specified, the utility will start up to 64 parallel processes depending on how many segment instances it needs to start.

-d master_data_directory
Optional. The master host data directory. If not specified, the value set for $MASTER_DATA_DIRECTORY will be used.

-l logfile_directory
The directory to write the log file. Defaults to ~/gpAdminLogs.

-m
Optional. Starts the master instance only, which may be useful for maintenance tasks. This mode only allows connections to the master in utility mode. For example:

```bash
PGOPTIONS='-c gp_session_role=utility' psql
```

The consistency of the heap checksum setting on master and segment instances is not checked.

-q
Run in quiet mode. Command output is not displayed on the screen, but is still written to the log file.

-R
Starts Greenplum Database in restricted mode (only database superusers are allowed to connect).

--skip-heap-checksum-validation
During startup, the utility does not validate the consistency of the heap checksum setting among the Greenplum Database master and segment instances. The default is to ensure that the heap checksum setting is the same on all instances, either enabled or disabled.
**Warning:** Starting Greenplum Database without this validation could lead to data loss. Use this option to start Greenplum Database only when it is necessary to ignore the heap checksum verification errors to recover data or to troubleshoot the errors.

-\( t \) **timeout_seconds**
  Specifies a timeout in seconds to wait for a segment instance to start up. If a segment instance was shutdown abnormally (due to power failure or killing its `postgres` database listener process, for example), it may take longer to start up due to the database recovery and validation process. If not specified, the default timeout is 60 seconds.

-\( v \)
  Displays detailed status, progress and error messages output by the utility.

-\( y \)
  Optional. Do not start the standby master host. The default is to start the standby master host and synchronization process.

-\( ? \) | -\( h \) | **--help**
  Displays the online help.

**--version**
  Displays the version of this utility.

**Examples**

Start a Greenplum Database system:

```
gpstart
```

Start a Greenplum Database system in restricted mode (only allow superuser connections):

```
gpstart -R
```

Start the Greenplum master instance only and connect in utility mode:

```
gpstart -m PGOPTIONS='\-c gp_session_role=utility' psql
```

**See Also**

`gpstop`, `gpinitsystem`

**gpstate**

Shows the status of a running Greenplum Database system.

**Synopsis**

```
gpstate [-d master_data_directory] [-B parallel_processes]
[\-s | \-b | \-Q | \-e] [-m | \-c] [-p] [-i] [-f] [-v | -q]
[\-l log_directory]
gpstate \-? | \-h | --help
```

**Description**

The `gpstate` utility displays information about a running Greenplum Database instance. There is additional information you may want to know about a Greenplum Database system, since it is comprised
of multiple PostgreSQL database instances (segments) spanning multiple machines. The `gpstate` utility provides additional status information for a Greenplum Database system, such as:

- Which segments are down.
- Master and segment configuration information (hosts, data directories, etc.).
- The ports used by the system.
- A mapping of primary segments to their corresponding mirror segments.

**Options**

-b *(brief status)*
Optional. Display a brief summary of the state of the Greenplum Database system. This is the default option.

-B *parallel_processes*
The number of segments to check in parallel. If not specified, the utility will start up to 60 parallel processes depending on how many segment instances it needs to check.

-c *(show primary to mirror mappings)*
Optional. Display mapping of primary segments to their corresponding mirror segments.

-d *master_data_directory*
Optional. The master data directory. If not specified, the value set for `$MASTER_DATA_DIRECTORY` will be used.

-e *(show segments with mirror status issues)*
Show details on primary/mirror segment pairs that have potential issues such as 1) the active segment is running in change tracking mode, meaning a segment is down 2) the active segment is in resynchronization mode, meaning it is catching up changes to the mirror 3) a segment is not in its preferred role, for example a segment that was a primary at system initialization time is now acting as a mirror, meaning you may have one or more segment hosts with unbalanced processing load.

-f *(show standby master details)*
Display details of the standby master host if configured.

-i *(show Greenplum Database version)*
Display the Greenplum Database software version information for each instance.

-l *logfile_directory*
The directory to write the log file. Defaults to `~/gpAdminLogs`.

-m *(list mirrors)*
Optional. List the mirror segment instances in the system, their current role, and synchronization status.

-p *(show ports)*
List the port numbers used throughout the Greenplum Database system.

-q *(no screen output)*
Optional. Run in quiet mode. Except for warning messages, command output is not displayed on the screen. However, this information is still written to the log file.

-Q *(quick status)*
Optional. Checks segment status in the system catalog on the master host. Does not poll the segments for status.

-s *(detailed status)*
Optional. Displays detailed status information for the Greenplum Database system.

-v *(verbose output)*
Optional. Displays error messages and outputs detailed status and progress information.

-? | -h | --help (help)
  Displays the online help.

**Output Field Definitions**
The following output fields are reported by `gpstate -s` for the master:

**Table 70: gpstate output data for the master**

<table>
<thead>
<tr>
<th>Output Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master host</td>
<td>host name of the master</td>
</tr>
<tr>
<td>Master postgres process ID</td>
<td>PID of the master database listener process</td>
</tr>
<tr>
<td>Master data directory</td>
<td>file system location of the master data directory</td>
</tr>
<tr>
<td>Master port</td>
<td>port of the master database listener process</td>
</tr>
<tr>
<td>Master current role</td>
<td>dispatch = regular operating mode, utility = maintenance mode</td>
</tr>
<tr>
<td>Greenplum array configuration type</td>
<td>Standard = one NIC per host, Multi-Home = multiple NICs per host</td>
</tr>
<tr>
<td>Greenplum initssystem version</td>
<td>version of Greenplum Database when system was first initialized</td>
</tr>
<tr>
<td>Greenplum current version</td>
<td>current version of Greenplum Database</td>
</tr>
<tr>
<td>Postgres version</td>
<td>version of PostgreSQL that Greenplum Database is based on</td>
</tr>
<tr>
<td>Greenplum mirroring status</td>
<td>physical mirroring or none</td>
</tr>
<tr>
<td>Master standby</td>
<td>host name of the standby master</td>
</tr>
<tr>
<td>Standby master state</td>
<td>status of the standby master: active or passive</td>
</tr>
</tbody>
</table>

The following output fields are reported by `gpstate -s` for each segment:

**Table 71: gpstate output data for segments**

<table>
<thead>
<tr>
<th>Output Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hostname</td>
<td>system-configured host name</td>
</tr>
<tr>
<td>Address</td>
<td>network address host name (NIC name)</td>
</tr>
<tr>
<td>Datadir</td>
<td>file system location of segment data directory</td>
</tr>
<tr>
<td>Port</td>
<td>port number of segment database listener process</td>
</tr>
<tr>
<td>Current Role</td>
<td>current role of a segment: Mirror or Primary</td>
</tr>
<tr>
<td>Preferred Role</td>
<td>role at system initialization time: Mirror or Primary</td>
</tr>
</tbody>
</table>
### Output Data

<table>
<thead>
<tr>
<th>Output Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mirror Status</td>
<td>status of a primary/mirror segment pair:</td>
</tr>
<tr>
<td></td>
<td><em>Synchronized</em> = data is up to date on both</td>
</tr>
<tr>
<td></td>
<td><em>Resynchronization</em> = data is currently being copied from one to the other</td>
</tr>
<tr>
<td></td>
<td><em>Change Tracking</em> = segment down and active segment is logging changes</td>
</tr>
<tr>
<td>Change tracking data size</td>
<td>when in <em>Change Tracking</em> mode, the size of the change log file (may grow and shrink as compression is applied)</td>
</tr>
<tr>
<td>Estimated total data to synchronize</td>
<td>when in <em>Resynchronization</em> mode, the estimated size of data left to synchronize</td>
</tr>
<tr>
<td>Data synchronized</td>
<td>when in <em>Resynchronization</em> mode, the estimated size of data that has already been synchronized</td>
</tr>
<tr>
<td>Estimated resync progress with mirror</td>
<td>When in <em>Resynchronization</em> mode, the estimated percentage of completion</td>
</tr>
<tr>
<td>Estimated resync end time</td>
<td>when in <em>Resynchronization</em> mode, the estimated time to complete</td>
</tr>
<tr>
<td>File <em>postmaster.pid</em></td>
<td>status of <em>postmaster.pid</em> lock file: <em>Found</em> or <em>Missing</em></td>
</tr>
<tr>
<td>PID from <em>postmaster.pid</em> file</td>
<td>PID found in the <em>postmaster.pid</em> file</td>
</tr>
<tr>
<td>Lock files in <em>/tmp</em></td>
<td>a segment port lock file for its <em>postgres</em> process is created in <em>/tmp</em> (file is removed when a segment shuts down)</td>
</tr>
<tr>
<td>Active PID</td>
<td>active process ID of a segment</td>
</tr>
<tr>
<td>Master reports status as</td>
<td>segment status as reported in the system catalog: <em>Up</em> or <em>Down</em></td>
</tr>
<tr>
<td>Database status</td>
<td>status of Greenplum Database to incoming requests: <em>Up</em>, <em>Down</em>, or <em>Suspended</em>. A <em>Suspended</em> state means database activity is temporarily paused while a segment transitions from one state to another.</td>
</tr>
</tbody>
</table>

The following output fields are reported by `gpstate -f` for standby master replication status:

**Table 72: gpstate output data for master replication**

<table>
<thead>
<tr>
<th>Output Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standby address</td>
<td>hostname of the standby master</td>
</tr>
<tr>
<td>Standby data dir</td>
<td>file system location of the standby master data directory</td>
</tr>
<tr>
<td>Standby port</td>
<td>port of the standby master <em>postgres</em> database listener process</td>
</tr>
<tr>
<td>Standby PID</td>
<td>process ID of the standby master</td>
</tr>
</tbody>
</table>
Output Data | Description
---|---
Standby status | status of the standby master: Standby host passive
WAL Sender State | write-ahead log (WAL) streaming state: streaming, startup, backup, catchup
Sync state | WAL sender synchronization state: sync
Sent Location | WAL sender transaction log (xlog) record sent location
Flush Location | WAL receiver xlog record flush location
Replay Location | standby xlog record replay location

Examples
Show detailed status information of a Greenplum Database system:

```
gpstate -s
```

Do a quick check for down segments in the master host system catalog:

```
gpstate -Q
```

Show information about mirror segment instances:

```
gpstate -m
```

Show information about the standby master configuration:

```
gpstate -f
```

Display the Greenplum software version information:

```
gpstate -i
```

See Also
`gpstart, gplogfilter`

**gpstop**

Stops or restarts a Greenplum Database system.

Synopsis

```
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 --version
gpstop -? | -h | --help
```
Description

The `gpstop` utility is used to stop the database servers that comprise a Greenplum Database system. When you stop a Greenplum Database system, you are actually stopping several `postgres` database server processes at once (the master and all of the segment instances). The `gpstop` utility handles the shutdown of the individual instances. Each instance is shutdown in parallel.

By default, you are not allowed to shut down Greenplum Database if there are any client connections to the database. Use the `-M fast` option to roll back all in progress transactions and terminate any connections before shutting down. If there are any transactions in progress, the default behavior is to wait for them to commit before shutting down.

With the `-u` option, the utility uploads changes made to the master `pg_hba.conf` file or to `runtime` configuration parameters in the master `postgresql.conf` file without interruption of service. Note that any active sessions will not pickup the changes until they reconnect to the database.

Options

- `-a`
  Do not prompt the user for confirmation.

- `-B parallel_processes`
  The number of segments to stop in parallel. If not specified, the utility will start up to 64 parallel processes depending on how many segment instances it needs to stop.

- `-d master_data_directory`
  Optional. The master host data directory. If not specified, the value set for `$MASTER_DATA_DIRECTORY` will be used.

- `-l logfile_directory`
  The directory to write the log file. Defaults to `~/gpAdminLogs`.

- `-m`
  Optional. Shuts down a Greenplum master instance that was started in maintenance mode.

- `-M fast`
  Fast shut down. Any transactions in progress are interrupted and rolled back.

- `-M immediate`
  Immediate shut down. Any transactions in progress are aborted. This mode kills all `postgres` processes without allowing the database server to complete transaction processing or clean up any temporary or in-process work files.

- `-M smart`
  Smart shut down. If there are active connections, this command fails with a warning. This is the default shutdown mode.

- `-q`
  Run in quiet mode. Command output is not displayed on the screen, but is still written to the log file.

- `-r`
  Restart after shutdown is complete.

- `-t timeout_seconds`
  Specifies a timeout threshold (in seconds) to wait for a segment instance to shutdown. If a segment instance does not shutdown in the specified number of seconds, `gpstop` displays a message indicating that one or more segments are still in the process of shutting down and that you cannot restart Greenplum Database until the segment instance(s) are stopped. This option is useful in situations where `gpstop` is executed and
there are very large transactions that need to rollback. These large transactions can take over a minute to rollback and surpass the default timeout period of 600 seconds.

-u

This option reloads the pg_hba.conf files of the master and segments and the runtime parameters of the postgresql.conf files but does not shutdown the Greenplum Database array. Use this option to make new configuration settings active after editing postgresql.conf or pg_hba.conf. Note that this only applies to configuration parameters that are designated as runtime parameters.

-v

Displays detailed status, progress and error messages output by the utility.

-y

Do not stop the standby master process. The default is to stop the standby master.

-? | -h | --help

Displays the online help.

--version

Displays the version of this utility.

Examples

Stop a Greenplum Database system in smart mode:

gpstop

Stop a Greenplum Database system in fast mode:

gpstop -M fast

Stop all segment instances and then restart the system:

gpstop -r

Stop a master instance that was started in maintenance mode:

gpstop -m

Reload the postgresql.conf and pg_hba.conf files after making configuration changes but do not shutdown the Greenplum Database array:

gpstop -u

See Also

gpstart

gpsys1

Displays information about your operating system.

Synopsis

gpsys1 [ -a | -m | -p ]
gpsys1 -?
gpsys1 --version

Description
gpsys1 displays the platform and installed memory (in bytes) of the current host. For example:

```
linux 1073741824
```

Options
- `--version`
  Displays the version of this utility.
- `-a (show all)`
  Shows both platform and memory information for the current host. This is the default.
- `-m (show memory only)`
  Shows system memory installed in bytes.
- `-p (show platform only)`
  Shows the OS platform. Platform can be `linux`, `darwin` or `sunos5`.
- `-? (help)`
  Displays the online help.

Examples
Show information about the current host operating system:

```
gpsys1
```

See Also
gpcheckperf
gptransfer

The `gptransfer` utility copies objects from databases in a source Greenplum Database system to databases in a destination Greenplum Database system.

Synopsis

```
gptransfer
{
  --full
  { [-d database1 [ -d database2 ... ] ] |
    [-t db.schema.table [ -t db.schemal.table1 ... ] ] |
    [-f table-file [ --partition-transfer
      [ --partition-transfer-non-partition-target ] ]
    [-T db.schema.table [ -T db.schemal.table1 ... ] ]
    [-F table-file] } }
  [ --skip-existing | --truncate | --drop]
  [ --analyze] [ --validate=type ] [ -x ] [ --dry-run ]
  [ --schema-only ]
  [ --source-host=source_host [ --source-port=source_port]
    [ --source-user=source_user] ]
  [ --base-port=base_gpfdist_port]
  [ --dest-host=dest_host --source-map-file=host_map_file ]
  [ --dest-port=port ] [ --dest-user=dest_user ]
}
The `gptransfer` utility copies database objects from a source Greenplum Database system to a destination system. You can perform one of the following types of operations:

- **Copy a Greenplum Database system with the `--full` option.**

  This option copies all user created databases in a source system to a different destination system. If you specify the `--full` option, you must specify both a source and destination system. The destination system cannot contain any user-defined databases, only the default databases postgres, template0, and template1.

- **Copy a set of user defined database tables to a destination system.** The `-f` and `-t` options copy a specified set of user defined tables, table data, and re-creates the table indexes. The `-d` option copies all user defined tables, table data, and re-creates the table indexes from a specified database.

  If the destination system is the same as the source system, you must also specify a destination database with the `--dest-database` option. When you specify a destination database, the source database tables are copied into the specified destination database.

  For partitioned tables, you can specify the `--partition-transfer` or the `--partition-transfer-non-partition-target` option with `-f` option to copy specific leaf child partitions of partitioned tables from a source database. The leaf child partitions are the lowest level partitions of a partitioned database. For the `--partition-transfer` option, the destination tables are leaf child partitions. For the `--partition-transfer-non-partition-target` option, the destination tables are non-partitioned tables.

  If an invalid set of `gptransfer` options are specified, or if a specified source table or database does not exist, `gptransfer` returns an error and quits. No data is copied.

To copy database objects between Greenplum Database systems `gptransfer` utility uses:

- The Greenplum Database utility `gpfdist` on the source database system. The `gpfdists` protocol is not supported.
- Writable external tables on the source database system and readable external tables on the destination database system.
- Named pipes that transfer the data between a writable external table and a readable external table.

When copying data into the destination system, it is redistributed on the Greenplum Database segments of the destination system. This is the flow of data when `gptransfer` copies database data:

writable external table > `gpfdist` > named pipe > `gpfdist` > readable external table

---

```bash
[--dest-database=dest_database_name]  
[--batch-size=batch_size]  
[--sub-batch-size=sub_batch_size]  
[--timeout=seconds]  
[--max-line-length=length]  
[--work-base-dir=work_dir]  
[--format=[CSV|TEXT]]  
[--quote=character]  
[--no-final-count]  

[[-v | --verbose]  
[-q | --quiet]  
[--gpfdist-verbose]  
[--gpfdist-very-verbose]  
[-a]  

gptransfer --version  
gptransfer -h | -? | --help
```
For information about transferring data with `gptransfer`, see "Migrating Data with Gptransfer" in the Greenplum Database Administrator Guide.

**About Database, Schema, and Table Names**

When you transfer an entire database, schema and table names in the database can contain only alphanumeric characters and the underscore character (_). Also, Unicode characters are not supported. The same naming restrictions apply when you specify a database, schema, or table as an option or in a file.

**Notes**

The `gptransfer` utility efficiently transfers tables with large amounts of data. Because of the overhead required to set up parallel transfers, the utility is not recommended for transferring tables with small amounts of data. It might be more efficient to copy the schema and smaller tables to the destination database using other methods, such as the SQL `COPY` command, and then use `gptransfer` to transfer large tables in batches.

When copying database data between different Greenplum Database systems, `gptransfer` requires a text file that lists all the source segment host names and IP addresses. Specify the name and location of the file with the `--source-map-file` option. If the file is missing or not all segment hosts are listed, `gptransfer` returns an error and quits. See the description of the option for file format information.

The source and destination Greenplum Database segment hosts need to be able to communicate with each other. To ensure that the segment hosts can communicate, you can use a tool such as the Linux `netperf` utility.

If a filespace has been created for a source Greenplum Database system, a corresponding filespace must exist on the target system.

SSH keys must be exchanged between the two systems before using `gptransfer`. The `gptransfer` utility connects to the source system with SSH to create the named pipes and start the `gpfdist` instances. You can use the Greenplum Database `gpssh-exkeys` utility with a list of all the source and destination primary hosts to exchange keys between Greenplum Database hosts.

Source and destination systems must be able to access the `gptransfer` work directory. The default directory is the user's home directory. You can specify a different directory with the `--work-base-dir` option.

The `gptransfer` utility logs messages in the `~/gpAdminLogs` directory on the master host. `gptransfer` creates a log file with the name `gptransfer_date.log` and appends messages to it each time it runs on that day. You can specify a different directory for the log file using the `-l log_directory` option.

Work directories named `~/gptransfer_process_id` are created on segment hosts in the source cluster. Log files for the `gpfdist` instances that `gptransfer` creates are in these directories. Adding the `--gpfdist-verbose` or `--gpfdist-very-verbose` options to the `gptransfer` command line increases the `gpfdist` logging level.

The `gptransfer` utility does not move configuration files such as `postgres.conf` and `pg_hba.conf`. You must set up the destination system configuration separately.

The `gptransfer` utility does not move external objects such as Greenplum Database extensions, third party jar files, and shared object files. You must install the external objects separately.

The `gptransfer` utility does not move dependent database objects unless you specify the `--full` option. For example, if a table has a default value on a column that is a user-defined function, that function must exist in the destination system database when using the `-t`, `-d`, or `-f` options.

If you move a set of database tables with the `-d`, `-t`, or `-f` option, and the destination table or database does not exist, `gptransfer` creates it. The utility re-creates any indexes on tables before copying data.
If a table exists on the destination system and one of the options `--skip-existing`, `--truncate`, or `--drop` is not specified, `gptransfer` returns an error and quits.

If an error occurs when during the process of copying a table, or table validation fails, `gptransfer` continues copying the other specified tables. After `gptransfer` finishes, it displays a list of tables where an error occurred, writes the names of tables that failed into a text file, and then prints the name of the file. You can use this file with the `gptransfer -f` option to retry copying tables.

The name of the file that contains the list of tables where errors occurred is `failed_migrated_tables_yyyymmdd_hhmmss.txt`. The `yyyymmdd_hhmmss` is a timestamp when the `gptransfer` process was started. The file is created in the directory where `gptransfer` is executed.

After `gptransfer` completes copying database objects, the utility compares the row count of each table copied to the destination databases with the table in the source database. The utility returns the validation results for each table. You can disable the table row count validation by specifying the `--no-final-count` option.

**Note:** If the number of rows do not match, the table is not added to the file that lists the tables where transfer errors occurred.

The `gp_external_max_segs` server configuration parameter controls the number of segment instances that can access a single `gpfdist` instance simultaneously. Setting a low value might affect `gptransfer` performance. For information about the parameter, see the *Greenplum Database Reference Guide*.

**Limitation for the Source and Destination Systems**

If you are copying data from a system with a larger number of segments to a system with a fewer number of segment hosts, then the total number of primary segments on the destination system must be greater than or equal to the total number of segment hosts on the source system.

For example, assume a destination system has a total of 24 primary segments. This means that the source system cannot have more than 24 segment hosts.

When you copy data from a source Greenplum Database system with a larger number of primary segment instances than on the destination system, the data transfer might be slower when compared to a transfer where the source system has fewer segment instances than the destination system. The `gptransfer` utility uses a different configuration of named pipes and `gpfdist` instances in the two situations.

**Options**

-a

Quiet mode, do not prompt the user for confirmation.

--analyze

Run the `ANALYZE` command on non-system tables. The default is to not run the `ANALYZE` command.

--base-port=base_gpfdist_port

Base port for `gpfdist` on source segment systems. If not specified, the default is 8000.

--batch-size=batch_size

Sets the maximum number of tables that `gptransfer` concurrently copies to the destination database. If not specified, the default is 2. The maximum is 10.

**Note:** If the order of the transfer is important, specify a value of 1. The tables are transferred sequentially based on the order specified in the `-t` and `-f` options.

-d database

A source database to copy. This option can be specified multiple times to copy multiple databases to the destination system. All the user defined tables and table data are copied to the destination system.
A set of databases can be specified using the Python regular expression syntax. The regular expression pattern must be enclosed in slashes (/RE_pattern/). If you use a regular expression, the name must be enclosed in double quotes (" "). This example –d "demo/. */" specifies all databases in the Greenplum Database installation that begin with demo.

**Note:** Note the following two examples for the -d option are equivalent. They both specify a set of databases that begins with demo and ends with zero or more digits.

```
-d "demo/[0-9]*/"
-d "/demo[0-9]*/"
```

If the source database does not exist, gptransfer returns an error and quits. If a destination database does not exist a database is created.

Not valid with the --full, -f, -t, --partition-transfer, or --partition-transfer-non-partition-target options.

Alternatively, specify the -t or -f option to copy a specified set of tables.

**--delimiter=** *delim*

Delimiter to use for writable external tables created by gptransfer. Specify a single ASCII character that separates columns within each row of data. The default value is a comma (,). If *delim* is a comma (,) or if this option is not specified, gptransfer uses the CSV format for writable external tables. Otherwise, gptransfer uses the TEXT format.

If --delimiter, --format, and --quote options are not specified, these are settings for writable external tables:

```
FORMAT 'CSV' ( DELIMITER ',', QUOTE E'\001' )
```

You can specify a delimiter character such as a non-printing character with the format "
digits" (octal). A backslash followed by the octal value for the character. The octal format must be enclosed in double quotes. This example specifies the octal character \001, the SOH character:

```
--delimiter="\001"
```

**--dest-database=** *dest_database_name*

The database in the destination Greenplum Database system. If not specified, the source tables are copied into a destination system database with the same name as the source system database.

This option is required if the source and destination Greenplum Database systems are the same.

If destination database does not exist, it is created.

Not valid with the --full, --partition-transfer, or --partition-transfer-non-partition-target options.

**--dest-host=** *dest_host*

Destination Greenplum Database hostname or IP address. If not specified, the default is the host the system running gptransfer (127.0.0.1)

**--dest-port=** *dest_port*

Destination Greenplum Database port number. If not specified, the default is 5432.

**--dest-user=** *dest_user*

User ID that is used to connect to the destination Greenplum Database system. If not specified, the default is the user gpadmin.
- **--drop**
  Specify this option to drop the table that is in the destination database if it already exists. Before copying table data, `gptransfer` drops the table and creates it again.
  At most, only one of the options can be specified `--skip-existing`, `--truncate`, or `--drop`. If one of them is not specified and the table exists in the destination system, `gptransfer` returns an error and quits.
  Not valid with the `--full`, `--partition-transfer`, or `--partition-transfer-non-partition-target` options.

- **--dry-run**
  When you specify this option, `gptransfer` generates a list of the migration operations that would have been performed with the specified options. The data is not migrated.
  The information is displayed at the command line and written to the log file.

- **-f table-file**
  The location and name of file containing list of fully qualified table names to copy from the Greenplum Database source system. In the text file, you specify a single fully qualified table per line (`database.schema.table`).
  A set of tables can be specified using the Python regular expression syntax. See the `-d` option for information about using regular expressions.
  If the source table does not exist, `gptransfer` returns an error and quits. If the destination database or table does not exist, it is created.
  Only the table and table data are copied and indexes are re-created. Dependent objects are not copied.
  You cannot specify views, or system catalog tables. The `--full` option copies user defined views.
  If you specify the `-d` option to copy all the tables from a database, you cannot specify individual tables from the database.
  Not valid with the `--full`, `-d`, or `-t` options.

- **--partition-transfer (partitioned destination table)**
  Specify this option with the `-f` option to copy data from leaf child partition tables of partitioned tables from a source database to the leaf child partition tables in a destination database. The text file specified by the `-f` option contains a list of fully qualified leaf child partition table names with this syntax.

```plaintext
src_db.src_schema.src_prt_tbl[, dst_db.dst_schema.dst_prt_tbl]
```

Wildcard characters are not supported in the fully qualified table names. The destination partitioned table must exist. If the destination leaf child partition table is not specified in the file, `gptransfer` copies the data to the same fully qualified table name (`db_name.schema.table`) in the destination Greenplum Database system. If the source and destination Greenplum Database systems are the same, you must specify a destination table where at least one of the following must be different between the source and destination table: `db_name`, `schema`, or `table`.

If either the source or destination table is not a leaf child partition, the utility returns an error and no data are transferred.

These characteristics must be the same for the partitioned table in the source and destination database.

- Number of table columns and the order of the column data types (the source and destination table names and table column names can be different)
- Partition level of the specified source and destination tables
Partitioning criteria of the specified source and destination leaf child partitions and child partitions above them in the hierarchy (partition type and partition column)

This option is not valid with these options: -d, --dest-database, --drop, -F, --full, --schema-only, -T, -t.

Note: If a destination table is not empty or the data in the source or destination table changes during a transfer operation (rows are inserted or deleted), the table row count validation fails due to row count mismatch.

If the destination table is not empty, you can specify the --truncate option to truncate the table before the transfer operation.

You can specify the -x option to acquire exclusive locks on the tables during a transfer operation.

--partition-transfer-non-partition-target (non-partitioned destination table)

Specify this option with the -f option to copy data from leaf child partition tables of partitioned tables in a source database to non-partitioned tables in a destination database. The text file specified by the -f option contains a list of fully qualified leaf child partition table names in the source database and non-partitioned tables names in the destination database with this syntax.

```
src_db.src_schema.src_part_tbl, dest_db.dest_schema.dest_tbl
```

Wildcard characters are not supported in the fully qualified table names. The destination tables must exist, and both source and destination table names are required in the file.

If a source table is not a leaf child partition table or a destination table is not a normal (non-partitioned) table, the utility returns an error and no data are transferred.

If the source and destination Greenplum Database systems are the same, you must specify a destination table where at least one of the following must be different between the source and destination table: db_name, schema, or table.

For the partitioned table in the source database and the table in the destination database, the number of table columns and the order of the column data types must be the same (the source and destination table column names can be different).

The same destination table can be specified in the file for multiple source leaf child partition tables that belong to a single partitioned table. Transferring data from source leaf child partition tables that belong to different partitioned tables to a single non-partitioned table is not supported.

This option is not valid with these options: -d, --dest-database, --drop, -F, --full, --schema-only, -T, -t, --truncate, --validate.

Note: If the data in the source or destination table changes during a transfer operation (rows are inserted or deleted), the table row count validation fails due to row count mismatch.

You can specify the -x option to acquire exclusive locks on the tables during a transfer operation.

-F table-file

The location and name of file containing list of fully qualified table names to exclude from transferring to the destination system. In the text file, you specify a single fully qualified table per line.

A set of tables can be specified using the Python regular expression syntax. See the -d option for information about using regular expressions.
The utility removes the excluded tables from the list of tables that are being transferred to
the destination database before starting the transfer. If excluding tables results in no tables
being transferred, the database or schema is not created in the destination system.

If a source table does not exist, `gptransfer` displays a warning.

Only the specified tables are excluded. To exclude dependent objects, you must explicitly
specify them.

You cannot specify views, or system catalog tables.

Not valid with the `--full`, `--partition-transfer`, or `--partition-transfer-
non-partition-target` options.

You can specify the `--dry-run` option to test the command. The `-v` option, displays and
logs the excluded tables.

`--format=[CSV | TEXT]`

Specify the format of the writable external tables that are created by `gptransfer` to
transfer data. Values are `CSV` for comma separated values, or `TEXT` for plain text. The
default value is `CSV`.

If the options `--delimiter`, `--format`, and `--quote` are not specified, these are
default settings for writable external tables:

```
FORMAT 'CSV' ( DELIMITER ',' QUOTE E'\001' )
```

If you specify `TEXT`, you must also specify a non-comma delimiter with the `--
delimiter=delim` option. These are settings for writable external tables:

```
FORMAT 'TEXT' ( DELIMITER delim ESCAPE 'off' )
```

`--full`

Full migration of a Greenplum Database source system to a destination system. You must
specify the options for the destination system, the `--source-map-file` option, the `--
dest-host` option, and if necessary, the other destination system options.

The `--full` option cannot be specified with the `-t`, `-d`, `-f`, `--partition-transfer`, or
`--partition-transfer-non-partition-target` options.

A full migration copies all database objects including, tables, indexes, views, users, roles,
functions, and resource queues for all user defined databases. The default databases,
postgres, template0 and template1 are not moved.

If a database exists in the destination system, besides the default postgres, template0 and
template1 databases, `gptransfer` returns an error and quits.

**Note:** The `--full` option is recommended only when the databases contain
a large number of tables with large amounts of data. Because of the overhead
required to set up parallel transfers, the utility is not recommended when the
databases contain tables with small amounts of data. For more information,
see Notes.

`--gpfdist-verbose`

Set the logging level for `gpfdist` processes to verbose (`-v`). Cannot be specified with `--
gpfdist-very-verbose`.

The output is recorded in `gpfdist` log files in the `~/gptransfer_process_id` directory
on segment hosts in the source Greenplum Database cluster.

`--gpfdist-very-verbose`

Set the logging level for `gpfdist` processes to very verbose (`-V`). Cannot be specified
with `--gpfdist-verbose`.

The output is recorded in `gpfdist` log files in the `~/gptransfer_process_id` directory
on segment hosts in the source Greenplum Database cluster.
-l log_dir
  Specify the gptransfer log file directory. If not specified, the default is ~/gpAdminLogs. This directory is created on the master host in the source Greenplum cluster.

--max-line-length=length
  Sets the maximum allowed data row length in bytes for the gpfdist utility. If not specified, the default is 10485760. Valid range is 32768 (32K) to 268435456 (256MB).
  Should be used when user data includes very wide rows (or when line too long error message occurs). Should not be used otherwise as it increases resource allocation.

--no-final-count
  Disable table row count validation that is performed after gptransfer completes copying database objects to the target database. The default is to compare the row count of tables copied to the destination databases with the tables in the source database.

-q | --quiet
  If specified, suppress status messages. Messages are only sent to the log file.

--quote=character
  The quotation character when gptransfer creates writable external tables with the CSV format. Specify a single ASCII character that is used to enclose column data. The default value is the octal character \001, the SOH character.
  You can specify a delimiter character such as a non-printing character with the format "\digits" (octal). A backslash followed by the octal value for the character. The octal value must be enclosed in double quotes.

--schema-only
  Create only the schemas specified by the command. Data is not transferred.
  If specified with the --full option, gptransfer replicates the complete database schema, including all tables, indexes, views, user defined types (UDT), and user defined functions (UDF) for the source databases. No data is transferred.
  If you specify tables with the -t or -f option with --schema-only, gptransfer creates only the tables and indexes. No data is transferred.
  Not valid with the --partition-transfer, --partition-transfer-non-partition-target, or --truncate options.

  Note: Because of the overhead required to set up parallel transfers, the --schema-only option is not recommended when transferring information for a large number of tables. For more information, see Notes.

--skip-existing
  Specify this option to skip copying a table from the source database if the table already exists in the destination database.
  At most, only one of the options can be specified --skip-existing, --truncate, or --drop. If one of them is not specified and the table exists in the destination system, gptransfer returns an error and quits.
  Not valid with the --full option.

--source-host=source_host
  Source Greenplum Database host name or IP address. If not specified, the default host is the system running gptransfer (127.0.0.1).

--source-map-file=host_map_file
  File that lists source segment host name and IP addresses. If the file is missing or not all segment hosts are listed, gptransfer returns an error and quits.
Each line of the file contains a source host name and the host IP address separated by a comma: hostname,IPaddress. This example lists four Greenplum Database hosts and their IP addresses.

```
sdw1,192.0.2.1
sdw2,192.0.2.2
sdw3,192.0.2.3
sdw4,192.0.2.4
```

This option is required if the --full option is specified or if the source Greenplum Database system is different than the destination system. This option is not required if source and destination systems are the same.

`--source-port=source_port`
Source Greenplum Database port number. If not specified, the default is 5432.

`--source-user=source_user`
User ID that is used to connect to the source Greenplum Database system. If not specified, the default is the user gpadmin.

`--sub-batch-size=sub_batch_size`
Specifies the maximum degree of parallelism of the operations performed when migrating a table such as starting gpfdist instances, creating named pipes for the move operations. If not specified, the default is 25. The maximum is 50.

Specify the --batch-size option to control the maximum number of tables that gptransfer concurrently processes.

`-t db.schema.table`
A table from the source database system to copy. The fully qualified table name must be specified.

A set of tables can be specified using the Python regular expression syntax. See the -d option for information about using regular expressions.

If the destination table or database does not exist, it is created. This option can be specified multiple times to include multiple tables. Only the table and table data are copied and indexes are re-created. Dependent objects are not copied.

If the source table does not exist, gptransfer returns an error and quits.

If you specify the -d option to copy all the tables from a database, you do not need to specify individual tables from the database.

**Not valid with the --full, --d, --f, --partition-transfer, or --partition-transfer-non-partition-target options.**

`-T db.schema.table`
A table from the source database system to exclude from transfer. The fully qualified table name must be specified.

A set of tables can be specified using the Python regular expression syntax. See the -d option for information about using regular expressions.

This option can be specified multiple times to include multiple tables. Only the specified tables are excluded. To exclude dependent objects, you must explicitly specify them.

The utility removes the excluded tables from the list of tables that are being transferred to the destination database before starting the transfer. If excluding tables results in no tables being transferred, the database or schema is not created in the destination system.

If a source table does not exist, gptransfer displays a warning.

**Not valid with the --full, --partition-transfer, or --partition-transfer-non-partition-target options.**
You can specify the --dry-run option to test the command. The -v option displays and logs the excluded tables.

--timeout seconds
Specify the time out value in seconds that gptransfer passes the gpfdist processes that gptransfer uses. The value is the time allowed for Greenplum Database to establish a connection to a gpfdist process. You might need to increase this value when operating on high-traffic networks.

The default value is 300 seconds (5 minutes). The minimum value is 2 seconds, the maximum value is 600 seconds.

--truncate
Specify this option to truncate the table that is in the destination database if it already exists.

At most, only one of the options can be specified --skip-existing, --truncate, or --drop. If one of them is not specified and the table exists in the destination system, gptransfer returns an error and quits.

Not valid with the --full option.

--validate=type
Perform data validation on table data. These are the supported types of validation.

  count - Specify this value to compare row counts between source and destination table data.

  MD5 - Specify this value to compare MD5 values between source and destination table data.

If validation for a table fails, gptransfer displays the name of the table and writes the file name to the text file failed_migrated_tables_yyyyMMdd_hhmmss.txt. TheyyyyMMdd_hhmmss is a time stamp when the gptransfer process was started. The file is created in the directory where gptransfer is executed.

  Note: The file contains the table names where validation failed or other errors occurred during table migration.

-v | --verbose
If specified, sets the logging level to verbose. Additional log information is written to the log file and the command line during command execution.

--work-base-dir=work_dir
Specify the directory that gptransfer uses to store temporary working files such as PID files and named pipes. The default directory is the user's home directory.

Source and destination systems must be able to access the gptransfer work directory.

-x
Acquire an exclusive lock on tables during the migration to prevent insert or updates.

On the source database, an exclusive lock is acquired when gptransfer inserts into the external table and is released after validation.

On the destination database, an exclusive lock is acquired when gptransfer selects from external table and released after validation.

If -x option is not specified and --validate is specified, validation failures occur if data is inserted into either the source or destination table during the migration process. The gptransfer utility displays messages if validation errors occur.

-h | -? | --help
Displays the online help.
--version

Displays the version of this utility.

Examples

This command copies the table public.t1 from the database db1 and all tables in the database db2 to the system mytest2.

```
gptransfer -t db1.public.t1 -d db2 --dest-host=mytest2 \ 
--source-map-file=gp-source-hosts --truncate
```

If the databases db1 and db2 do not exist on the system mytest2, they are created. If any of the source tables exist on the destination system, gptransfer truncates the table and copies the data from the source to the destination table.

This command copies leaf child partition tables from a source system to a destination system.

```
gptransfer -f input_file --partition-transfer --source-host=source_host \ 
--source-user=source_user --source-port=source_port --dest-host=dest_host \ 
--dest-user=dest_user --dest-port=dest_port --source-map-file=host_map_file
```

This line in input_file copies a leaf child partition from the source system to the destination system.

```
srcdb.people.person_1_prt_experienced, destdb.public.employee_1_prt_seniors
```

The line assumes partitioned tables in the source and destination systems similar to the following tables.

- In the people schema of the srcdb database of the source system, a partitioned table with a leaf child partition table person_1_prt_experienced. This CREATE TABLE command creates a partitioned table with the leaf child partition table.

```
CREATE TABLE person(id int, title char(1))
    DISTRIBUTED BY (id)
    PARTITION BY list (title)
    (PARTITION experienced VALUES ('S'),
     PARTITION entry_level VALUES ('J'),
     DEFAULT PARTITION other);
```

- In the public schema of the destdb database of the source system, a partitioned table with a leaf child partition table public.employee_1_prt_seniors. This CREATE TABLE command creates a partitioned table with the leaf child partition table.

```
CREATE TABLE employee(id int, level char(1))
    DISTRIBUTED BY (id)
    PARTITION BY list (level)
    (PARTITION seniors VALUES ('S'),
     PARTITION juniors VALUES ('J'),
     DEFAULT PARTITION other);
```

This example uses Python regular expressions in a filter file to specify the set of tables to transfer. This command specifies the -f option with the filter file /tmp/filter_file to limit the tables that are transferred.

```
gptransfer -f /tmp/filter_file --source-port 5432 --source-host test4 \ 
--source-user gpadmin --dest-user gpadmin --dest-port 5432 --dest-host test1 \ 
--source-map-file /home/gpadmin/source_map_file
```
This is the contents of /tmp/filter_file.

"test1.arc/.*/.*/
"test1.c/(..)/y/.*/"

In the first line, the regular expressions for the schemas, arc/.*/, and for the tables, /.*/, limit the transfer to all tables with the schema names that start with arc.

In the second line, the regular expressions for the schemas, c/(..)/y, and for the tables, /.*/, limit the transfer to all tables with the schema names that are four characters long and that start with c and end with y, for example, crty.

When the command is run, tables in the database test1 that satisfy either condition are transferred to the destination database.

See Also

gpfdist

For information about loading and unloading data, see the Greenplum Database Administrator Guide.

**pgbouncer**

Manages database connection pools.

**Synopsis**

```
pgbouncer [OPTION ...] config.ini
```

<table>
<thead>
<tr>
<th>OPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -d</td>
</tr>
<tr>
<td>[ -R</td>
</tr>
<tr>
<td>[ -q</td>
</tr>
<tr>
<td>[ -v</td>
</tr>
<tr>
<td>[ {-u</td>
</tr>
</tbody>
</table>

```
pgbouncer [ -V | --version ] | [ -h | --help ]
```

**Description**

PgBouncer is a light-weight connection pool manager for Greenplum and PostgreSQL databases. Databases may be on different Greenplum Database clusters or PostgreSQL backends. PgBouncer creates a pool for each database user and database combination. A pooled connection can only be reused for another connection request for the same user and database. The client application connects to the connection pool's host and port instead of the Greenplum Database master host and port. PgBouncer either creates a new database connection for the client or reuses an existing connection. When the client disconnects, the connection is returned to the pool for re-use.

PgBouncer supports the standard connection interface that PostgreSQL and Greenplum Database share. A client requesting a database connection provides the host name and port where PgBouncer is running, as well as the database name, username, and password. PgBouncer looks up the requested database (which may be an alias for the actual database) in its configuration file to find the host name, port, and database name for the database connection. The configuration file entry also determines how to authenticate the user and what database role will be used for the connection—a “forced user” can override the username provided with the client's connection request.

PgBouncer requires an authentication file, a text file that contains a list of users and passwords. Passwords may be either clear text, MD5-encoded, or an LDAP/AD lookup string. You can also set up PgBouncer to query the pg_shadow table in the destination database for users that are not in the authentication file.
PgBouncer shares connections in one of three pool modes:

- **Session pooling** – When a client connects, a connection is assigned to it as long as it remains connected. When the client disconnects, the connection is placed back into the pool.
- **Transaction pooling** – A connection is assigned to a client for the duration of a transaction. When PgBouncer notices the transaction is done, the connection is placed back into the pool. This mode can be used only with applications that do not use features that depend upon a session.
- **Statement pooling** – Statement pooling is like transaction pooling, but multi-statement transactions are not allowed. This mode is intended to enforce autocommit mode on the client and is targeted for PL/Proxy on PostgreSQL.

A default pool mode can be set for the PgBouncer instance and the mode can be overridden for individual databases and users.

By connecting to a virtual `pgbouncer` database, you can monitor and manage PgBouncer using SQL-like commands. Configuration parameters can be changed without having to restart PgBouncer, and the configuration file can be reloaded to pick up changes.

PgBouncer does not yet support SSL connections. If you want to encrypt traffic between clients and PgBouncer, you can use `stunnel`, a free software utility that creates TLS-encrypted tunnels using the OpenSSL cryptography library. See “Securing PgBouncer Connections with stunnel” in the Greenplum Database Administrator Guide for directions.

See the *PgBouncer FAQ* for additional usage information.

This reference topic includes the following additional reference information:

- PgBouncer Configuration File
- PgBouncer Authentication File Format
- PgBouncer Administration Console Commands

### Options

- `-d` | `--daemon`  
  Run PgBouncer as a daemon (a background process). The default is to run as a foreground process.

  PgBouncer displays start up messages when starting as a daemon. To disable the display of messages add the `-q` option.

  To shut down a PgBouncer daemon, log in to the administration console and issue the `SHUTDOWN` command.

  **Note:** This option does not work on Windows servers.

- `-R` | `--restart`  
  Restart PgBouncer using the specified command line arguments. Non-TLS connections to databases are maintained during restart; TLS connections are dropped.

  If you specify only the `-R` option, PgBouncer displays log information on the command line after restart. To restart PgBouncer as a daemon specify the options `-Rd`.

  **Note:** Works only if the operating system supports Unix sockets and the PgBouncer configuration has no `unix_socket_dir`. This option does not work on Windows servers.

- `-q` | `--quiet`  
  Run quietly. Do not display messages on the command line (stdout).

- `-v` | `--verbose`  
  Increase message verbosity. Display additional messages. Can be used multiple times.

- `{ -u | --user }=username`
The PgBouncer process assumes the identity of `username`.

```
- V | --version
    Show version and exit.
- h | --help
    Show help message and exit.
```

**PgBouncer Configuration File**

The PgBouncer configuration file (usually `pgbouncer.ini`) is in the "ini" format. Section names are enclosed in square braces (`[ ]`). Lines beginning with `;` or `#` are comments and are ignored. The characters `;` and `#` are not recognized when they appear later in the line.

**Synopsys**

```
[databases]
db = ...

[pgbouncer]
...

[users]
...
```

**Description**

A PgBouncer configuration file has up to three sections:

- **[databases] Section**
- **[pgbouncer] Section**
- **[users] Section**

**[databases] Section**

The databases section contains key=value pairs, where the key is a database name and the value is a libpq connect-string list of key=value pairs.

A database name can contain characters `[0-9A-Za-z_.-]` without quoting. Names that contain other chars must be quoted with standard SQL identifier quoting

- Enclose names in double quotes ("")
- Represent a double-quote within an identifier with two consecutive double quote characters

The database name "*" is the fallback database. The value for this key is a connect string for the requested database. Automatically created database entries like these are cleaned up if they remain idle longer than the time specified in `autodb_idle_timeout` parameter.

The PgBouncer configuration file can contain `%include` directives, which specify another file to read and process. This allows splitting the configuration file into separate parts. For example:

```
%include filename
```

**Location Parameters**

The following parameters may be included in the value to specify the location of the database.

**dbname**

The destination database name.
Default: same as the client-side database name.

**host**
The name or IP address of the Greenplum master host. Host names are resolved at connect time. If DNS returns several results, they are used in a round-robin manner. The DNS result is cached and the `dns_max_ttl` parameter determines when the cache entry expires.

Default: not set, means the connection is made through a Unix socket.

**port**
The Greenplum Database master port. Default: 5432

**user, password**
If `user=` is set, all connections to the destination database are made with the specified user. This means there will be just one pool for the database.

If the `user=` parameter is not set, PgBouncer attempts to log in to the destination database with the user name passed by the client. This means there will be one pool for each user who connects to the database.

**auth_user**
If `auth_user` is set, any user who is not specified in `auth_file` is authenticated by querying the `pg_shadow` table in the database as the `auth_user`. The `auth_user` password must be set in the `auth_file`.

### Pool Configuration

**pool_size**
Set maximum size of pools for this database. If not set, the `default_pool_size` is used.

**connect_query**
Query to be executed after a connection is established, but before allowing the connection to be used by any clients. If the query raises errors, they are logged but ignored otherwise.

**pool_mode**
Set the pool mode for this database. If not set, the default `pool_mode` is used.

**max_db_connections**
Set a database-wide maximum number of connections for this database. The total number of connections for all pools for this database will not exceed this value.

### Extra Parameters

The following parameters allow setting default parameters on server connections.

Note that since version 1.1 PgBouncer tracks client changes for their values, so their use in `pgbouncer.ini` is deprecated now.

**client_encoding**
Ask specific `client_encoding` from server.

**datestyle**
Ask specific `datestyle` from server.

**timezone**
Ask specific `timezone` from server.

#### [pgbouncer] Section

**logfile**
Specifies the location of the log file. The log file is kept open. After log rotation execute `kill -HUP` or run the `RELOAD;` command in the PgBouncer Administrative Console.

Default: not set.

**Note:** On Windows machines, the service must be stopped and started.

**pidfile**

The name of the pid file. Without a pidfile, PgBouncer cannot be run as a background process (daemon).

Default: not set.

**listen_addr**

A list of interface addresses where PgBouncer listens for TCP connections. You may also use `*`, which means to listen on all interfaces. If not set, only Unix socket connections are allowed.

Addresses can be specified numerically (IPv4/IPv6) or by name.

Default: not set

**listen_port**

Which port to listen on. Applies to both TCP and Unix sockets.

Default: 6432

**unix_socket_dir**

Specifies location for Unix sockets. Applies to both listening socket and server connections. If set to an empty string, Unix sockets are disabled. Required for online reboot (`-R` option) to work.

**Note:** Not supported on Windows machines.

Default: `/tmp`

**unix_socket_mode**

Filesystem mode for Unix socket.

Default: 0777

**unix_socket_group**

Group name to use for Unix socket.

Default: not set

**user**

If set, specifies the Unix user to change to after startup. This only works if PgBouncer is started as root or if `user` is the same as the current user. **Note:** Not supported on Windows machines.

Default: not set

**auth_file**

The name of the file containing the user names and passwords to load. The file format is the same as the Greenplum Database `pg_auth/pg_pwd` file, so this parameter can be set to one of those backend files. See Authentication File Format for details.

Default: not set

**auth_type**

How to authenticate users.

**cert**
Clients must connect with TLS using a valid client certificate. The client's username is taken from CommonName field in the certificate.

**md5**

Use MD5-based password check. `auth_file` may contain both MD5-encrypted or plain-text passwords. This is the default authentication method.

**plain**

Clear-text password is sent over wire. *Deprecated*.

**trust**

No authentication is done. The username must still exist in the `auth_file`.

**any**

Like the `trust` method, but the username supplied is ignored. Requires that all databases are configured to log in with a specific user. Additionally, the console database allows any user to log in as admin.

**auth_query**

Query to load a user's password from database. If a user does not exist in the `auth_file` and the database entry includes an `auth_user`, this query is run in the database as `auth_user` to lookup up the user.

Default: `SELECT usename, passwd FROM pg_shadow WHERE usename=$1`

**pool_mode**

Specifies when a server connection can be reused by other clients.

**session**

Connection is returned to the pool when the client disconnects. Default.

**transaction**

Connection is returned to the pool when the transaction finishes.

**statement**

Connection is returned to the pool when the current query finishes. Long transactions with multiple statements are disallowed in this mode.

**max_client_conn**

Maximum number of client connections allowed. When increased then the file descriptor limits should also be increased. The actual number of file descriptors used is more than `max_client_conn`. The theoretical maximum used, when each user connects with its own username to the server is:

```
max_client_conn + (max_pool_size * total_databases * total_users)
```

If a database user is specified in the connect string, all users connect using the same username. Then the theoretical maximum connections is:

```
max_client_conn + (max_pool_size * total_databases)
```

The theoretical maximum should be never reached, unless someone deliberately crafts a load for it. Still, it means you should set the number of file descriptors to a safely high number. Search for `ulimit` in your operating system documentation.

**Note:** `ulimit` does not apply in a Windows environment.

Default: 100

**default_pool_size**

The number of server connections to allow per user/database pair. This can be overridden in the per-database configuration.
Default: 20

**min_pool_size**
Add more server connections to the pool when it is lower than this number. This improves behavior when the usual load drops and then returns suddenly after a period of total inactivity.
Default: 0 (disabled)

**reserve_pool_size**
The number of additional connections to allow for a pool. 0 disables.
Default: 0 (disabled)

**reserve_pool_timeout**
If a client has not been serviced in this many seconds, PgBouncer enables use of additional connections from reserve pool. 0 disables.
Default: 5.0

**max_db_connections**
The maximum number of connections per database. If you hit the limit, closing a client connection to one pool does not immediately allow a server connection to be established for another pool, because the server connection for the first pool is still open. Once the server connection closes (due to idle timeout), a new server connection will be opened for the waiting pool.
Default: unlimited

**max_user_connections**
The maximum number of connections per-user. When you hit the limit, closing a client connection to one pool does not immediately allow a connection to be established for another pool, because the connection for the first pool is still open. After the connection for the first pool has closed (due to idle timeout), a new server connection is opened for the waiting pool.

**server_round_robin**
By default, PgBouncer reuses server connections in LIFO (last-in, first-out) order, so that a few connections get the most load. This provides the best performance when a single server serves a database. But if there is TCP round-robin behind a database IP, then it is better if PgBouncer also uses connections in that manner to achieve uniform load.
Default: 0

**ignore_startup_parameters**
By default, PgBouncer allows only parameters it can keep track of in startup packets: client_encoding, datestyle, timezone, and standard_conforming_strings. All others parameters raise an error. To allow other parameters, specify them here so that PgBouncer can ignore them.
Default: empty

**disable_pqexec**
Disable Simple Query protocol (PQexec). Unlike Extended Query protocol, Simple Query protocol allows multiple queries in one packet, which allows some classes of SQL-injection attacks. Disabling it can improve security. This means that only clients that exclusively use Extended Query protocol will work.
Default: 0

**application_name_add_host**
Add the client host address and port to the application name setting set on connection start. This helps in identifying the source of bad queries. The setting is overwritten without detection if the application executes `SET APPLICATION_NAME` after connecting.

Default: 1

### Log Settings

**syslog**

Toggles syslog on and off. On Windows, eventlog is used instead.

Default: 0

**syslog_ident**

Under what name to send logs to syslog.

Default: `pgbouncer`

**syslog_facility**

Under what facility to send logs to syslog. Some possibilities are: `auth`, `authpriv`, `daemon`, `user`, `local0-7`

Default: `daemon`

**log_connections**

Log successful logins.

Default: 1

**log_disconnections**

Log disconnections, with reasons.

Default: 1

**log_pooler_errors**

Log error messages that the pooler sends to clients.

Default: 1

**stats_period**

How often to write aggregated statistics to the log.

Default: 60

### Console Access Control

**admin_users**

Comma-separated list of database users that are allowed to connect and run all commands on console. Ignored when `auth_mode=any`, in which case any username is allowed in as admin.

Default: empty

**stats_users**

Comma-separated list of database users that are allowed to connect and run read-only queries on console. That means all SHOW commands except SHOW FDS.

Default: empty

### Connection Sanity Checks, Timeouts

**server_reset_query**

Query sent to server on connection release, before making it available to other clients. At that moment no transaction is in progress so it should not include `ABORT` or `ROLLBACK`.
A good choice for Postgres 8.2 and below, and Greenplum Database, is:

```
server_reset_query = RESET ALL; SET SESSION AUTHORIZATION DEFAULT;
```

For Postgres 8.3 and above, the following is sufficient:

```
server_reset_query = DISCARD ALL;
```

When transaction pooling is used, the `server_reset_query` should be empty, as clients should not use any session features. If clients do use session features, they will be broken because transaction pooling does not guarantee that the next query will run on the same connection.

Default: `RESET ALL; SET SESSION AUTHORIZATION DEFAULT;`

### `server_reset_query_always`

Whether `server_reset_query` should be run in all pooling modes. When this setting is off (default), the `server_reset_query` will be run only in pools that are in session pooling mode. Connections in transaction pooling mode should not have any need for reset query.

Default: 0

### `server_check_delay`

How long to keep released connections available for re-use without running sanity-check queries on it. If 0 then the query is run always.

Default: 30.0

### `server_check_query`

A simple do-nothing query to test the server connection.

If an empty string, then sanity checking is disabled.

Default: `SELECT 1;`

### `server_lifetime`

The pooler tries to close server connections that have been connected longer than this number of seconds. Setting it to 0 means the connection is to be used only once, then closed.

Default: 3600.0

### `server_idle_timeout`

If a server connection has been idle more than this many seconds it is dropped. If this parameter is set to 0, timeout is disabled. [seconds]

Default: 600.0

### `server_connect_timeout`

If connection and login will not finish in this number of seconds, the connection will be closed.

Default: 15.0

### `server_login_retry`

If a login fails due to failure from `connect()` or authentication, the pooler waits this many seconds before retrying to connect.

Default: 15.0

### `client_login_timeout`
If a client connects but does not manage to login in this number of seconds, it is disconnected. This is needed to avoid dead connections stalling SUSPEND and thus online restart.

Default: 60.0

**autodb_idle_timeout**

If database pools created automatically (via `***`) have been unused this many seconds, they are freed. Their statistics are also forgotten.

Default: 3600.0

**dns_max_ttl**

How long to cache DNS lookups, in seconds. If a DNS lookup returns several answers, PgBouncer round-robins between them in the meantime. The actual DNS TTL is ignored.

Default: 15.0

**dns_nxdomain_ttl**

How long error and NXDOMAIN DNS lookups can be cached, in seconds.

Default: 15.0

**dns_zone_check_period**

Period to check if zone serial numbers have changed.

PgBouncer can collect DNS zones from hostnames (everything after first dot) and then periodically check if the zone serial numbers change. If changes are detected, all hostnames in that zone are looked up again. If any host IP changes, its connections are invalidated.

Works only with UDNS backend (**--with-udns** to configure).

Default: 0.0 (disabled)

**Dangerous Timeouts**

Setting the following timeouts can cause unexpected errors.

**query_timeout**

Queries running longer than this (seconds) are canceled. This parameter should be used only with a slightly smaller server-side statement_timeout, to trap queries with network problems. [seconds]

Default: 0.0 (disabled)

**query_wait_timeout**

The maximum time, in seconds, queries are allowed to wait for execution. If the query is not assigned a connection during that time, the client is disconnected. This is used to prevent unresponsive servers from grabbing up connections.

Default: 0.0 (disabled)

**client_idle_timeout**

Client connections idling longer than this many seconds are closed. This should be larger than the client-side connection lifetime settings, and only used for network problems.

Default: 0.0 (disabled)

**idle_transaction_timeout**

If client has been in "idle in transaction" state longer than this (seconds), it is disconnected.

Default: 0.0 (disabled)
Low-level Network Settings

**pkt_buf**
Internal buffer size for packets. Affects the size of TCP packets sent and general memory usage. Actual libpq packets can be larger than this so there is no need to set it large.

Default: 2048

**max_packet_size**
Maximum size for packets that PgBouncer accepts. One packet is either one query or one result set row. A full result set can be larger.

Default: 2147483647

**listen_backlog**
Backlog argument for the `listen(2)` system call. It how many new unanswered connection attempts are kept in queue. When the queue is full, further new connection attempts are dropped.

Default: 128

**sbuf_loopcnt**
How many times to process data on one connection, before proceeding. Without this limit, one connection with a big result set can stall PgBouncer for a long time. One loop processes one `pkt_buf` amount of data. 0 means no limit.

Default: 5

**suspend_timeout**
How many seconds to wait for buffer flush during SUSPEND or reboot (-R). Connection is dropped if flush does not succeed.

Default: 10

**tcp_defer_accept**
For details on this and other TCP options, please see the tcp(7) man page.

Default: 45 on Linux, otherwise 0

**tcp_socket_buffer**
Default: not set

**tcp_keepalive**
Turns on basic keepalive with OS defaults.

On Linux, the system defaults are `tcp_keepidle=7200, tcp_keepintvl=75, tcp_keepcnt=9`.

Default: 1

**tcp_keepcnt**
Default: not set

**tcp_keepidle**
Default: not set

**tcp_keepintvl**
Default: not set

[**users**] Section
This section contains **key=value** pairs, where the key is a user name and the value is a libpq connect-string list of **key=value** pairs.
Pool configuration

pool_mode

Set the pool mode to be used for all connections from this user. If not set, the database or default pool_mode is used.

Example Configuration Files

Minimal Configuration

```
[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
```

Use connection parameters passed by the client:

```
[databases]
* =

[pgbouncer]
listen_port = 65432
listen_addr = 0.0.0.0
auth_type = trust
auth_file = bouncer/users.txt
logfile = pgbouncer.log
pidfile = pgbouncer.pid
ignore_startup_parameters=options
```

Database Defaults

```
[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=baz password=foo
client_encoding=UNICODE datestyle=ISO
```

PgBouncer Authentication File Format

PgBouncer requires its own user database, a text file in following format:

```
"username1" "password" ...
"username2" "md5abcdef012342345" ...
```
There is one line per user. Each line must have at least two fields. Fields are enclosed in double quotes (""). The first field is the user name and the second is either a plain-text or an MD5-encoded password. The remainder of the line is ignored.

This file format is similar to text files used by Greenplum Database for authentication information, and PgBouncer can work directly with the Greenplum Database authentication files.

To avoid plain-text passwords, encode user passwords with MD5. The format for an MD5 encoded password is:

"md5" + md5(password + username)

For example, the following command generates the MD5 string for the user admin with password 1234:

```bash
$ echo -n "1234admin" | md5sum
$ 45f2603610af569b6155c45067268c6b
```

The MD5-hidden password is:

```bash
md545f2603610af569b6155c45067268c6b
```

### PgBouncer Administration Console Commands

The PgBouncer Administration Console is accessed by connecting to the database `pgbouncer`.

```bash
$ psql -p 6543 pgbouncer
```

Only users listed in configuration parameters `admin_users` or `stats_users` can log in to the console. However, when `auth_mode=any`, then any user may log in as a `stats_user`.

The user name `pgbouncer` may also log in without a password through a Unix socket if the client has the same Unix user UID as the running process.

#### Administration Console Command Syntax

```
pgbouncer=# show help;
NOTICE:  Console usage
DETAIL:
   SHOW HELP|CONFIG|DATABASES|POOLS|CLIENTS|SERVERS|VERSION
   SHOW STATS|FDS|SOCKETS|ACTIVE_SOCKETS|LISTS|MEM
   SHOW DNS_HOSTS|DNS_ZONES
   SET key = arg
   RELOAD
   PAUSE [</db>]
   RESUME [</db>]
   DISABLE </db>
   ENABLE </db>
   KILL </db>
   SUSPEND
   SHUTDOWN
```

#### Administration Commands

From the PgBouncer Administrator console you can control connections between PgBouncer and Greenplum Database. You can also set PgBouncer configuration parameters.

The following PgBouncer administration commands control the PgBouncer process.

**PAUSE [database]**
If no database is specified, PgBouncer tries to disconnect from all servers, first waiting for all queries to complete. The command will not return before all queries are finished. This command is to be used to prepare to restart the database.

If a database name is specified, only that database is paused.

If you run a PAUSE database command, and then a PAUSE command to pause all databases, you must execute two RESUME commands, one for all databases, and one for the named database.

SUSPEND

All socket buffers are flushed and PgBouncer stops listening for data on them. The command will not return before all buffers are empty. To be used when rebooting PgBouncer online.

RESUME [ database ]

Resume work from a previous PAUSE or SUSPEND command.

If a database was specified for the PAUSE command, the database must also be specified with the RESUME command.

After pausing all databases with the PAUSE command, resuming a single database with RESUME database is not supported.

DISABLE database

Reject all new client connections on the database.

ENABLE database

Allow new client connections on the database.

KILL database

Immediately drop all client and server connections to the named database.

SHUTDOWN

Stop PgBouncer process. To exit from the psql command line session, enter \q.

RELOAD

The PgBouncer process reloads the current configuration file and updates the changeable settings.

SET key = value

Override specified configuration setting. See the SHOW CONFIG; command.

SHOW Command

The SHOW category command displays different types of PgBouncer information. You can specify one of the following categories:

- ACTIVE_SOCKETS
- CLIENTS
- CONFIG
- DATABASES
- DNS_ZONES
- FDS
- POOLS
- SERVERS
- STATS
- LISTS
- MEM
- USERS
### ACTIVE_SOCKETS

**Table 73: Active Socket Information**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>S, for server, C for client.</td>
</tr>
<tr>
<td>user</td>
<td>Username pgbouncer uses to connect to server.</td>
</tr>
<tr>
<td>database</td>
<td>Database name.</td>
</tr>
<tr>
<td>state</td>
<td>State of the server connection, one of active, used or idle.</td>
</tr>
<tr>
<td>addr</td>
<td>IP address of PostgreSQL server.</td>
</tr>
<tr>
<td>port</td>
<td>Port of PostgreSQL server.</td>
</tr>
<tr>
<td>local_addr</td>
<td>Connection start address on local machine.</td>
</tr>
<tr>
<td>local_port</td>
<td>Connection start port on local machine.</td>
</tr>
<tr>
<td>connect_time</td>
<td>When the connection was made.</td>
</tr>
<tr>
<td>request_time</td>
<td>When last request was issued.</td>
</tr>
<tr>
<td>ptr</td>
<td>Address of internal object for this connection. Used as unique ID.</td>
</tr>
<tr>
<td>link</td>
<td>Address of client connection the server is paired with.</td>
</tr>
<tr>
<td>recv_pos</td>
<td>Receive position in the I/O buffer.</td>
</tr>
<tr>
<td>pkt_pos</td>
<td>Parse position in the I/O buffer.</td>
</tr>
<tr>
<td>pkt_remain</td>
<td>Number of packets remaining on the socket.</td>
</tr>
<tr>
<td>send_pos</td>
<td>Send position in the packet.</td>
</tr>
<tr>
<td>send_remain</td>
<td>Total packet length remaining to send.</td>
</tr>
<tr>
<td>pkt_avail</td>
<td>Amount of I/O buffer left to parse.</td>
</tr>
<tr>
<td>send_avail</td>
<td>Amount of I/O buffer left to send.</td>
</tr>
</tbody>
</table>

### CLIENTS

**Table 74: Clients**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>C, for client.</td>
</tr>
<tr>
<td>user</td>
<td>Client connected user.</td>
</tr>
<tr>
<td>database</td>
<td>Database name.</td>
</tr>
<tr>
<td>state</td>
<td>State of the client connection, one of active, used, waiting or idle.</td>
</tr>
<tr>
<td>addr</td>
<td>IP address of client, or unix for a socket connection.</td>
</tr>
<tr>
<td>port</td>
<td>Port client is connected to.</td>
</tr>
<tr>
<td>local_addr</td>
<td>Connection end address on local machine.</td>
</tr>
<tr>
<td>local_port</td>
<td>Connection end port on local machine.</td>
</tr>
<tr>
<td>Column</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------------------------------------</td>
</tr>
<tr>
<td>connect_time</td>
<td>Timestamp of connect time.</td>
</tr>
<tr>
<td>request_time</td>
<td>Timestamp of latest client request.</td>
</tr>
<tr>
<td>ptr</td>
<td>Address of internal object for this connection. Used as unique ID.</td>
</tr>
<tr>
<td>link</td>
<td>Address of server connection the client is paired with.</td>
</tr>
<tr>
<td>remote_pid</td>
<td>Process ID, if client connects with Unix socket and the OS supports getting it.</td>
</tr>
</tbody>
</table>

**CONFIG**

List of current PgBouncer parameter settings

**Table 75: Config**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>Configuration variable name</td>
</tr>
<tr>
<td>value</td>
<td>Configuration value</td>
</tr>
<tr>
<td>changeable</td>
<td>Either yes or no. Shows whether the variable can be changed while running. If no, the variable can be changed only at boot time.</td>
</tr>
</tbody>
</table>

**DATABASES**

**Table 76: Databases**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of configured database entry.</td>
</tr>
<tr>
<td>host</td>
<td>Host pgbouncer connects to.</td>
</tr>
<tr>
<td>port</td>
<td>Port pgbouncer connects to.</td>
</tr>
<tr>
<td>database</td>
<td>Actual database name pgbouncer connects to.</td>
</tr>
<tr>
<td>force_user</td>
<td>When user is part of the connection string, the connection between pgbouncer and the database server is forced to the given user, whatever the client user.</td>
</tr>
<tr>
<td>pool_size</td>
<td>Maximum number of server connections.</td>
</tr>
<tr>
<td>reserve_pool</td>
<td>The number of additional connections that can be created if the pool reaches pool_size.</td>
</tr>
<tr>
<td>pool_mode</td>
<td>The database’s override pool_mode or NULL if the default will be used instead.</td>
</tr>
<tr>
<td>max_connections</td>
<td>Maximum number of connections for all pools for this database.</td>
</tr>
<tr>
<td>current_connections</td>
<td>The total count of connections for all pools for this database.</td>
</tr>
</tbody>
</table>
**DNS_ZONES**

Table 77: DNS Zones in Cache

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zonename</td>
<td>Zone name</td>
</tr>
<tr>
<td>serial</td>
<td>Current DNS serial number</td>
</tr>
<tr>
<td>count</td>
<td>Hostnames belonging to this zone</td>
</tr>
</tbody>
</table>

**FDS**

`SHOW FDS` is an internal command used for an online restart, for example when upgrading to a new PgBouncer version. It shows a list of file descriptors in use with the internal state attached to them. This command blocks the internal event loop, so it should not be used while PgBouncer is in use.

When the connected user has username "pgbouncer", connects through a Unix socket, and has the same UID as the running process, the actual file descriptors are passed over the connection. Note: This does not work on Windows machines.

Table 78: FDS

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>File descriptor numeric value.</td>
</tr>
<tr>
<td>task</td>
<td>One of <code>pooler</code>, <code>client</code>, or <code>server</code>.</td>
</tr>
<tr>
<td>user</td>
<td>User of the connection using the file descriptor.</td>
</tr>
<tr>
<td>database</td>
<td>Database of the connection using the file descriptor.</td>
</tr>
<tr>
<td>addr</td>
<td>IP address of the connection using the file descriptor, &quot;unix&quot; if a Unix socket is used.</td>
</tr>
<tr>
<td>port</td>
<td>Port used by the connection using the file descriptor.</td>
</tr>
<tr>
<td>cancel</td>
<td>Cancel key for this connection.</td>
</tr>
<tr>
<td>link</td>
<td>File descriptor for corresponding server/client. NULL if idle.</td>
</tr>
<tr>
<td>client_encoding</td>
<td>Character set used for the database.</td>
</tr>
<tr>
<td>std_strings</td>
<td>This controls whether ordinary string literals ('...') treat backslashes literally, as specified in the SQL standard.</td>
</tr>
<tr>
<td>datestyle</td>
<td>Display format for date and time values.</td>
</tr>
<tr>
<td>timezone</td>
<td>The timezone for interpreting and displaying time stamps.</td>
</tr>
</tbody>
</table>

**LISTS**

Shows the following PgBouncer statistics in two columns: the item label and value.

Table 79: Count of PgBouncer Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>databases</td>
<td>Count of databases.</td>
</tr>
<tr>
<td>users</td>
<td>Count of users.</td>
</tr>
<tr>
<td>Item</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>pools</td>
<td>Count of pools.</td>
</tr>
<tr>
<td>free_clients</td>
<td>Count of free clients.</td>
</tr>
<tr>
<td>used_clients</td>
<td>Count of used clients.</td>
</tr>
<tr>
<td>login_clients</td>
<td>Count of clients in login state.</td>
</tr>
<tr>
<td>free_servers</td>
<td>Count of free servers.</td>
</tr>
<tr>
<td>used_servers</td>
<td>Count of used servers.</td>
</tr>
<tr>
<td>dns_names</td>
<td>Count of DNS names.</td>
</tr>
<tr>
<td>dns_zones</td>
<td>Count of DNS zones.</td>
</tr>
<tr>
<td>dns_queries</td>
<td>Count of DNS queries.</td>
</tr>
<tr>
<td>dns_pending</td>
<td>Count of in-flight DNS queries.</td>
</tr>
</tbody>
</table>

**MEM**

Shows cache memory information for these PgBouncer caches:

- user_cache
- db_cache
- pool_cache
- server_cache
- client_cache
- iobuf_cache

**Table 80: In Memory Cache**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of cache.</td>
</tr>
<tr>
<td>size</td>
<td>The size of a single slot in the cache.</td>
</tr>
<tr>
<td>used</td>
<td>Number of used slots in the cache.</td>
</tr>
<tr>
<td>free</td>
<td>The number of available slots in the cache.</td>
</tr>
<tr>
<td>memtotal</td>
<td>Total bytes used by the cache.</td>
</tr>
</tbody>
</table>

**POOLS**

A new pool entry is made for each pair of (database, user).

**Table 81: Pools**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>Database name.</td>
</tr>
<tr>
<td>user</td>
<td>User name.</td>
</tr>
<tr>
<td>cl_active</td>
<td>Client connections that are linked to server connection and can process queries.</td>
</tr>
<tr>
<td>cl_waiting</td>
<td>Client connections have sent queries but have not yet got a server connection.</td>
</tr>
<tr>
<td>Column</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>sv_active</td>
<td>Server connections that linked to client.</td>
</tr>
<tr>
<td>sv_idle</td>
<td>Server connections that are unused and immediately usable for client queries.</td>
</tr>
<tr>
<td>sv_used</td>
<td>Server connections that have been idle more than server_check_delay. The server_check_query query must be run on them before they can be used.</td>
</tr>
<tr>
<td>sv_tested</td>
<td>Server connections that are currently running either server_reset_query or server_check_query.</td>
</tr>
<tr>
<td>sv_login</td>
<td>Server connections currently in process of logging in.</td>
</tr>
<tr>
<td>maxwait</td>
<td>How long the first (oldest) client in the queue has waited, in seconds. If this begins to increase, the current pool of servers does not handle requests fast enough. The cause may be either an overloaded server or the pool_size setting is too small.</td>
</tr>
<tr>
<td>pool_mode</td>
<td>The pooling mode in use.</td>
</tr>
</tbody>
</table>

**SERVERS**

Table 82: Servers

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>S, for server.</td>
</tr>
<tr>
<td>user</td>
<td>User ID that pgbouncer uses to connect to server.</td>
</tr>
<tr>
<td>database</td>
<td>Database name.</td>
</tr>
<tr>
<td>state</td>
<td>State of the pgbouncer server connection, one of active, used, or idle.</td>
</tr>
<tr>
<td>addr</td>
<td>IP address of the Greenplum or PostgreSQL server.</td>
</tr>
<tr>
<td>port</td>
<td>Port of the Greenplum or PostgreSQL server.</td>
</tr>
<tr>
<td>local_addr</td>
<td>Connection start address on local machine.</td>
</tr>
<tr>
<td>local_port</td>
<td>Connection start port on local machine.</td>
</tr>
<tr>
<td>connect_time</td>
<td>When the connection was made.</td>
</tr>
<tr>
<td>request_time</td>
<td>When the last request was issued.</td>
</tr>
<tr>
<td>ptr</td>
<td>Address of the internal object for this connection. Used as unique ID.</td>
</tr>
<tr>
<td>link</td>
<td>Address of the client connection the server is paired with.</td>
</tr>
<tr>
<td>remote_pid</td>
<td>Pid of backend server process. If the connection is made over Unix socket and the OS supports getting process ID info, it is the OS pid. Otherwise it is extracted from the cancel packet the server sent, which should be PID in case server is PostgreSQL, but it is a random number in case server is another PgBouncer.</td>
</tr>
</tbody>
</table>

**STATS**

Shows statistics.
Table 83: Stats

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>Statistics are presented per database.</td>
</tr>
<tr>
<td>total_requests</td>
<td>Total number of SQL requests pooled by pgbouncer.</td>
</tr>
<tr>
<td>total_received</td>
<td>Total volume in bytes of network traffic received by pgbouncer.</td>
</tr>
<tr>
<td>total_sent</td>
<td>Total volume in bytes of network traffic sent by pgbouncer.</td>
</tr>
<tr>
<td>total_query_time</td>
<td>Total number of microseconds spent by pgbouncer when actively connected to the database server.</td>
</tr>
<tr>
<td>avg_req</td>
<td>Average requests per second in last stat period.</td>
</tr>
<tr>
<td>avg_recv</td>
<td>Average received (from clients) bytes per second.</td>
</tr>
<tr>
<td>avg_sent</td>
<td>Average sent (to clients) bytes per second.</td>
</tr>
<tr>
<td>avg_query</td>
<td>Average query duration in microseconds.</td>
</tr>
</tbody>
</table>

**USERS**

Table 84: Users

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The user name</td>
</tr>
<tr>
<td>pool_mode</td>
<td>The user’s override pool_mode, or NULL if the default will be used instead.</td>
</tr>
</tbody>
</table>

**VERSION**

Display PgBouncer version information.

*Note:* This reference documentation is based on the PgBouncer 1.6.1 documentation.
Client Utility Reference

Describes the command-line client utilities provided with Greenplum Database.

Greenplum Database uses the standard PostgreSQL client programs and provides additional client utilities for administering a distributed Greenplum Database DBMS. Greenplum Database client utilities reside in $GPHOME/bin.

The following are the Greenplum Database client utilities.

- clusterdb
- createdb
- createlang
- createuser
- dropdb
- droplang
- dropuser
- pg_config
- pg_dump
- pg_dumpall
- pg_restore
- psql
- reindexdb
- vacuumdb

Client Utility Summary

clusterdb
Reclusters tables that were previously clustered with CLUSTER.

```
clusterdb [connection-option ...] [-v] [-t table] [-d] dbname
clusterdb [connection-option ...] [-a] [-v]
clusterdb --help
clusterdb --version
```

See `clusterdb` for more information.

createdb
Creates a new database.

```
createdb [connection_option ...] [-D tablespace] [-E encoding]
[-O owner] [-T template] [-e] [dbname ['description']]
createdb --help
createdb --version
```

See `createdb` for more information.
**createlang**
Defines a new procedural language for a database.

```
createlang [connection_option ...] [-e] langname [[-d] dbname]
createlang [connection-option ...] -l dbname
createlang --help
createlang --version
```

See `createlang` for more information.

**createuser**
Creates a new database role.

```
createuser [connection_option ...] [role_attribute ...] [-e] role_name
createuser --help
createuser --version
```

See `createuser` for more information.

**dropdb**
Removes a database.

```
dropdb [connection_option ...] [-e] [-i] dbname
dropdb --help
dropdb --version
```

See `dropdb` for more information.

**droplang**
Removes a procedural language.

```
droplang [connection-option ...] [-e] langname [[-d] dbname]
droplang [connection-option ...] [-e] -l dbname
droplang --help
droplang --version
```

See `droplang` for more information.

**dropuser**
Removes a database role.

```
dropuser [connection_option ...] [-e] [-i] role_name
dropuser --help
```
dropuser --version
See dropuser for more information.

pg_config
Retrieves information about the installed version of Greenplum Database.

pg_config [option ...]
See pg_config for more information.

pg_dump
Extracts a database into a single script file or other archive file.

pg_dump [connection_option ...] [dump_option ...] dbname
See pg_dump for more information.

pg_dumpall
Extracts all databases in a Greenplum Database system to a single script file or other archive file.

pg_dumpall [connection_option ...] [dump_option ...]
See pg_dumpall for more information.

pg_restore
Restores a database from an archive file created by pg_dump.

pg_restore [connection_option ...] [restore_option ...] filename
See pg_restore for more information.

psql
Interactive command-line interface for Greenplum Database

psql [option ...] [dbname [username]]
See psql for more information.

reindexdb
Rebuilds indexes in a database.

reindexdb [connection-option ...] [--table | -t table ]
        [--index | -i index ] [dbname]
reindexdb [connection-option ...] [--all | -a]
reindexdb [connection-option ...] [--system | -s] [dbname]
reindexdb --help
reindexdb --version
See `reindexdb` for more information.

**vacuumdb**

Garbage-collects and analyzes a database.

```
vacuumdb [connection-option...] [--full | -f] [--verbose | -v]
                   [--analyze | -z] [--table | -t table [ ( column [...]) ] ] dbname
```

```
vacuumdb [connection-options...] [--all | -a] [--full | -f] [-F]
                   [--analyze | -z]
```

```
vacuumdb --help
vacuumdb --version
```

See `vacuumdb` for more information.

**clusterdb**

Reclusters tables that were previously clustered with `CLUSTER`.

**Synopsis**

```
clusterdb [connection-option ...] [-v] [-t table] [ -d dbname]
```

```
clusterdb [connection-option ...] [-a] [-v]
```

```
clusterdb --help
clusterdb --version
```

**Description**

To cluster a table means to physically reorder a table on disk according to an index so that index scan operations can access data on disk in a somewhat sequential order, thereby improving index seek performance for queries that use that index.

The `clusterdb` utility will find any tables in a database that have previously been clustered with the `CLUSTER` SQL command, and clusters them again on the same index that was last used. Tables that have never been clustered are not affected.

`clusterdb` is a wrapper around the SQL command `CLUSTER`. Although clustering a table in this way is supported in Greenplum Database, it is not recommended because the `CLUSTER` operation itself is extremely slow.

If you do need to order a table in this way to improve your query performance, Greenplum recommends using a `CREATE TABLE AS` statement to reorder the table on disk rather than using `CLUSTER`. If you do ‘cluster’ a table in this way, then `clusterdb` would not be relevant.

**Options**

- `-a | --all`
  - Cluster all databases.

- `[-d] dbname | [--dbname] dbname`
  - Specifies the name of the database to be clustered. If this is not specified, the database name is read from the environment variable `PGDATABASE`. If that is not set, the user name specified for the connection is used.
-e | --echo
    Echo the commands that clusterdb generates and sends to the server.

-q | --quiet
    Do not display a response.

-t table | --table table
    Cluster the named table only.

-v | --verbose
    Print detailed information during processing.

Connection Options

-h host | --host host
    The host name of the machine on which the Greenplum master database server is running. If not specified, reads from the environment variable PGHOST or defaults to localhost.

-p port | --port port
    The TCP port on which the Greenplum master database server is listening for connections. If not specified, reads from the environment variable PGPORT or defaults to 5432.

-U username | --username username
    The database role name to connect as. If not specified, reads from the environment variable PGUSER or defaults to the current system role name.

-w | --no-password
    Never issue a password prompt. If the server requires password authentication and a password is not available by other means such as a .pgpass file, the connection attempt will fail. This option can be useful in batch jobs and scripts where no user is present to enter a password.

-W | --password
    Force a password prompt.

Examples

To cluster the database test:

```
clusterdb test
```

To cluster a single table foo in a database named xyzzy:

```
clusterdb --table foo xyzzyb
```

See Also

CLUSTER in the Greenplum Database Reference Guide

createdb

Creates a new database.

Synopsis

```
createdb [connection_option ...] [-D tablespace] [-E encoding] [-O owner] [-T template] [-e] [dbname ['description']] 
createdb --help
```
createdb --version

Description

createdb creates a new database in a Greenplum Database system.

Normally, the database user who executes this command becomes the owner of the new database. However a different owner can be specified via the -O option, if the executing user has appropriate privileges.

createdb is a wrapper around the SQL command CREATE DATABASE.

Options
database

The name of the database to be created. The name must be unique among all other databases in the Greenplum system. If not specified, reads from the environment variable PGDATABASE, then PGUSER or defaults to the current system user.

description

A comment to be associated with the newly created database. Descriptions containing white space must be enclosed in quotes.

-D tablespace | --tablespace tablespace

The default tablespace for the database.

e echo

Echo the commands that createdb generates and sends to the server.

-E encoding | --encoding encoding

Character set encoding to use in the new database. Specify a string constant (such as 'UTF8'), an integer encoding number, or DEFAULT to use the default encoding. See the Greenplum Database Reference Guide for information about supported character sets.

-O owner | --owner owner

The name of the database user who will own the new database. Defaults to the user executing this command.

-T template | --template template

The name of the template from which to create the new database. Defaults to template1.

Connection Options

-h host | --host host

The host name of the machine on which the Greenplum master database server is running. If not specified, reads from the environment variable PGHOST or defaults to localhost.

-p port | --port port

The TCP port on which the Greenplum master database server is listening for connections. If not specified, reads from the environment variable PGPORT or defaults to 5432.

-U username | --username username

The database role name to connect as. If not specified, reads from the environment variable PGUSER or defaults to the current system role name.

-w | --no-password

Never issue a password prompt. If the server requires password authentication and a password is not available by other means such as a .pgpass file, the connection attempt will fail. This option can be useful in batch jobs and scripts where no user is present to enter a password.
Examples
To create the database test using the default options:

```bash
createdb test
```

To create the database demo using the Greenplum master on host gpmaster, port 54321, using the LATIN1 encoding scheme:

```bash
createdb -p 54321 -h gpmaster -E LATIN1 demo
```

See Also
CREATE DATABASE in the Greenplum Database Reference Guide

createuser
Creates a new database role.

Synopsis

```bash
createuser [connection_option ...] [role_attribute ...] [-e] role_name
createuser --help
createuser --version
```

Description

createuser creates a new Greenplum Database role. You must be a superuser or have the CREATEROLE privilege to create new roles. You must connect to the database as a superuser to create new superusers.

Superusers can bypass all access permission checks within the database, so superuser privileges should not be granted lightly.

createuser is a wrapper around the SQL command CREATE ROLE.

Options

role_name

The name of the role to be created. This name must be different from all existing roles in this Greenplum Database installation.

-c number | --connection-limit number

Set a maximum number of connections for the new role. The default is to set no limit.

-D | --no-createdb

The new role will not be allowed to create databases. This is the default.

-d | --createdb

The new role will be allowed to create databases.

-e | --echo

Echo the commands that createuser generates and sends to the server.

-E | --encrypted
Encrypts the role’s password stored in the database. If not specified, the default password behavior is used.

- i | --inherit
    The new role will automatically inherit privileges of roles it is a member of. This is the default.

- I | --no-inherit
    The new role will not automatically inherit privileges of roles it is a member of.

- l | --login
    The new role will be allowed to log in to Greenplum Database. This is the default.

- L | --no-login
    The new role will not be allowed to log in (a group-level role).

- N | --unencrypted
    Does not encrypt the role’s password stored in the database. If not specified, the default password behavior is used.

- P | --pwprompt
    If given, createuser will issue a prompt for the password of the new role. This is not necessary if you do not plan on using password authentication.

- r | --createrole
    The new role will be allowed to create new roles (CREATEROLE privilege).

- R | --no-createrole
    The new role will not be allowed to create new roles. This is the default.

- s | --superuser
    The new role will be a superuser.

- S | --no-superuser
    The new role will not be a superuser. This is the default.

Connection Options
- h host | --host host
    The host name of the machine on which the Greenplum master database server is running. If not specified, reads from the environment variable PGHOST or defaults to localhost.

- p port | --port port
    The TCP port on which the Greenplum master database server is listening for connections. If not specified, reads from the environment variable PGPORT or defaults to 5432.

- U username | --username username
    The database role name to connect as. If not specified, reads from the environment variable PGUSER or defaults to the current system role name.

- w | --no-password
    Never issue a password prompt. If the server requires password authentication and a password is not available by other means such as a .pgpass file, the connection attempt will fail. This option can be useful in batch jobs and scripts where no user is present to enter a password.

- W | --password
    Force a password prompt.
Examples

Create a role named joe using the default options:

```
createuser 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

To create the same role joe using connection options and avoiding the prompts and taking a look at the underlying command:

```
createuser -h masterhost -p 54321 -S -D -R -e joe
```

CREATE ROLE Joe NOSUPERUSER NOCREATEDB NOCREATEROLE INHERIT 
LOGIN; 
CREATE ROLE

To create the role joe as a superuser, and assign password admin123 immediately:

```
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

In the above example, the new password is not actually echoed when typed, but we show what was typed for clarity. However the password will appear in the echoed command, as illustrated if the -e option is used.

See Also

CREATE ROLE in the Greenplum Database Reference Guide

dropdb

Removes a database.

Synopsis

```
dropdb [connection_option ...] [-e] [-i] dbname

dropdb --help

dropdb --version
```

Description

dropdb destroys an existing database. The user who executes this command must be a superuser or the owner of the database being dropped.

dropdb is a wrapper around the SQL command DROP DATABASE. See the Greenplum Database Reference Guide for information about DROP DATABASE.

Options

```
dbname
```

The name of the database to be removed.
-e | --echo
    Echo the commands that dropdb generates and sends to the server.

-i | --interactive
    Issues a verification prompt before doing anything destructive.

**Connection Options**

- **-h host | --host host**
  The host name of the machine on which the Greenplum master database server is running. If not specified, reads from the environment variable PGHOST or defaults to localhost.

- **-p port | --port port**
  The TCP port on which the Greenplum master database server is listening for connections. If not specified, reads from the environment variable PGPORT or defaults to 5432.

- **-U username | --username username**
  The database role name to connect as. If not specified, reads from the environment variable PGUSER or defaults to the current system role name.

- **-w | --no-password**
  Never issue a password prompt. If the server requires password authentication and a password is not available by other means such as a .pgpass file, the connection attempt will fail. This option can be useful in batch jobs and scripts where no user is present to enter a password.

- **-W | --password**
  Force a password prompt.

**Examples**

To destroy the database named demo using default connection parameters:

```
dropdb demo
```

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
```

**See Also**

DROP DATABASE in the Greenplum Database Reference Guide

droplang

Removes a procedural language.

**Synopsis**

```
droplang [connection-option ...] [-e] langname [[-d] dbname]
droplang [connection-option ...] [-e] -l dbname
droplang --help
```
droplang --version

Description

droplang removes an existing programming language from a database. droplang can drop any procedural language, even those not supplied by the Greenplum Database distribution.

Although programming languages can be removed directly using several SQL commands, it is recommended to use droplang because it performs a number of checks and is much easier to use.

droplang is a wrapper for the SQL command DROP LANGUAGE.

Options

langname

Specifies the name of the programming language to be removed.

[-d] dbname | [--dbname] dbname

Specifies from which database the language should be removed. The default is to use the PGDATABASE environment variable setting, or the same name as the current system user.

-e | --echo

Echo the commands that droplang generates and sends to the server.

-l | --list

Show a list of already installed languages in the target database.

Connection Options

-h host | --host host

The host name of the machine on which the Greenplum master database server is running. If not specified, reads from the environment variable PGHOST or defaults to localhost.

-p port | --port port

The TCP port on which the Greenplum master database server is listening for connections. If not specified, reads from the environment variable PGPORT or defaults to 5432.

-U username | --username username

The database role name to connect as. If not specified, reads from the environment variable PGUSER or defaults to the current system role name.

-w | --no-password

Never issue a password prompt. If the server requires password authentication and a password is not available by other means such as a .pgpass file, the connection attempt will fail. This option can be useful in batch jobs and scripts where no user is present to enter a password.

-W | --password

Force a password prompt.

Examples

To remove the language pltcl from the mydatabase database:

droplang pltcl mydatabase

See Also

DROP LANGUAGE in the Greenplum Database Reference Guide
dropuser

Removes a database role.

Synopsis

```
dropuser [connection_option ...] [-e] [-i] role_name
```

```
dropuser --help
```

```
dropuser --version
```

Description

dropuser removes an existing role from Greenplum Database. Only superusers and users with the CREATEROLE privilege can remove roles. To remove a superuser role, you must yourself be a superuser.
dropuser is a wrapper around the SQL command DROP ROLE.

Options

```
role_name
```

The name of the role to be removed. You will be prompted for a name if not specified on the command line.

```
-e | --echo
```

Echo the commands that dropuser generates and sends to the server.

```
-i | --interactive
```

Prompt for confirmation before actually removing the role.

Connection Options

```
h | --host host
```

The host name of the machine on which the Greenplum master database server is running. If not specified, reads from the environment variable PGHOST or defaults to localhost.

```
p | --port port
```

The TCP port on which the Greenplum master database server is listening for connections. If not specified, reads from the environment variable PGPORT or defaults to 5432.

```
-U username | --username username
```

The database role name to connect as. If not specified, reads from the environment variable PGUSER or defaults to the current system role name.

```
w | --no-password
```

Never issue a password prompt. If the server requires password authentication and a password is not available by other means such as a .pgpass file, the connection attempt will fail. This option can be useful in batch jobs and scripts where no user is present to enter a password.

```
-W | --password
```

Force a password prompt.

Examples

To remove the role joe using default connection options:

```
dropuser joe
```
To remove the role joe using connection options, with verification, and a peek at the underlying command:

```
dropuser -p 54321 -h masterhost -i -e joe
Role "joe" will be permanently removed.
Are you sure? (y/n) y
```

```
DROP ROLE "joe"
```

See Also

DROP ROLE in the Greenplum Database Reference Guide

createlang

Defines a new procedural language for a database.

**Synopsis**

```
createlang [connection_option ...] [-e] langname [[-d] dbname]
createlang [connection-option ...] -l dbname
createlang --help
createlang --version
```

**Description**

The createlang utility adds a new programming language to a database. createlang is a wrapper around the SQL command CREATE LANGUAGE.

The procedural language packages included in the standard Greenplum Database distribution are:

- PL/pgSQL
- PL/Perl
- PL/Python

The PL/pgSQL language is registered in all databases by default.

Greenplum Database also has language handlers for PL/Java and PL/R, but those languages are not pre-installed with Greenplum Database. See the Procedural Languages section in the PostgreSQL documentation for more information.

**Options**

- `langname`
  Specifies the name of the procedural programming language to be defined.

- `[-d] dbname | [--dbname] dbname`
  Specifies to which database the language should be added. The default is to use the PGDATABASE environment variable setting, or the same name as the current system user.

- `-e | --echo`
  Echo the commands that createlang generates and sends to the server.

- `-l dbname | --list dbname`
  Show a list of already installed languages in the target database.

**Connection Options**
-h host | --host host
   The host name of the machine on which the Greenplum master database server is
   running. If not specified, reads from the environment variable PGHOST or defaults to
   localhost.

-p port | --port port
   The TCP port on which the Greenplum master database server is listening for connections.
   If not specified, reads from the environment variable PGPORT or defaults to 5432.

-U username | --username username
   The database role name to connect as. If not specified, reads from the environment
   variable PGUSER or defaults to the current system role name.

-w | --no-password
   Never issue a password prompt. If the server requires password authentication and a
   password is not available by other means such as a .pgpass file, the connection attempt
   will fail. This option can be useful in batch jobs and scripts where no user is present to
   enter a password.

-W | --password
   Force a password prompt.

Examples
To install the language plperl into the database mytestdb:

```
createllang plperl mytestdb
```

See Also
CREATE LANGUAGE and DROP LANGUAGE in the Greenplum Database Reference Guide

pg_config
Retrieves information about the installed version of Greenplum Database.

Synopsis

```
pg_config [option ...]
```

Description
The pg_config utility prints configuration parameters of the currently installed version of Greenplum
Database. It is intended, for example, to be used by software packages that want to interface to Greenplum
Database to facilitate finding the required header files and libraries. Note that information printed out by
pg_config is for the Greenplum Database master only.

If more than one option is given, the information is printed in that order, one item per line. If no options are
given, all available information is printed, with labels.

Options
--bindir
   Print the location of user executables. Use this, for example, to find the psql program.
   This is normally also the location where the pg_config program resides.

--docdir
   Print the location of documentation files.
--includedir
    Print the location of C header files of the client interfaces.

--pkgincludedir
    Print the location of other C header files.

--includedir-server
    Print the location of C header files for server programming.

--libdir
    Print the location of object code libraries.

--pkglibdir
    Print the location of dynamically loadable modules, or where the server would search for
    them. (Other architecture-dependent data files may also be installed in this directory.)

--localedir
    Print the location of locale support files.

--mandir
    Print the location of manual pages.

--sharedir
    Print the location of architecture-independent support files.

--sysconfdir
    Print the location of system-wide configuration files.

--pgxs
    Print the location of extension makefiles.

--configure
    Print the options that were given to the configure script when Greenplum Database was
    configured for building.

--cc
    Print the value of the CC variable that was used for building Greenplum Database. This
    shows the C compiler used.

--cppflags
    Print the value of the CPPFLAGS variable that was used for building Greenplum Database.
    This shows C compiler switches needed at preprocessing time.

--cflags
    Print the value of the CFLAGS variable that was used for building Greenplum Database.
    This shows C compiler switches.

--cflags_sl
    Print the value of the CFLAGS_SL variable that was used for building Greenplum
    Database. This shows extra C compiler switches used for building shared libraries.

--ldflags
    Print the value of the LDFLAGS variable that was used for building Greenplum Database.
    This shows linker switches.

--ldflags_sl
    Print the value of the LDFLAGS_SL variable that was used for building Greenplum
    Database. This shows linker switches used for building shared libraries.

--libs
Print the value of the LIBS variable that was used for building Greenplum Database. This normally contains -l switches for external libraries linked into Greenplum Database.

--version

Print the version of Greenplum Database.

Examples

To reproduce the build configuration of the current Greenplum Database installation, run the following command:

eval ./configure 'pg_config --configure'

The output of pg_config --configure contains shell quotation marks so arguments with spaces are represented correctly. Therefore, using eval is required for proper results.

pg_dump

Extracts a database into a single script file or other archive file.

Synopsis

```
pg_dump [connection_option ...] [dump_option ...] dbname
```

Description

pg_dump is a standard PostgreSQL utility for backing up a database, and is also supported in Greenplum Database. It creates a single (non-parallel) dump file. For routine backups of Greenplum Database, it is better to use the Greenplum Database backup utility, gpcrondump, for the best performance.

Use pg_dump if you are migrating your data to another database vendor's system, or to another Greenplum Database system with a different segment configuration (for example, if the system you are migrating to has greater or fewer segment instances). To restore, you must use the corresponding pg_restore utility (if the dump file is in archive format), or you can use a client program such as psql (if the dump file is in plain text format).

Since pg_dump is compatible with regular PostgreSQL, it can be used to migrate data into Greenplum Database. The pg_dump utility in Greenplum Database is very similar to the PostgreSQL pg_dump utility, with the following exceptions and limitations:

- If using pg_dump to backup a Greenplum Database database, keep in mind that the dump operation can take a long time (several hours) for very large databases. Also, you must make sure you have sufficient disk space to create the dump file.
- If you are migrating data from one Greenplum Database system to another, use the --gp-syntax command-line option to include the DISTRIBUTED BY clause in CREATE TABLE statements. This ensures that Greenplum Database table data is distributed with the correct distribution key columns upon restore.

pg_dump makes consistent backups even if the database is being used concurrently. pg_dump does not block other users accessing the database (readers or writers).

When used with one of the archive file formats and combined with pg_restore, pg_dump provides a flexible archival and transfer mechanism. pg_dump can be used to backup an entire database, then pg_restore can be used to examine the archive and/or select which parts of the database are to be restored. The most flexible output file format is the custom format (-Fc). It allows for selection and reordering of all archived items, and is compressed by default. The tar format (-Ft) is not compressed and it is not possible to reorder data when loading, but it is otherwise quite flexible. It can be manipulated with standard UNIX tools such as tar.
Note: The --ignore-version option is deprecated and will be removed in a future release.

Options

dbname

Specifies the name of the database to be dumped. If this is not specified, the environment variable PGDATABASE is used. If that is not set, the user name specified for the connection is used.

Dump Options

-a | --data-only

Dump only the data, not the schema (data definitions). This option is only meaningful for the plain-text format. For the archive formats, you may specify the option when you call pg_restore.

-b | --blobs

Include large objects in the dump. This is the default behavior except when --schema, --table, or --schema-only is specified, so the -b switch is only useful to add large objects to selective dumps.

-c | --clean

Adds commands to the text output file to clean (drop) database objects prior to (the commands for) creating them. Note that objects are not dropped before the dump operation begins, but DROP commands are added to the DDL dump output files so that when you use those files to do a restore, the DROP commands are run prior to the CREATE commands. This option is only meaningful for the plain-text format. For the archive formats, you may specify the option when you call pg_restore.

-C | --create

Begin the output with a command to create the database itself and reconnect to the created database. (With a script of this form, it doesn't matter which database you connect to before running the script.) This option is only meaningful for the plain-text format. For the archive formats, you may specify the option when you call pg_restore.

-d | --inserts

Dump data as INSERT commands (rather than COPY). This will make restoration very slow; it is mainly useful for making dumps that can be loaded into non-PostgreSQL-based databases. Also, since this option generates a separate command for each row, an error in reloading a row causes only that row to be lost rather than the entire table contents. Note that the restore may fail altogether if you have rearranged column order. The -D option is safe against column order changes, though even slower.

-D | --column-inserts | --attribute-inserts

Dump data as INSERT commands with explicit column names (INSERT INTO table (column, ...) VALUES ...). This will make restoration very slow; it is mainly useful for making dumps that can be loaded into non-PostgreSQL-based databases. Also, since this option generates a separate command for each row, an error in reloading a row causes only that row to be lost rather than the entire table contents.

-E encoding | --encoding=encoding

Create the dump in the specified character set encoding. By default, the dump is created in the database encoding. (Another way to get the same result is to set the PGCLIENTENCODING environment variable to the desired dump encoding.)

-f file | --file=file

Send output to the specified file. If this is omitted, the standard output is used.

-F p|c|t | --format=plain|custom|tar

Selects the format of the output. format can be one of the following:
p | plain — Output a plain-text SQL script file (the default).

c | custom — Output a custom archive suitable for input into `pg_restore`. This is the most flexible format in that it allows reordering of loading data as well as object definitions. This format is also compressed by default.

t | tar — Output a tar archive suitable for input into `pg_restore`. Using this archive format allows reordering and/or exclusion of database objects at the time the database is restored. It is also possible to limit which data is reloaded at restore time.

-i | --ignore-version

   Note: This option is deprecated and will be removed in a future release.

Ignore version mismatch between `pg_dump` and the database server. `pg_dump` can dump from servers running previous releases of Greenplum Database (or PostgreSQL), but very old versions may not be supported anymore. Use this option if you need to override the version check.

-n schema | --schema=schema

   Dump only schemas matching the schema pattern; this selects both the schema itself, and all its contained objects. When this option is not specified, all non-system schemas in the target database will be dumped. Multiple schemas can be selected by writing multiple `-n` switches. Also, the schema parameter is interpreted as a pattern according to the same rules used by `psql`'s `\d` commands, so multiple schemas can also be selected by writing wildcard characters in the pattern. When using wildcards, be careful to quote the pattern if needed to prevent the shell from expanding the wildcards.

   Note: When `-n` is specified, `pg_dump` makes no attempt to dump any other database objects that the selected schema(s) may depend upon. Therefore, there is no guarantee that the results of a specific-schema dump can be successfully restored by themselves into a clean database.

   Note: Non-schema objects such as blobs are not dumped when `-n` is specified. You can add blobs back to the dump with the `--blobs` switch.

-N schema | --exclude-schema=schema

   Do not dump any schemas matching the schema pattern. The pattern is interpreted according to the same rules as for `-n`. `-N` can be given more than once to exclude schemas matching any of several patterns. When both `-n` and `-N` are given, the behavior is to dump just the schemas that match at least one `-n` switch but no `-N` switches. If `-N` appears without `-n`, then schemas matching `-N` are excluded from what is otherwise a normal dump.

-o | --oids

   Dump object identifiers (OIDs) as part of the data for every table. Use of this option is not recommended for files that are intended to be restored into Greenplum Database.

-O | --no-owner

   Do not output commands to set ownership of objects to match the original database. By default, `pg_dump` issues `ALTER OWNER` or `SET SESSION AUTHORIZATION` statements to set ownership of created database objects. These statements will fail when the script is run unless it is started by a superuser (or the same user that owns all of the objects in the script). To make a script that can be restored by any user, but will give that user ownership of all the objects, specify `-O`. This option is only meaningful for the plain-text format. For the archive formats, you may specify the option when you call `pg_restore`.

-s | --schema-only

   Dump only the object definitions (schema), not data.

-S username | --superuser=username
Specify the superuser user name to use when disabling triggers. This is only relevant if --disable-triggers is used. It is better to leave this out, and instead start the resulting script as a superuser.

**Note:** Greenplum Database does not support user-defined triggers.

```
-t table | --table=table
```

Dump only tables (or views or sequences) matching the table pattern. Specify the table in the format `schema.table`.

Multiple tables can be selected by writing multiple `-t` switches. Also, the table parameter is interpreted as a pattern according to the same rules used by `psql`'s `\d` commands, so multiple tables can also be selected by writing wildcard characters in the pattern. When using wildcards, be careful to quote the pattern if needed to prevent the shell from expanding the wildcards. The `-n` and `-h` switches have no effect when `-t` is used, because tables selected by `-t` will be dumped regardless of those switches, and non-table objects will not be dumped.

**Note:** When `-t` is specified, `pg_dump` makes no attempt to dump any other database objects that the selected table(s) may depend upon. Therefore, there is no guarantee that the results of a specific-table dump can be successfully restored by themselves into a clean database.

Also, `-t` cannot be used to specify a child table partition. To dump a partitioned table, you must specify the parent table name.

```
-T table | --exclude-table=table
```

Do not dump any tables matching the table pattern. The pattern is interpreted according to the same rules as for `-t`. `-T` can be given more than once to exclude tables matching any of several patterns. When both `-t` and `-T` are given, the behavior is to dump just the tables that match at least one `-t` switch but no `-T` switches. If `-T` appears without `-t`, then tables matching `-T` are excluded from what is otherwise a normal dump.

```
-v | --verbose
```

Specifies verbose mode. This will cause `pg_dump` to output detailed object comments and start/stop times to the dump file, and progress messages to standard error.

```
-x | --no-privileges | --no-acl
```

Prevent dumping of access privileges (`GRANT/REVOKE` commands).

```
--disable-dollar-quoting
```

This option disables the use of dollar quoting for function bodies, and forces them to be quoted using SQL standard string syntax.

```
--disable-triggers
```

This option is only relevant when creating a data-only dump. It instructs `pg_dump` to include commands to temporarily disable triggers on the target tables while the data is reloaded. Use this if you have triggers on the tables that you do not want to invoke during data reload. The commands emitted for `--disable-triggers` must be done as superuser. So, you should also specify a superuser name with `-S`, or preferably be careful to start the resulting script as a superuser. This option is only meaningful for the plain-text format. For the archive formats, you may specify the option when you call `pg_restore`.

**Note:** Greenplum Database does not support user-defined triggers.

```
--use-set-session-authorization
```

Output SQL-standard `SET SESSION AUTHORIZATION` commands instead of `ALTER OWNER` commands to determine object ownership. This makes the dump more standards compatible, but depending on the history of the objects in the dump, may not restore properly. A dump using `SET SESSION AUTHORIZATION` will require superuser privileges to restore correctly, whereas `ALTER OWNER` requires lesser privileges.

**Note:** Greenplum Database does not support user-defined triggers.
--gp-syntax | --no-gp-syntax

Use --gp-syntax to dump Greenplum Database syntax in the CREATE TABLE statements. This allows the distribution policy (DISTRIBUTED BY or DISTRIBUTED RANDOMLY clauses) of a Greenplum Database table to be dumped, which is useful for restoring into other Greenplum Database systems. The default is to include Greenplum Database syntax when connected to a Greenplum Database system, and to exclude it when connected to a regular PostgreSQL system.

-Z 0..9 | --compress=0..9

Specify the compression level to use in archive formats that support compression. Currently only the custom archive format supports compression.

Connection Options
-h host| --host host

The host name of the machine on which the Greenplum Database master database server is running. If not specified, reads from the environment variable PGHOST or defaults to localhost.

-p port| --port port

The TCP port on which the Greenplum Database master database server is listening for connections. If not specified, reads from the environment variable PGPORT or defaults to 5432.

-U username| --username username

The database role name to connect as. If not specified, reads from the environment variable PGUSER or defaults to the current system role name.

-W | --password

Force a password prompt.

Notes

When a data-only dump is chosen and the option --disable-triggers is used, pg_dump emits commands to disable triggers on user tables before inserting the data and commands to re-enable them after the data has been inserted. If the restore is stopped in the middle, the system catalogs may be left in the wrong state.

Members of tar archives are limited to a size less than 8 GB. (This is an inherent limitation of the tar file format.) Therefore this format cannot be used if the textual representation of any one table exceeds that size. The total size of a tar archive and any of the other output formats is not limited, except possibly by the operating system.

The dump file produced by pg_dump does not contain the statistics used by the optimizer to make query planning decisions. Therefore, it is wise to run ANALYZE after restoring from a dump file to ensure good performance.

Examples

Dump a database called mydb into a SQL-script file:

```
pg_dump mydb > db.sql
```

To reload such a script into a (freshly created) database named newdb:

```
psql -d newdb -f db.sql
```

Dump a Greenplum Database in tar file format and include distribution policy information:

```
pg_dump -Ft --gp-syntax mydb > db.tar
```
To dump a database into a custom-format archive file:

```
pg_dump -Fc mydb > db.dump
```

To reload an archive file into a (freshly created) database named `newdb`:

```
pg_restore -d newdb db.dump
```

To dump a single table named `mytab`:

```
pg_dump -t mytab mydb > db.sql
```

To specify an upper-case or mixed-case name in `-t` and related switches, you need to double-quote the name; else it will be folded to lower case. But double quotes are special to the shell, so in turn they must be quoted. Thus, to dump a single table with a mixed-case name, you need something like:

```
pg_dump -t ""MixedCaseName"" mydb > mytab.sql
```

See Also

`pg_dumpall`, `pg_restore`, `psql`

**pg_dumpall**

Extracts all databases in a Greenplum Database system to a single script file or other archive file.

**Synopsis**

```
pg_dumpall [connection_option ...] [dump_option ...]
```

**Description**

`pg_dumpall` is a standard PostgreSQL utility for backing up all databases in a Greenplum Database (or PostgreSQL) instance, and is also supported in Greenplum Database. It creates a single (non-parallel) dump file. For routine backups of Greenplum Database it is better to use the Greenplum Database backup utility, `gpcrondump`, for the best performance.

`pg_dumpall` creates a single script file that contains SQL commands that can be used as input to `psql` to restore the databases. It does this by calling `pg_dump` for each database. `pg_dumpall` also dumps global objects that are common to all databases. (`pg_dump` does not save these objects.) This currently includes information about database users and groups, and access permissions that apply to databases as a whole.

Since `pg_dumpall` reads tables from all databases you will most likely have to connect as a database superuser in order to produce a complete dump. Also you will need superuser privileges to execute the saved script in order to be allowed to add users and groups, and to create databases.

The SQL script will be written to the standard output. Shell operators should be used to redirect it into a file.

`pg_dumpall` needs to connect several times to the Greenplum Database master server (once per database). If you use password authentication it is likely to ask for a password each time. It is convenient to have a `~/.pgpass` file in such cases.

**Note:** The `--ignore-version` option is deprecated and will be removed in a future release.

**Options**

**Dump Options**

```
-a | --data-only
```
Dump only the data, not the schema (data definitions). This option is only meaningful for the plain-text format. For the archive formats, you may specify the option when you call `pg_restore`.

-\texttt{c} | --clean

Output commands to clean (drop) database objects prior to (the commands for) creating them. This option is only meaningful for the plain-text format. For the archive formats, you may specify the option when you call `pg_restore`.

-\texttt{d} | --inserts

Dump data as \texttt{INSERT} commands (rather than \texttt{COPY}). This will make restoration very slow; it is mainly useful for making dumps that can be loaded into non-PostgreSQL-based databases. Also, since this option generates a separate command for each row, an error in reloading a row causes only that row to be lost rather than the entire table contents. Note that the restore may fail altogether if you have rearranged column order. The --D option is safe against column order changes, though even slower.

-\texttt{D} | --column-inserts | --attribute-inserts

Dump data as \texttt{INSERT} commands with explicit column names (\texttt{INSERT INTO table (column, ...) VALUES ...}). This will make restoration very slow; it is mainly useful for making dumps that can be loaded into non-PostgreSQL-based databases. Also, since this option generates a separate command for each row, an error in reloading a row causes only that row to be lost rather than the entire table contents.

-\texttt{F} | --filespaces

Dump filesystem definitions.

-\texttt{f} filename | --file=filename

Send output to the specified file.

-\texttt{g} | --globals-only

Dump only global objects (roles and tablespaces), no databases.

-\texttt{i} | --ignore-version

Note: This option is deprecated and will be removed in a future release.

Ignore version mismatch between `pg_dump` and the database server. `pg_dump` can dump from servers running previous releases of Greenplum Database (or PostgreSQL), but very old versions may not be supported anymore. Use this option if you need to override the version check.

-\texttt{o} | --oids

Dump object identifiers (OIDs) as part of the data for every table. Use of this option is not recommended for files that are intended to be restored into Greenplum Database.

-\texttt{O} | --no-owner

Do not output commands to set ownership of objects to match the original database. By default, `pg_dump` issues ALTER OWNER or SET SESSION AUTHORIZATION statements to set ownership of created database objects. These statements will fail when the script is run unless it is started by a superuser (or the same user that owns all of the objects in the script). To make a script that can be restored by any user, but will give that user ownership of all the objects, specify --O. This option is only meaningful for the plain-text format. For the archive formats, you may specify the option when you call `pg_restore`.

-\texttt{s} | --schema-only

Dump only the object definitions (schema), not data.

-\texttt{S} username | --superuser=username
Specify the superuser user name to use when disabling triggers. This is only relevant if --\n\ndisable-triggers is used. It is better to leave this out, and instead start the resulting\nscript as a superuser.

**Note:** Greenplum Database does not support user-defined triggers.

- **t** | --tablespaces-only
      Dump only tablespaces, not databases or roles.

- **v** | --verbose
      Specifies verbose mode. This will cause `pg_dump` to output detailed object comments and\n      start/stop times to the dump file, and progress messages to standard error.

- **x** | --no-privileges | --no-acl
      Prevent dumping of access privileges (GRANT/REVOKE commands).

- **disable-dollar-quoting**
      This option disables the use of dollar quoting for function bodies, and forces them to be\n      quoted using SQL standard string syntax.

- **disable-triggers**
      This option is only relevant when creating a data-only dump. It instructs `pg_dumpall`\n      to include commands to temporarily disable triggers on the target tables while the data\n      is reloaded. Use this if you have triggers on the tables that you do not want to invoke\n      during data reload. The commands emitted for `--disable-triggers` must be done as\n      superuser. So, you should also specify a superuser name with `-S`, or preferably be careful\n      to start the resulting script as a superuser.

      **Note:** Greenplum Database does not support user-defined triggers.

- **resource-queues**
      Dump resource queue definitions.

- **roles-only**
      Dump only roles, not databases, tablespaces, or filespaces.

- **use-set-session-authorization**
      Output SQL-standard SET SESSION AUTHORIZATION commands instead of ALTER\n      OWNER commands to determine object ownership. This makes the dump more standards\n      compatible, but depending on the history of the objects in the dump, may not restore\n      properly. A dump using SET SESSION AUTHORIZATION will require superuser privileges\n      to restore correctly, whereas ALTER OWNER requires lesser privileges.

- **gp-syntax**
      Output Greenplum Database syntax in the CREATE TABLE statements. This allows\n      the distribution policy (DISTRIBUTED BY or DISTRIBUTED RANDOMLY clauses) of\n      a Greenplum Database table to be dumped, which is useful for restoring into other\n      Greenplum Database systems.

- **no-gp-syntax**
      Do not output the table distribution clauses in the CREATE TABLE statements.

**Connection Options**

- **h host** | --host=host
      The host name of the machine on which the Greenplum master database server is\n      running. If not specified, reads from the environment variable PGHOST or defaults to\n      localhost.

- **l dbname** | --database=dbname
Specifies the name of the database in which to connect to dump global objects. If not specified, the `postgres` database is used. If the `postgres` database does not exist, the `template1` database is used.

```
-p port | --port=port
```

The TCP port on which the Greenplum master database server is listening for connections. If not specified, reads from the environment variable `PGPORT` or defaults to 5432.

```
-U username | --username= username
```

The database role name to connect as. If not specified, reads from the environment variable `PGUSER` or defaults to the current system role name.

```
-w
```

Never prompt for a password.

```
-W | --password
```

Force a password prompt.

**Notes**

Since `pg_dumpall` calls `pg_dump` internally, some diagnostic messages will refer to `pg_dump`. Once restored, it is wise to run `ANALYZE` on each database so the query planner has useful statistics. You can also run `vacuumdb -a -z` to analyze all databases.

`pg_dumpall` requires all needed tablespace (filespace) directories to exist before the restore or database creation will fail for databases in non-default locations.

**Examples**

To dump all databases:

```
pg_dumpall > db.out
```

To reload this file:

```
psql template1 -f db.out
```

To dump only global objects (including filespaces and resource queues):

```
pg_dumpall -g -F --resource-queues
```

**See Also**

`pg_dump`

`pg_restore`

Restores a database from an archive file created by `pg_dump`.

**Synopsis**

```
pg_restore [connection_option ...] [restore_option ...] filename
```

**Description**

`pg_restore` is a utility for restoring a database from an archive created by `pg_dump` in one of the non-plain-text formats. It will issue the commands necessary to reconstruct the database to the state it was in at
the time it was saved. The archive files also allow `pg_restore` to be selective about what is restored, or even to reorder the items prior to being restored.

`pg_restore` can operate in two modes. If a database name is specified, the archive is restored directly into the database. Otherwise, a script containing the SQL commands necessary to rebuild the database is created and written to a file or standard output. The script output is equivalent to the plain text output format of `pg_dump`. Some of the options controlling the output are therefore analogous to `pg_dump` options.

`pg_restore` cannot restore information that is not present in the archive file. For instance, if the archive was made using the "dump data as INSERT commands" option, `pg_restore` will not be able to load the data using `COPY` statements.

**Note:** The `--ignore-version` option is deprecated and will be removed in a future release.

## Options

### `filename`  
Specifies the location of the archive file to be restored. If not specified, the standard input is used.

### Restore Options  

- `a` | `--data-only`  
  Restore only the data, not the schema (data definitions).

- `c` | `--clean`  
  Clean (drop) database objects before recreating them.

- `C` | `--create`  
  Create the database before restoring into it. (When this option is used, the database named with `-d` is used only to issue the initial `CREATE DATABASE` command. All data is restored into the database name that appears in the archive.)

- `d dbname` | `--dbname=dbname`  
  Connect to this database and restore directly into this database. The default is to use the `PGDATABASE` environment variable setting, or the same name as the current system user.

- `e` | `--exit-on-error`  
  Exit if an error is encountered while sending SQL commands to the database. The default is to continue and to display a count of errors at the end of the restoration.

- `f outfilename` | `--file=outfilename`  
  Specify output file for generated script, or for the listing when used with `-l`. Default is the standard output.

- `F t |c` | `--format=tar | custom`  
  The format of the archive produced by `pg_dump`. It is not necessary to specify the format, since `pg_restore` will determine the format automatically. Format can be either `tar` or `custom`.

- `i` | `--ignore-version`  
  **Note:** This option is deprecated and will be removed in a future release. Ignore database version checks.

- `I index` | `--index=index`  
  Restore definition of named index only.

- `l` | `--list`  
  List the contents of the archive. The output of this operation can be used with the `-L` option to restrict and reorder the items that are restored.
-L list-file | --use-list=list-file
    Restore elements in the list-file only, and in the order they appear in the file. Lines can be moved and may also be commented out by placing a ; at the start of the line.

-n schema | --schema=schema
    Restore only objects that are in the named schema. This can be combined with the -t option to restore just a specific table.

-O | --no-owner
    Do not output commands to set ownership of objects to match the original database. By default, pg_restore issues ALTER OWNER or SET SESSION AUTHORIZATION statements to set ownership of created schema elements. These statements will fail unless the initial connection to the database is made by a superuser (or the same user that owns all of the objects in the script). With -O, any user name can be used for the initial connection, and this user will own all the created objects.

-P 'function-name(argtype [, ...])' | --function='function-name(argtype [, ...])'
    Restore the named function only. The function name must be enclosed in quotes. Be careful to spell the function name and arguments exactly as they appear in the dump file's table of contents (as shown by the --list option).

-s | --schema-only
    Restore only the schema (data definitions), not the data (table contents). Sequence current values will not be restored, either. (Do not confuse this with the --schema option, which uses the word schema in a different meaning.)

-S username | --superuser=username
    Specify the superuser user name to use when disabling triggers. This is only relevant if --disable-triggers is used.

    Note: Greenplum Database does not support user-defined triggers.

-t table | --table=table
    Restore definition and/or data of named table only.

-T trigger | --trigger=trigger
    Restore named trigger only.

    Note: Greenplum Database does not support user-defined triggers.

-v | --verbose
    Specifies verbose mode.

-x | --no-privileges | --no-acl
    Prevent restoration of access privileges (GRANT/REVOKE commands).

--disable-triggers
    This option is only relevant when performing a data-only restore. It instructs pg_restore to execute commands to temporarily disable triggers on the target tables while the data is reloaded. Use this if you have triggers on the tables that you do not want to invoke during data reload. The commands emitted for --disable-triggers must be done as superuser. So, you should also specify a superuser name with -S, or preferably run pg_restore as a superuser.

    Note: Greenplum Database does not support user-defined triggers.

--no-data-for-failed-tables
    By default, table data is restored even if the creation command for the table failed (e.g., because it already exists). With this option, data for such a table is skipped. This behavior is useful when the target database may already contain the desired table contents.
Specifying this option prevents duplicate or obsolete data from being loaded. This option is effective only when restoring directly into a database, not when producing SQL script output.

**Connection Options**

- **-h host | --host host**
  The host name of the machine on which the Greenplum master database server is running. If not specified, reads from the environment variable `PGHOST` or defaults to localhost.

- **-p port | --port port**
  The TCP port on which the Greenplum Database master database server is listening for connections. If not specified, reads from the environment variable `PGPORT` or defaults to 5432.

- **-U username | --username username**
  The database role name to connect as. If not specified, reads from the environment variable `PGUSER` or defaults to the current system role name.

- **-W | --password**
  Force a password prompt.

- **-l | --single-transaction**
  Execute the restore as a single transaction. This ensures that either all the commands complete successfully, or no changes are applied.

**Notes**

If your installation has any local additions to the `template1` database, be careful to load the output of `pg_restore` into a truly empty database; otherwise you are likely to get errors due to duplicate definitions of the added objects. To make an empty database without any local additions, copy from `template0` not `template1`, for example:

```
CREATE DATABASE foo WITH TEMPLATE template0;
```

When restoring data to a pre-existing table and the option `--disable-triggers` is used, `pg_restore` emits commands to disable triggers on user tables before inserting the data then emits commands to re-enable them after the data has been inserted. If the restore is stopped in the middle, the system catalogs may be left in the wrong state.

`pg_restore` will not restore large objects for a single table. If an archive contains large objects, then all large objects will be restored.

See also the `pg_dump` documentation for details on limitations of `pg_dump`.

Once restored, it is wise to run `ANALYZE` on each restored table so the query planner has useful statistics.

**Examples**

Assume we have dumped a database called `mydb` into a custom-format dump file:

```
pg_dump -Fc mydb > db.dump
```

To drop the database and recreate it from the dump:

```
dropdb mydb
pg_restore -C -d template1 db.dump
```
To reload the dump into a new database called newdb. Notice there is no -C, we instead connect directly to the database to be restored into. Also note that we clone the new database from template0 not template1, to ensure it is initially empty:

```bash
createdb -T template0 newdb
go_restore -d newdb db.dump
```

To reorder database items, it is first necessary to dump the table of contents of the archive:

```
pgo_restore -l db.dump > db.list
```

The listing file consists of a header and one line for each item, for example,

```
; Archive created at Fri Jul 28 22:28:36 2006
;  dbname: mydb
;    TOC Entries: 74
;    Compression: 0
;    Dump Version: 1.4-0
;    Format: CUSTOM
;
; Selected TOC Entries:
;
2; 145344 TABLE species postgres
3; 145344 ACL species
4; 145359 TABLE nt_header postgres
5; 145359 ACL nt_header
6; 145402 TABLE species_records postgres
7; 145402 ACL species_records
8; 145416 TABLE ss_old postgres
9; 145416 ACL ss_old
10; 145433 TABLE map_resolutions postgres
11; 145433 ACL map_resolutions
12; 145443 TABLE hs_old postgres
13; 145443 ACL hs_old
```

Semicolons start a comment, and the numbers at the start of lines refer to the internal archive ID assigned to each item. Lines in the file can be commented out, deleted, and reordered. For example,

```
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
```

Could be used as input to `pg_restore` and would only restore items 10 and 6, in that order:

```
pgo_restore -L db.list db.dump
```

**See Also**

- `pg_dump`
- `psql`

**Synopsis**

```
psql [option ...] [dbname [username]]
```
Description

`psql` is a terminal-based front-end to Greenplum Database. It enables you to type in queries interactively, issue them to Greenplum Database, and see the query results. Alternatively, input can be from a file. In addition, it provides a number of meta-commands and various shell-like features to facilitate writing scripts and automating a wide variety of tasks.

Options

- **-a | --echo-all**
  Print all input lines to standard output as they are read. This is more useful for script processing rather than interactive mode.

- **-A | --no-align**
  Switches to unaligned output mode. (The default output mode is aligned.)

- **-c 'command' | --command 'command'**
  Specifies that `psql` is to execute the specified command string, and then exit. This is useful in shell scripts. `command` must be either a command string that is completely parseable by the server, or a single backslash command. Thus you cannot mix SQL and `psql` meta-commands with this option. To achieve that, you could pipe the string into `psql`, like this:

  ```
  echo '\x \ SELECT * FROM foo;' | psql
  ```

  (`\x` is the separator meta-command.)

  If the command string contains multiple SQL commands, they are processed in a single transaction, unless there are explicit `BEGIN/COMMIT` commands included in the string to divide it into multiple transactions. This is different from the behavior when the same string is fed to `psql`'s standard input.

- **-d dbname | --dbname dbname**
  Specifies the name of the database to connect to. This is equivalent to specifying `dbname` as the first non-option argument on the command line.

  If this parameter contains an equals sign, it is treated as a `conninfo` string; for example you can pass `dbname=postgres user=username password=mypass` as `dbname`.

- **-e | --echo-queries**
  Copy all SQL commands sent to the server to standard output as well.

- **-E | --echo-hidden**
  Echo the actual queries generated by `\d` and other backslash commands. You can use this to study `psql`'s internal operations.

- **-f filename | --file filename**
  Use a file as the source of commands instead of reading commands interactively. After the file is processed, `psql` terminates. If `filename` is `- (hyphen), then standard input is read. Using this option is subtly different from writing `psql <filename`. In general, both will do what you expect, but using `-f` enables some nice features such as error messages with line numbers.

- **-F separator | --field-separator separator**
  Use the specified separator as the field separator for unaligned output.

- **-H | --html**
  Turn on HTML tabular output.

- **-l | --list**
  List all available databases, then exit. Other non-connection options are ignored.
-L filename | --log-file filename
Write all query output into the specified log file, in addition to the normal output destination.

-o filename | --output filename
Put all query output into the specified file.

-P assignment | --pset assignment
Allows you to specify printing options in the style of \pset on the command line. Note that here you have to separate name and value with an equal sign instead of a space. Thus to set the output format to LaTeX, you could write -P format=latex.

-q | --quiet
Specifies that psql should do its work quietly. By default, it prints welcome messages and various informational output. If this option is used, none of this happens. This is useful with the -c option.

-R separator | --record-separator separator
Use separator as the record separator for unaligned output.

-s | --single-step
Run in single-step mode. That means the user is prompted before each command is sent to the server, with the option to cancel execution as well. Use this to debug scripts.

-S | --single-line
Runs in single-line mode where a new line terminates an SQL command, as a semicolon does.

-t | --tuples-only
Turn off printing of column names and result row count footers, etc. This command is equivalent to \pset tuples_only and is provided for convenience.

-T table_options | --table-attr table_options
Allows you to specify options to be placed within the HTML table tag. See \pset for details.

-v assignment | --set assignment | --variable assignment
Perform a variable assignment, like the \set internal command. Note that you must separate name and value, if any, by an equal sign on the command line. To unset a variable, leave off the equal sign. To just set a variable without a value, use the equal sign but leave off the value. These assignments are done during a very early stage of start-up, so variables reserved for internal purposes might get overwritten later.

-V | --version
Print the psql version and exit.

-x | --expanded
Turn on the expanded table formatting mode.

-X | --no-psqlrc
Do not read the start-up file (neither the system-wide psqlrc file nor the user's ~/.psqlrc file).

-1 | --single-transaction
When psql executes a script with the -f option, adding this option wraps BEGIN/COMMIT around the script to execute it as a single transaction. This ensures that either all the commands complete successfully, or no changes are applied.

If the script itself uses BEGIN, COMMIT, or ROLLBACK, this option will not have the desired effects. Also, if the script contains any command that cannot be executed inside a transaction block, specifying this option will cause that command (and hence the whole transaction) to fail.
-? | --help
  Show help about psql command line arguments, and exit.

Connection Options
-h host | --host host
  The host name of the machine on which the Greenplum master database server is
  running. If not specified, reads from the environment variable PGHOST or defaults to
  localhost.
-p port | --port port
  The TCP port on which the Greenplum master database server is listening for connections.
  If not specified, reads from the environment variable PGPORT or defaults to 5432.
-U username | --username username
  The database role name to connect as. If not specified, reads from the environment
  variable PGUSER or defaults to the current system role name.
-W | --password
  Force a password prompt. psql should automatically prompt for a password whenever the
  server requests password authentication. However, currently password request detection
  is not totally reliable, hence this option to force a prompt. If no password prompt is issued
  and the server requires password authentication, the connection attempt will fail.
-w --no-password
  Never issue a password prompt. If the server requires password authentication and a
  password is not available by other means such as a .pgpass file, the connection attempt
  will fail. This option can be useful in batch jobs and scripts where no user is present to
  enter a password.
  Note: This option remains set for the entire session, and so it affects uses of the meta-
  command \connect as well as the initial connection attempt.

Exit Status
  psql returns 0 to the shell if it finished normally, 1 if a fatal error of its own (out of memory, file not found)
  occurs, 2 if the connection to the server went bad and the session was not interactive, and 3 if an error
  occurred in a script and the variable ON_ERROR_STOP was set.

Usage
Connecting to a Database
  psql is a client application for Greenplum Database. In order to connect to a database you need to know
  the name of your target database, the host name and port number of the Greenplum master server and
  what database user name you want to connect as. psql can be told about those parameters via command
  line options, namely -d, -h, -p, and -U respectively. If an argument is found that does not belong to any
  option it will be interpreted as the database name (or the user name, if the database name is already
  given). Not all these options are required; there are useful defaults. If you omit the host name, psql will
  connect via a UNIX-domain socket to a master server on the local host, or via TCP/IP to localhost on
  machines that do not have UNIX-domain sockets. The default master port number is 5432. If you use a
different port for the master, you must specify the port. The default database user name is your UNIX user
  name, as is the default database name. Note that you cannot just connect to any database under any user
  name. Your database administrator should have informed you about your access rights.

  When the defaults are not right, you can save yourself some typing by setting any or all of the environment
  variables PGAPPNAME, PGDATABASE, PGHOST, PGPORT, and PGUSER to appropriate values.
It is also convenient to have a `~/.pgpass` file to avoid regularly having to type in passwords. This file should reside in your home directory and contain lines of the following format:

```
hostname:port:database:username:password
```

The permissions on `.pgpass` must disallow any access to world or group (for example: `chmod 0600 ~/.pgpass`). If the permissions are less strict than this, the file will be ignored. (The file permissions are not currently checked on Microsoft Windows clients, however.)

If the connection could not be made for any reason (insufficient privileges, server is not running, etc.), `psql` will return an error and terminate.

### Entering SQL Commands

In normal operation, `psql` provides a prompt with the name of the database to which `psql` is currently connected, followed by the string `=>` for a regular user or `=#` for a superuser. For example:

```

```
```
testdb=

testdb=#
```

At the prompt, the user may type in SQL commands. Ordinarily, input lines are sent to the server when a command-terminating semicolon is reached. An end of line does not terminate a command. Thus commands can be spread over several lines for clarity. If the command was sent and executed without error, the results of the command are displayed on the screen.

### Meta-Commands

Anything you enter in `psql` that begins with an unquoted backslash is a `psql` meta-command that is processed by `psql` itself. These commands help make `psql` more useful for administration or scripting. Meta-commands are more commonly called slash or backslash commands.

The format of a `psql` command is the backslash, followed immediately by a command verb, then any arguments. The arguments are separated from the command verb and each other by any number of whitespace characters.

To include whitespace into an argument you may quote it with a single quote. To include a single quote into such an argument, use two single quotes. Anything contained in single quotes is furthermore subject to C-like substitutions for `\n` (new line), `\t` (tab), `\d` (octal), and `\x` (hexadecimal).

If an unquoted argument begins with a colon (`:`), it is taken as a `psql` variable and the value of the variable is used as the argument instead.

Arguments that are enclosed in backquotes (` `) are taken as a command line that is passed to the shell. The output of the command (with any trailing newline removed) is taken as the argument value. The above escape sequences also apply in backquotes.

Some commands take an SQL identifier (such as a table name) as argument. These arguments follow the syntax rules of SQL: Unquoted letters are forced to lowercase, while double quotes (```) protect letters from case conversion and allow incorporation of whitespace into the identifier. Within double quotes, paired double quotes reduce to a single double quote in the resulting name. For example, `FOO"BAR"BAZ` is interpreted as `fooBARbaz`, and `"A weird"" name` becomes `A weird" name`.

Parsing for arguments stops when another unquoted backslash occurs. This is taken as the beginning of a new meta-command. The special sequence `\ ` (two backslashes) marks the end of arguments and continues parsing SQL commands, if any. That way SQL and `psql` commands can be freely mixed on a line. But in any case, the arguments of a meta-command cannot continue beyond the end of the line.

The following meta-commands are defined:

\a
If the current table output format is unaligned, it is switched to aligned. If it is not unaligned, it is set to unaligned. This command is kept for backwards compatibility. See \pset for a more general solution.

\cd \[directory\]
Changes the current working directory. Without argument, changes to the current user’s home directory. To print your current working directory, use \!pwd.

\C \[title\]
Sets the title of any tables being printed as the result of a query or unset any such title. This command is equivalent to \pset title.

\c | \connect \[dbname [username] [host] [port]\]
Establishes a new connection. If the new connection is successfully made, the previous connection is closed. If any of dbname, username, host or port are omitted, the value of that parameter from the previous connection is used. If the connection attempt failed, the previous connection will only be kept if psql is in interactive mode. When executing a non-interactive script, processing will immediately stop with an error. This distinction was chosen as a user convenience against typos, and a safety mechanism that scripts are not accidentally acting on the wrong database.

\conninfo
Displays information about the current connection including the database name, the user name, the type of connection (UNIX domain socket, TCP/IP, etc.), the host, and the port.

\copy \{table \[(column_list)\] | (query)\} \{from \| to\} \{'filename' \| stdin \| stdout \| pstdin | pstdout\} \[with\] \[binary\] \[oids\] \[delimiter \[as\] 'character'\] \[null \[as\] 'string'\] \[csv \[header\] \[quote \[as\] 'character'\] \[escape \[as\] 'character'\] \[force quote column_list\] \[force not null column_list\]
Performs a frontend (client) copy. This is an operation that runs an SQL COPY command, but instead of the server reading or writing the specified file, psql reads or writes the file and routes the data between the server and the local file system. This means that file accessibility and privileges are those of the local user, not the server, and no SQL superuser privileges are required.

The syntax of the command is similar to that of the SQL COPY command. Note that, because of this, special parsing rules apply to the \copy command. In particular, the variable substitution rules and backslash escapes do not apply.

\copy ... from stdin \| to stdout reads/writes based on the command input and output respectively. All rows are read from the same source that issued the command, continuing until \. is read or the stream reaches EOF. Output is sent to the same place as command output. To read/write from psql's standard input or output, use pstdin or pstdout. This option is useful for populating tables in-line within a SQL script file.

This operation is not as efficient as the SQL COPY command because all data must pass through the client/server connection.

\copyright
Shows the copyright and distribution terms of PostgreSQL on which Greenplum Database is based.

\d \[relation_pattern\] | \d+ \[relation_pattern\] | \dS \[relation_pattern\]
For each relation (table, external table, view, index, or sequence) matching the relation pattern, show all columns, their types, the tablespace (if not the default) and any special attributes such as NOT NULL or defaults, if any. Associated indexes, constraints, rules, and triggers are also shown, as is the view definition if the relation is a view.
• The command form \d+ is identical, except that more information is displayed: any comments associated with the columns of the table are shown, as is the presence of OIDs in the table.

For partitioned tables, the command \d or \d+ specified with the root partition table or child partition table displays information about the table including partition keys on the current level of the partition table. The command \d+ also displays the immediate child partitions of the table and whether the child partition is an external table or regular table.

For append-optimized tables and column-oriented tables, \d+ displays the storage options for a table. For append-optimized tables, the options are displayed for the table. For column-oriented tables, storage options are displayed for each column.

• The command form \ds is identical, except that system information is displayed as well as user information. For example, \dt displays user tables, but not system tables; \dt$s$ displays both user and system tables. Both these commands can take the + parameter to display additional information, as in \dt+ and \dt$s$.

If \d is used without a pattern argument, it is equivalent to \dt$v$s which will show a list of all tables, views, and sequences.

\da [aggregate_pattern]
  Lists all available aggregate functions, together with the data types they operate on. If a pattern is specified, only aggregates whose names match the pattern are shown.

\db [tablespace_pattern] | \db+ [tablespace_pattern]
  Lists all available tablespaces and their corresponding filespace locations. If pattern is specified, only tablespaces whose names match the pattern are shown. If + is appended to the command name, each object is listed with its associated permissions.

\dc [conversion_pattern]
  Lists all available conversions between character-set encodings. If pattern is specified, only conversions whose names match the pattern are listed.

\dC
  Lists all available type casts.

\dd [object_pattern]
  Lists all available objects. If pattern is specified, only matching objects are shown.

\dD [domain_pattern]
  Lists all available domains. If pattern is specified, only matching domains are shown.

\dE [relation_pattern]
  Lists all external tables, or only those that match the pattern.

\df [function_pattern] | \df+ [function_pattern ]
  Lists available functions, together with their argument and return types. If pattern is specified, only functions whose names match the pattern are shown. If the form \df+ is used, additional information about each function, including language and description, is shown. To reduce clutter, \df+ does not show data type I/O functions. This is implemented by ignoring functions that accept or return type cstring.

\dg [role_pattern]
  Lists all database roles. If pattern is specified, only those roles whose names match the pattern are listed.

\distPvES [index | sequence | table | parent table | view | external_table | system_object]
  This is not the actual command name: the letters i, s, t, P, v, E, S stand for index, sequence, table, parent table, view, external table, and system table, respectively. You
can specify any or all of these letters, in any order, to obtain a listing of all the matching objects. The letter S restricts the listing to system objects; without S, only non-system objects are shown. If + is appended to the command name, each object is listed with its associated description, if any. If a pattern is specified, only objects whose names match the pattern are listed.

\dl

This is an alias for \lo_list, which shows a list of large objects.

\dn [schema_pattern] | \dn+ [schema_pattern]

Lists all available schemas (namespaces). If pattern is specified, only schemas whose names match the pattern are listed. Non-local temporary schemas are suppressed. If + is appended to the command name, each object is listed with its associated permissions and description, if any.

\do [operator_pattern]

Lists available operators with their operand and return types. If pattern is specified, only operators whose names match the pattern are listed.

\dp [relation_pattern_to_show_privileges]

Produces a list of all available tables, views and sequences with their associated access privileges. If pattern is specified, only tables, views and sequences whose names match the pattern are listed. The GRANT and REVOKE commands are used to set access privileges.

\dT [datatype_pattern] | \dT+ [datatype_pattern]

Lists all data types or only those that match pattern. The command form \dT+ shows extra information.

\du [role_pattern]

Lists all database roles, or only those that match pattern.

\dx [extension_pattern] | \dx+ [extension_pattern]

Lists all installed extensions, or only those that match the pattern. \dx and \dx+ are functionally equivalent.

\e | \edit [filename]

If a file name is specified, the file is edited; after the editor exits, its content is copied back to the query buffer. If no argument is given, the current query buffer is copied to a temporary file which is then edited in the same fashion. The new query buffer is then re-parsed according to the normal rules of psql, where the whole buffer is treated as a single line. (Thus you cannot make scripts this way. Use \i for that.) This means also that if the query ends with (or rather contains) a semicolon, it is immediately executed. In other cases it will merely wait in the query buffer.

psql searches the environment variables PSQL_EDITOR, EDITOR, and VISUAL (in that order) for an editor to use. If all of them are unset, vi is used on UNIX systems, notepad.exe on Windows systems.

\echo text [ ... ]

Prints the arguments to the standard output, separated by one space and followed by a newline. This can be useful to intersperse information in the output of scripts.

If you use the \o command to redirect your query output you may wish to use \qecho instead of this command.

\encoding [encoding]

Sets the client character set encoding. Without an argument, this command shows the current encoding.

\f [field_separator_string]
Sets the field separator for unaligned query output. The default is the vertical bar (|). See also \pset for a generic way of setting output options.

\g [{filename | command}]

Sends the current query input buffer to the server and optionally stores the query's output in a file or pipes the output into a separate UNIX shell executing command. A bare \g is virtually equivalent to a semicolon. A \g with argument is a one-shot alternative to the \o command.

\h | \help [sql_command]

Gives syntax help on the specified SQL command. If a command is not specified, then psql will list all the commands for which syntax help is available. Use an asterisk (*) to show syntax help on all SQL commands. To simplify typing, commands that consist of several words do not have to be quoted.

\H

Turns on HTML query output format. If the HTML format is already on, it is switched back to the default aligned text format. This command is for compatibility and convenience, but see \pset about setting other output options.

\i input_filename

Reads input from a file and executes it as though it had been typed on the keyboard. If you want to see the lines on the screen as they are read you must set the variable ECHO to all.

\l | \list | \l+ | \list+

List the names, owners, and character set encodings of all the databases in the server. If + is appended to the command name, database descriptions are also displayed.

\lo_export loid filename

Reads the large object with OID loid from the database and writes it to filename. Note that this is subtly different from the server function lo_export, which acts with the permissions of the user that the database server runs as and on the server's file system. Use \lo_list to find out the large object's OID.

\lo_import large_object_filename [comment]

Stores the file into a large object. Optionally, it associates the given comment with the object. Example:

mydb=> \lo_import '/home/gpadmin/pictures/photo.xcf' 'a picture of me'
lo_import 152801

The response indicates that the large object received object ID 152801 which one ought to remember if one wants to access the object ever again. For that reason it is recommended to always associate a human-readable comment with every object. Those can then be seen with the \lo_list command. Note that this command is subtly different from the server-side lo_import because it acts as the local user on the local file system, rather than the server's user and file system.

\lo_list

Shows a list of all large objects currently stored in the database, along with any comments provided for them.

\lo_unlink largeobject_oid

Deletes the large object of the specified OID from the database. Use \lo_list to find out the large object's OID.

\o [{query_result_filename | command}]

Saves future query results to a file or pipes them into a UNIX shell command. If no arguments are specified, the query output will be reset to the standard output. Query
results include all tables, command responses, and notices obtained from the database server, as well as output of various backslash commands that query the database (such as `\d`), but not error messages. To intersperse text output in between query results, use `\echo`.

\p

Print the current query buffer to the standard output.

\password [username]

Changes the password of the specified user (by default, the current user). This command prompts for the new password, encrypts it, and sends it to the server as an `ALTER ROLE` command. This makes sure that the new password does not appear in cleartext in the command history, the server log, or elsewhere.

\prompt [ text ] name

Prompts the user to set a variable `name`. Optionally, you can specify a prompt. Enclose prompts longer than one word in single quotes.

By default, `\prompt` uses the terminal for input and output. However, use the `-f` command line switch to specify standard input and standard output.

\pset print_option [value]

This command sets options affecting the output of query result tables. `print_option` describes which option is to be set. Adjustable printing options are:

- **format** – Sets the output format to one of `unaligned`, `aligned`, `html`, `latex`, `troff-ms`, or `wrapped`. First letter abbreviations are allowed. Unaligned writes all columns of a row on a line, separated by the currently active field separator. This is intended to create output that might be intended to be read in by other programs. Aligned mode is the standard, human-readable, nicely formatted text output that is default. The HTML and LaTeX modes put out tables that are intended to be included in documents using the respective mark-up language. They are not complete documents! (This might not be so dramatic in HTML, but in LaTeX you must have a complete document wrapper.)

  The wrapped option sets the output format like the `aligned` parameter, but wraps wide data values across lines to make the output fit in the target column width. The target width is set with the `columns` option. To specify the column width and select the wrapped format, use two `\pset` commands; for example, to set the width to 72 columns and specify wrapped format, use the commands `\pset columns 72` and then `\pset format wrapped`.

  **Note:** Since `psql` does not attempt to wrap column header titles, the wrapped format behaves the same as aligned if the total width needed for column headers exceeds the target.

- **border** – The second argument must be a number. In general, the higher the number the more borders and lines the tables will have, but this depends on the particular format. In HTML mode, this will translate directly into the `border=...` attribute, in the others only values 0 (no border), 1 (internal dividing lines), and 2 (table frame) make sense.

- **columns** – Sets the target width for the `wrapped` format, and also the width limit for determining whether output is wide enough to require the pager. The default is zero. Zero causes the target width to be controlled by the environment variable `COLUMNS`, or the detected screen width if `COLUMNS` is not set. In addition, if `columns` is zero then the wrapped format affects screen output only. If columns is nonzero then file and pipe output is wrapped to that width as well.

After setting the target width, use the command `\pset format wrapped` to enable the wrapped format.
• **expanded | x** – Toggles between regular and expanded format. When expanded format is enabled, query results are displayed in two columns, with the column name on the left and the data on the right. This mode is useful if the data would not fit on the screen in the normal horizontal mode. Expanded mode is supported by all four output formats.

• **linestyle [unicode | asci | old-asci]** – Sets the border line drawing style to one of unicode, asci, or old-asci. Unique abbreviations, including one letter, are allowed for the three styles. The default setting is asci. This option only affects the aligned and wrapped output formats.

  asci – uses plain ASCII characters. Newlines in data are shown using a + symbol in the right-hand margin. When the wrapped format wraps data from one line to the next without a newline character, a dot (.) is shown in the right-hand margin of the first line, and again in the left-hand margin of the following line.

  old-asci – style uses plain ASCII characters, using the formatting style used in PostgreSQL 8.4 and earlier. Newlines in data are shown using a : symbol in place of the left-hand column separator. When the data is wrapped from one line to the next without a newline character, a ; symbol is used in place of the left-hand column separator.

  unicode – style uses Unicode box-drawing characters. Newlines in data are shown using a carriage return symbol in the right-hand margin. When the data is wrapped from one line to the next without a newline character, an ellipsis symbol is shown in the right-hand margin of the first line, and again in the left-hand margin of the following line.

When the border setting is greater than zero, this option also determines the characters with which the border lines are drawn. Plain ASCII characters work everywhere, but Unicode characters look nicer on displays that recognize them.

• **null ’string’** – The second argument is a string to print whenever a column is null. The default is not to print anything, which can easily be mistaken for an empty string. For example, the command \texttt{\psetnull ’(empty)’} displays \texttt{(empty)} in null columns.

• **fieldsep** – Specifies the field separator to be used in unaligned output mode. That way one can create, for example, tab- or comma-separated output, which other programs might prefer. To set a tab as field separator, type \texttt{\pset fieldsep ’\t’}. The default field separator is \texttt{’|’} (a vertical bar).

• **footer** – Toggles the display of the default footer \texttt{(x rows)}.

• **numericlocale** – Toggles the display of a locale-aware character to separate groups of digits to the left of the decimal marker. It also enables a locale-aware decimal marker.

• **recordsep** – Specifies the record (line) separator to use in unaligned output mode. The default is a newline character.

• **title [text]** – Sets the table title for any subsequently printed tables. This can be used to give your output descriptive tags. If no argument is given, the title is unset.

• **tableattr \ T [text]** – Allows you to specify any attributes to be placed inside the HTML table tag. This could for example be \texttt{cellpadding} or \texttt{bgcolor}. Note that you probably don’t want to specify border here, as that is already taken care of by \texttt{\pset border}.

• **tuples_only | t [novalue | on | off]** – The \texttt{\pset tuples_only} command by itself toggles between tuples only and full display. The values \texttt{on} and \texttt{off} set the tuples display, regardless of the current setting. Full display may show extra information such as column headers, titles, and various footers. In tuples only mode, only actual table data is shown. The \texttt{\t} command is equivalent to \texttt{\psettuples_only} and is provided for convenience.

• **pager** – Controls the use of a pager for query and \texttt{psql} help output. When \texttt{on}, if the environment variable \texttt{PAGER} is set, the output is piped to the specified program.
Otherwise a platform-dependent default (such as `more`) is used. When `off`, the pager is not used. When `on`, the pager is used only when appropriate. Pager can also be set to `always`, which causes the pager to be always used.

\q
Quits the `psql` program.

\qecho text [ ... ]
This command is identical to `\echo` except that the output will be written to the query output channel, as set by `\o`.

\r
Resets (clears) the query buffer.

\s [history_filename]
Print or save the command line history to `filename`. If `filename` is omitted, the history is written to the standard output.

\set [name [value [ ... ]]]
Sets the internal variable `name` to `value` or, if more than one value is given, to the concatenation of all of them. If no second argument is given, the variable is just set with no value. To unset a variable, use the `\unset` command.

Valid variable names can contain characters, digits, and underscores. See "Variables" in Advanced Features. Variable names are case-sensitive.

Although you are welcome to set any variable to anything you want, `psql` treats several variables as special. They are documented in the topic about variables. This command is totally separate from the SQL command `SET`.

\t [novalue | on | off]
The `\t` command by itself toggles a display of output column name headings and row count footer. The values `on` and `off` set the tuples display, regardless of the current setting. This command is equivalent to `\pset tuples_only` and is provided for convenience.

\T table_options
Allows you to specify attributes to be placed within the table tag in HTML tabular output mode.

\timing [novalue | on | off]
The `\timing` command by itself toggles a display of how long each SQL statement takes, in milliseconds. The values `on` and `off` set the time display, regardless of the current setting.

\w {filename | |command}
Outputs the current query buffer to a file or pipes it to a UNIX command.

\x
Toggles expanded table formatting mode.

\z [relation_to_show_privileges]
Produces a list of all available tables, views and sequences with their associated access privileges. If a pattern is specified, only tables, views and sequences whose names match the pattern are listed. This is an alias for `\dp`.

\! [command]
Escapes to a separate UNIX shell or executes the UNIX command. The arguments are not further interpreted, the shell will see them as is.

\?
Shows help information about the `psql` backslash commands.

**Patterns**

The various `\d` commands accept a pattern parameter to specify the object name(s) to be displayed. In the simplest case, a pattern is just the exact name of the object. The characters within a pattern are normally folded to lower case, just as in SQL names; for example, `\dt FOO` will display the table named `foo`. As in SQL names, placing double quotes around a pattern stops folding to lower case. Should you need to include an actual double quote character in a pattern, write it as a pair of double quotes within a double-quote sequence; again this is in accord with the rules for SQL quoted identifiers. For example, `\dt "FOO""BAR"` will display the table named `FOO"BAR` (not `foo"bar`). Unlike the normal rules for SQL names, you can put double quotes around just part of a pattern, for instance `\dt FOO"FOO"BAR` will display the table named `fooFOObar`.

Within a pattern, `*` matches any sequence of characters (including no characters) and `?` matches any single character. (This notation is comparable to UNIX shell file name patterns.) For example, `\dt int*` displays all tables whose names begin with `int`. But within double quotes, `*` and `?` lose these special meanings and are just matched literally.

A pattern that contains a dot (`.`) is interpreted as a schema name pattern followed by an object name pattern. For example, `\dt foo*.bar*` displays all tables whose table name starts with `bar` that are in schemas whose schema name starts with `foo`. When no dot appears, then the pattern matches only objects that are visible in the current schema search path. Again, a dot within double quotes loses its special meaning and is matched literally.

Advanced users can use regular-expression notations. All regular expression special characters work as specified in the PostgreSQL documentation on regular expressions, except for `.` which is taken as a separator as mentioned above, `*` which is translated to the regular-expression notation `.*`, and `?` which is translated to `.`. You can emulate these pattern characters at need by writing `?` for `..`, `(R|)` for `R*`, or `(R+)` for `R?`. Remember that the pattern must match the whole name, unlike the usual interpretation of regular expressions; write `*` at the beginning and/or end if you don't wish the pattern to be anchored. Note that within double quotes, all regular expression special characters lose their special meanings and are matched literally. Also, the regular expression special characters are matched literally in operator name patterns (such as the argument of `\do`).

Whenever the pattern parameter is omitted completely, the `\d` commands display all objects that are visible in the current schema search path – this is equivalent to using the pattern `*`. To see all objects in the database, use the pattern `*.*`.

**Advanced Features**

**Variables**

`psql` provides variable substitution features similar to common UNIX command shells. Variables are simply name/value pairs, where the value can be any string of any length. To set variables, use the `psql` meta-command `\set`:

```
  testdb=> \set foo bar
```

sets the variable `foo` to the value `bar`. To retrieve the content of the variable, precede the name with a colon and use it as the argument of any slash command:

```
  testdb=> \echo :foo
  bar
```

**Note:** The arguments of `\set` are subject to the same substitution rules as with other commands. Thus you can construct interesting references such as `\set :foo 'something'` and get 'soft links' or 'variable variables' of Perl or PHP fame, respectively. Unfortunately, there is no way to do anything useful with these constructs. On the other hand, `\set bar :foo` is a perfectly valid way to copy a variable.
If you call \set without a second argument, the variable is set, with an empty string as value. To unset (or delete) a variable, use the command \unset.

psql's internal variable names can consist of letters, numbers, and underscores in any order and any number of them. A number of these variables are treated specially by psql. They indicate certain option settings that can be changed at run time by altering the value of the variable or represent some state of the application. Although you can use these variables for any other purpose, this is not recommended, as the program behavior might behave unexpectedly. By convention, all specially treated variables consist of all upper-case letters (and possibly numbers and underscores). To ensure maximum compatibility in the future, avoid using such variable names for your own purposes. A list of all specially treated variables are as follows:

**AUTOCOMMIT**

When on (the default), each SQL command is automatically committed upon successful completion. To postpone commit in this mode, you must enter a \begin or \begin{} SQL command. When off or unset, SQL commands are not committed until you explicitly issue \commit or \end{}. The autocommit-on mode works by issuing an implicit \begin{} for you, just before any command that is not already in a transaction block and is not itself a \begin{} or other transaction-control command, nor a command that cannot be executed inside a transaction block (such as \vacuum).

In autocommit-off mode, you must explicitly abandon any failed transaction by entering \abort or \rollback. Also keep in mind that if you exit the session without committing, your work will be lost.

The autocommit-on mode is PostgreSQL's traditional behavior, but autocommit-off is closer to the SQL spec. If you prefer autocommit-off, you may wish to set it in your \~/.psqlrc file.

**DBNAME**

The name of the database you are currently connected to. This is set every time you connect to a database (including program start-up), but can be unset.

**ECHO**

If set to all, all lines entered from the keyboard or from a script are written to the standard output before they are parsed or executed. To select this behavior on program start-up, use the switch \-a. If set to queries, psql merely prints all queries as they are sent to the server. The switch for this is \-e.

**ECHO_HIDDEN**

When this variable is set and a backslash command queries the database, the query is first shown. This way you can study the Greenplum Database internals and provide similar functionality in your own programs. (To select this behavior on program start-up, use the switch \-E.) If you set the variable to the value noexec, the queries are just shown but are not actually sent to the server and executed.

**ENCODING**

The current client character set encoding.

**FETCH_COUNT**

If this variable is set to an integer value > 0, the results of \select{} queries are fetched and displayed in groups of that many rows, rather than the default behavior of collecting the entire result set before display. Therefore only a limited amount of memory is used, regardless of the size of the result set. Settings of 100 to 1000 are commonly used when enabling this feature. Keep in mind that when using this feature, a query may fail after having already displayed some rows.

Although you can use any output format with this feature, the default aligned format tends to look bad because each group of FETCH_COUNT rows will be formatted separately,
leading to varying column widths across the row groups. The other output formats work better.

**HISTCONTROL**

If this variable is set to `ignorespace`, lines which begin with a space are not entered into the history list. If set to a value of `ignoredups`, lines matching the previous history line are not entered. A value of `ignoreboth` combines the two options. If unset, or if set to any other value than those above, all lines read in interactive mode are saved on the history list.

**HISTFILE**

The file name that will be used to store the history list. The default value is `~/.psql_history`. For example, putting

```
\set HISTFILE ~/.psql_history- :DBNAME
```

in `~/.psqlrc` will cause `psql` to maintain a separate history for each database.

**HISTSIZE**

The number of commands to store in the command history. The default value is 500.

**HOST**

The database server host you are currently connected to. This is set every time you connect to a database (including program start-up), but can be unset.

**IGNOREEOF**

If unset, sending an EOF character (usually `CTRL+D`) to an interactive session of `psql` will terminate the application. If set to a numeric value, that many EOF characters are ignored before the application terminates. If the variable is set but has no numeric value, the default is 10.

**LASTOID**

The value of the last affected OID, as returned from an `INSERT` or `lo_insert` command. This variable is only guaranteed to be valid until after the result of the next SQL command has been displayed.

**ON_ERROR_ROLLBACK**

When on, if a statement in a transaction block generates an error, the error is ignored and the transaction continues. When interactive, such errors are only ignored in interactive sessions, and not when reading script files. When off (the default), a statement in a transaction block that generates an error aborts the entire transaction. The `on_error_rollback-on` mode works by issuing an implicit `SAVEPOINT` for you, just before each command that is in a transaction block, and rolls back to the savepoint on error.

**ON_ERROR_STOP**

By default, if non-interactive scripts encounter an error, such as a malformed SQL command or internal meta-command, processing continues. This has been the traditional behavior of `psql` but it is sometimes not desirable. If this variable is set, script processing will immediately terminate. If the script was called from another script it will terminate in the same fashion. If the outermost script was not called from an interactive `psql` session but rather using the `-f` option, `psql` will return error code 3, to distinguish this case from fatal error conditions (error code 1).

**PORT**

The database server port to which you are currently connected. This is set every time you connect to a database (including program start-up), but can be unset.

**PROMPT1**

**PROMPT2**

**PROMPT3**
These specify what the prompts `psql` issues should look like. See "Prompting".

**QUIET**
This variable is equivalent to the command line option `-q`. It is not very useful in interactive mode.

**SINGLELINE**
This variable is equivalent to the command line option `-S`.

**SINGLESTEP**
This variable is equivalent to the command line option `-s`.

**USER**
The database user you are currently connected as. This is set every time you connect to a database (including program start-up), but can be unset.

**VERBOSITY**
This variable can be set to the values default, verbose, or terse to control the verbosity of error reports.

**SQL Interpolation**
An additional useful feature of `psql` variables is that you can substitute (interpolate) them into regular SQL statements. The syntax for this is again to prepend the variable name with a colon (`:`).

```
testdb=> \set foo 'my_table'
testdb=> SELECT * FROM :foo;
```

would then query the table `my_table`. The value of the variable is copied literally, so it can even contain unbalanced quotes or backslash commands. You must make sure that it makes sense where you put it. Variable interpolation will not be performed into quoted SQL entities.

A popular application of this facility is to refer to the last inserted OID in subsequent statements to build a foreign key scenario. Another possible use of this mechanism is to copy the contents of a file into a table column. First load the file into a variable and then proceed as above.

```
testdb=> \set content '''' `cat my_file.txt` '''
testdb=> INSERT INTO my_table VALUES (:content);
```

One problem with this approach is that `my_file.txt` might contain single quotes. These need to be escaped so that they don't cause a syntax error when the second line is processed. This could be done with the program `sed`:

```
testdb=> \set content '''' `sed -e "s/'/''/g" < my_file.txt` '''
```

If you are using non-standard-conforming strings then you'll also need to double backslashes. This is a bit tricky:

```
testdb=> \set content '''' `sed -e "s/'/''/g" -e 's/\\/\\/g' < my_file.txt` '''
```

Note the use of different shell quoting conventions so that neither the single quote marks nor the backslashes are special to the shell. Backslashes are still special to `sed`, however, so we need to double them.

Since colons may legally appear in SQL commands, the following rule applies: the character sequence `":name"` is not changed unless "name" is the name of a variable that is currently set. In any case you can escape a colon with a backslash to protect it from substitution. (The colon syntax for variables is standard SQL for embedded query languages, such as ECPG. The colon syntax for array slices and type casts are Greenplum Database extensions, hence the conflict.)
Prompting

The prompts psql issues can be customized to your preference. The three variables PROMPT1, PROMPT2, and PROMPT3 contain strings and special escape sequences that describe the appearance of the prompt. Prompt 1 is the normal prompt that is issued when psql requests a new command. Prompt 2 is issued when more input is expected during command input because the command was not terminated with a semicolon or a quote was not closed. Prompt 3 is issued when you run an SQL COPY command and you are expected to type in the row values on the terminal.

The value of the selected prompt variable is printed literally, except where a percent sign (%) is encountered. Depending on the next character, certain other text is substituted instead. Defined substitutions are:

- **%M**: The full host name (with domain name) of the database server, or [local] if the connection is over a UNIX domain socket, or [local:/dir/name], if the UNIX domain socket is not at the compiled in default location.

- **%m**: The host name of the database server, truncated at the first dot, or [local] if the connection is over a UNIX domain socket.

- **%>**: The port number at which the database server is listening.

- **%n**: The database session user name. (The expansion of this value might change during a database session as the result of the command SET SESSION AUTHORIZATION.)

- **%/**: The name of the current database.

- **%~**: Like %/, but the output is ~ (tilde) if the database is your default database.

- **%#**: If the session user is a database superuser, then a #, otherwise a >. (The expansion of this value might change during a database session as the result of the command SET SESSION AUTHORIZATION.)

- **%R**: In prompt 1 normally =, but ^ if in single-line mode, and ! if the session is disconnected from the database (which can happen if \connect fails). In prompt 2 the sequence is replaced by -, *, a single quote, a double quote, or a dollar sign, depending on whether psql expects more input because the command wasn't terminated yet, because you are inside a /* ... */ comment, or because you are inside a quoted or dollar-escaped string. In prompt 3 the sequence doesn't produce anything.

- **%x**: Transaction status: an empty string when not in a transaction block, or * when in a transaction block, or ! when in a failed transaction block, or ? when the transaction state is indeterminate (for example, because there is no connection).

- **%digits**: The character with the indicated octal code is substituted.

- **%:name**: The value of the psql variable name. See "Variables" in Advanced Features for details.

- **% `command`**: The output of command, similar to ordinary back-tick substitution.
Prompts may contain terminal control characters which, for example, change the color, background, or style of the prompt text, or change the title of the terminal window. In order for line editing to work properly, these non-printing control characters must be designated as invisible by surrounding them with \%[ and \%]. Multiple pairs of these may occur within the prompt. For example,

```
testdb=> \set PROMPT1 '%[033[1;33;40m%]%n@%/%R%[033[0m%%'
```
results in a boldfaced (1;) yellow-on-black (33;40) prompt on VT100-compatible, color-capable terminals. To insert a percent sign into your prompt, write \%. The default prompts are '%/%R%# ' for prompts 1 and 2, and '>> ' for prompt 3.

**Command-Line Editing**

`psql` supports the NetBSD libedit library for convenient line editing and retrieval. The command history is automatically saved when `psql` exits and is reloaded when `psql` starts up. Tab-completion is also supported, although the completion logic makes no claim to be an SQL parser. If for some reason you do not like the tab completion, you can turn it off by putting this in a file named `.inputrc` in your home directory:

```
$if psql
   set disable-completion on
$endif
```

**Environment**

**PAGER**

If the query results do not fit on the screen, they are piped through this command. Typical values are more or less. The default is platform-dependent. The use of the pager can be disabled by using the `\pset` command.

**PGDATABASE**

**PGHOST**

**PGPORT**

**PGUSER**

Default connection parameters.

**PSQL_EDITOR**

**EDITOR**

**VISUAL**

Editor used by the `\e` command. The variables are examined in the order listed; the first that is set is used.

**SHELL**

Command executed by the `\!` command.

**TMPDIR**

Directory for storing temporary files. The default is `/tmp`.

**Files**

Before starting up, `psql` attempts to read and execute commands from the user's `~/.psqlrc` file. The command-line history is stored in the file `~/.psql_history`.
**Notes**

`psql` only works smoothly with servers of the same version. That does not mean other combinations will fail outright, but subtle and not-so-subtle problems might come up. Backslash commands are particularly likely to fail if the server is of a different version.

**Notes for Windows users**

`psql` is built as a console application. Since the Windows console windows use a different encoding than the rest of the system, you must take special care when using 8-bit characters within `psql`. If `psql` detects a problematic console code page, it will warn you at startup. To change the console code page, two things are necessary:

Set the code page by entering:

```
cmd.exe /c chcp 1252
```

1252 is a character encoding of the Latin alphabet, used by Microsoft Windows for English and some other Western languages. If you are using Cygwin, you can put this command in `/etc/profile`.

Set the console font to Lucida Console, because the raster font does not work with the ANSI code page.

**Examples**

Start `psql` in interactive mode:

```
psql -p 54321 -U sally mydatabase
```

In `psql` interactive mode, spread a command over several lines of input. Notice the changing prompt:

```
testdb=> CREATE TABLE my_table ( 
testdb(> first integer not null default 0, 
testdb(> second text) 
testdb(> ; 
CREATE TABLE
```

Look at the table definition:

```
testdb=> \d my_table 
 Table "my_table"
 Attribute | Type | Modifier
------------|------|-------------------
 first | integer | not null default 0
 second | text |
```

Run `psql` in non-interactive mode by passing in a file containing SQL commands:

```
psql -f /home/gpadmin/test/myscript.sql
```

**reindexdb**

Rebuilds indexes in a database.

**Synopsis**

```
reindexdb [connection-option ...] [--table | -t table ] 
[--index | -i index ] [dbname]
reindexdb [connection-option ...] [--all | -a]
```
reindexdb [connection-option ...] [--system | -s] [dbname]

reindexdb --help

reindexdb --version

Description

reindexdb is a utility for rebuilding indexes in Greenplum Database, and is a wrapper around the SQL command REINDEX.

Options

-a | --all
Reindex all databases.

[-d] dbname | [--dbname] dbname
Specifies the name of the database to be reindexed. If this is not specified and -all is not used, the database name is read from the environment variable PGDATABASE. If that is not set, the user name specified for the connection is used.

-e | --echo
Echo the commands that reindexdb generates and sends to the server.

-i index | --index index
Recreate index only.

-q | --quiet
Do not display a response.

-s | --system
Reindex system catalogs.

-t table | --table table
Reindex table only.

Connection Options

-h host | --host host
Specifies the host name of the machine on which the Greenplum master database server is running. If not specified, reads from the environment variable PGHOST or defaults to localhost.

-p port | --port port
Specifies the TCP port on which the Greenplum master database server is listening for connections. If not specified, reads from the environment variable PGPORT or defaults to 5432.

-U username | --username username
The database role name to connect as. If not specified, reads from the environment variable PGUSER or defaults to the current system user name.

-w | --no-password
Never issue a password prompt. If the server requires password authentication and a password is not available by other means such as a .pgpass file, the connection attempt will fail. This option can be useful in batch jobs and scripts where no user is present to enter a password.

-W | --password
Force a password prompt.
Notes

reindexdb might need to connect several times to the master server, asking for a password each time. It is convenient to have a ~/.pgpass file in such cases.

Examples

To reindex the database mydb:

reindexdb mydb

To reindex the table foo and the index bar in a database named abcd:

reindexdb --table foo --index bar abcd

See Also

REINDEX in the Greenplum Database Reference Guide

vacuumdb

Garbage-collects and analyzes a database.

Synopsis

```
vacuumdb [connection-option...] [--full | -f] [-F] [--verbose | -v]
    [--analyze | -z] [--table | -t table [{ column [, ...] }] ] [dbname]
vacuumdb [connection-options...] [--all | -a] [--full | -f] [-F]
    [--verbose | -v] [--analyze | -z]
vacuumdb --help
vacuumdb --version
```

Description

vacuumdb is a utility for cleaning a PostgreSQL database. vacuumdb will also generate internal statistics used by the PostgreSQL query optimizer.

vacuumdb is a wrapper around the SQL command VACUUM. There is no effective difference between vacuuming databases via this utility and via other methods for accessing the server.

Options

-a | --all

Vacuums all databases.

[-d] dbname | [--dbname] dbname

The name of the database to vacuum. If this is not specified and --all is not used, the database name is read from the environment variable PGDATABASE. If that is not set, the user name specified for the connection is used.

-e | --echo

Echo the commands that reindexdb generates and sends to the server.

-f | --full

Selects a full vacuum, which may reclaim more space, but takes much longer and exclusively locks the table.
Warning: A VACUUM FULL is not recommended in Greenplum Database.

- F | --freeze
Freeze row transaction information.

- q | --quiet
Do not display a response.

- t table [(column)] | --table table [(column)]
Clean or analyze this table only. Column names may be specified only in conjunction
with the --analyze option. If you specify columns, you probably have to escape the
parentheses from the shell.

- v | --verbose
Print detailed information during processing.

- z | --analyze
Collect statistics for use by the query planner.

Connection Options
- h host | --host host
Specifies the host name of the machine on which the Greenplum master database server
is running. If not specified, reads from the environment variable PGHOST or defaults to
localhost.

- p port | --port port
Specifies the TCP port on which the Greenplum master database server is listening for
connections. If not specified, reads from the environment variable PGPORT or defaults to
5432.

- U username | --username username
The database role name to connect as. If not specified, reads from the environment
variable PGUSER or defaults to the current system user name.

- w | --no-password
Never issue a password prompt. If the server requires password authentication and a
password is not available by other means such as a .pgpass file, the connection attempt
will fail. This option can be useful in batch jobs and scripts where no user is present to
enter a password.

- W | --password
Force a password prompt.

Notes
vacuumdb might need to connect several times to the master server, asking for a password each time. It is
convenient to have a ~/.pgpass file in such cases.

Examples
To clean the database test:

vacuumdb test

To clean and analyze a database named bigdb:

clean and analyze bigdb
To clean a single table `foo` in a database named `mydb`, and analyze a single column `bar` of the table. Note the quotes around the table and column names to escape the parentheses from the shell:

```
vacuumdb --analyze --verbose --table 'foo(bar)' mydb
```

**See Also**

`VACUUM` and `ANALYZE` in the *Greenplum Database Reference Guide*
Oracle Compatibility Functions

Describes the Oracle Compatibility SQL functions in Greenplum Database. These functions target PostgreSQL.

Installing Oracle Compatibility Functions

Before using any Oracle Compatibility Functions, run the installation script `$GPHOME/share/postgresql/contrib/orafunc.sql` once for each database. For example, to install the functions in database `testdb`, use the following command:

```
$ psql -d testdb -f $GPHOME/share/postgresql/contrib/orafunc.sql
```

To uninstall Oracle Compatibility Functions, run the `uninstall_orafunc.sql` script:

```
$GPHOME/share/postgresql/contrib/uninstall_orafunc.sql
```

The following functions are available by default and do not require running the Oracle Compatibility installer:

- `sinh`
- `tanh`
- `cosh`
- `decode`

**Note:** The Oracle Compatibility Functions reside in the `oracompat` schema. To access them, prefix the schema name (`oracompat`) or alter the database search path to include the schema name. For example:

```
ALTER DATABASE db_name SET search_path = $user, public, oracompat;
```

If you alter the database search path, you must restart the database.

Oracle and Greenplum Implementation Differences

There are some differences in the implementation of these compatibility functions in the Greenplum Database from the Oracle implementation. If you use validation scripts, the output may not be exactly the same as in Oracle. Some of the differences are as follows:

- Oracle performs a decimal round off, Greenplum Database does not:
  - 2.00 becomes 2 in Oracle.
  - 2.0.0 remains 2.00 in Greenplum Database.
- The provided Oracle Compatibility functions handle implicit type conversions differently. For example, using the `decode` function:

  ```
  decode(expression, value, return [,value, return]... [, default])
  ```

  Oracle automatically converts `expression` and each `value` to the datatype of the first `value` before comparing. Oracle automatically converts `return` to the same datatype as the first result.

  The Greenplum implementation restricts `return` and `default` to be of the same data type. The `expression` and `value` can be different types if the data type of `value` can be converted into the data type...
of the expression. This is done implicitly. Otherwise, `decode` fails with an invalid input syntax error. For example:

```sql
SELECT decode('M',true,false);  
CASE  
  WHEN 1 THEN  
  f  
END  
(1 row)
```

```sql
ERROR: Invalid input syntax for integer:"M"
LINE 1: SELECT decode(1,'M',true,false);
```

- Numbers in `bigint` format are displayed in scientific notation in Oracle, but not in Greenplum Database:
  - `9223372036854775` displays as `9.2234E+15` in Oracle.
  - `9223372036854775` remains `9223372036854775` in Greenplum Database.
- The default date and timestamp format in Oracle is different than the default format in Greenplum Database. If the following code is executed:

```sql
CREATE TABLE TEST(date1 date, time1 timestamp, time2 timestamp with timezone);
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 Database returns the row, but Oracle does not return any rows.

**Note:** The correct syntax in Oracle to return the row is:

```sql
SELECT DECODE(to_char(date1, 'YYYY-MM-DD'), '2001-11-11', '2001-01-01') FROM TEST
```

**Oracle Compatibility Functions Reference**

The following are the Oracle Compatibility Functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add_months</code></td>
<td><code>nanvl</code></td>
<td><code>next_day</code></td>
</tr>
<tr>
<td><code>bitand</code></td>
<td><code>next_day (2)</code></td>
<td><code>nlsdate</code></td>
</tr>
<tr>
<td><code>concat</code></td>
<td><code>nvl</code></td>
<td><code>nvl2</code></td>
</tr>
<tr>
<td><code>cosh</code></td>
<td><code>oracle.substr</code></td>
<td><code>oracle.substr</code></td>
</tr>
<tr>
<td><code>decode</code></td>
<td><code>reverse</code></td>
<td><code>round</code></td>
</tr>
<tr>
<td><code>dump</code></td>
<td><code>sinh</code></td>
<td><code>sinh</code></td>
</tr>
<tr>
<td><code>instr</code></td>
<td><code>tanh</code></td>
<td><code>tanh</code></td>
</tr>
<tr>
<td><code>last_day</code></td>
<td><code>trunc</code></td>
<td></td>
</tr>
<tr>
<td><code>listagg</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>listagg (2)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>lnvnl</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>months_between</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**add_months**

Oracle-compliant function to add a given number of months to a given date.

**Synopsis**

```
add_months(date_expression, months_to_add)
```

**Description**

This Oracle-compatible function adds `months_to_add` to a `date_expression` and returns a `DATE`.

If the `date_expression` specifies the last day of the month, or if the resulting month has fewer days than the `date_expression`, then the returned value is the last day of the resulting month. Otherwise, the returned value has the same day of the month as the `date_expression`.

**Parameters**

- `date_expression`
  - The starting date. This can be any expression that can be implicitly converted to `DATE`.
- `months_to_add`
  - The number of months to add to the `date_expression`. This is an integer or any value that can be implicitly converted to an integer. This parameter can be positive or negative.

**Example**

```
SELECT name, phone, nextcalldate FROM clientdb
WHERE nextcalldate >= add_months(CURRENT_DATE, 6);
```

Returns `name`, `phone`, and `nextcalldate` for all records where `nextcalldate` is at least six months in the future.

**Compatibility**

This command is compatible with Oracle syntax and is provided for convenience.

**bitand**

Oracle-compliant function that computes a logical `AND` operation on the bits of two non-negative values.

**Synopsis**

```
bitand(expr1, expr2)
```

**Description**

This Oracle-compatible function returns an integer representing an `AND` operation on the bits of two non-negative values (`expr1` and `expr2`). 1 is returned when the values are the same. 0 is returned when the values are different. Only significant bits are compared. For example, an `AND` operation on the integers 5 (binary 101) and 1 (binary 001 or 1) compares only the rightmost bit, and results in a value of 1 (binary 1).

The types of `expr1` and `expr2` are `NUMBER`, and the result is of type `NUMBER`. If either argument is `NULL`, the result is `NULL`.

The arguments must be in the range `-(2^{(n-1)})` . . . `((2^{(n-1)})-1)`. If an argument is out of this range, the result is undefined.
Note:

- The current implementation of `BITAND` defines \( n = 128 \).
- PL/SQL supports an overload of `BITAND` for which the types of the inputs and of the result are all `BINARY_INTEGER` and for which \( n = 32 \).

**Parameters**

- **expr1**
  A non-negative integer expression.

- **expr2**
  A non-negative integer expression.

**Example**

```sql
SELECT bitand(expr1, expr2)
FROM ClientDB;
```

**Compatibility**

This command is compatible with Oracle syntax and is provided for convenience.

**concat**

Oracle-compliant function to concatenate two strings together.

**Synopsis**

```sql
concat (string1, string2)
```

**Description**

This Oracle-compatible function concatenates two strings (\( \text{string1} \) and \( \text{string2} \)) together.

The string returned is in the same character set as \( \text{string1} \). Its datatype depends on the datatypes of the arguments.

In concatenations of two different datatypes, the datatype returned is the one that results in a lossless conversion. Therefore, if one of the arguments is a \( \text{LOB} \), then the returned value is a \( \text{LOB} \). If one of the arguments is a national datatype, then the returned value is a national datatype. For example:

- `concat(CLOB, NCLOB)` returns NCLOB
- `concat(NCLOB, NCHAR)` returns NCLOB
- `concat(NCLOB, CHAR)` returns NCLOB
- `concat(NCHAR, CLOB)` returns NCLOB

This function is equivalent to the concatenation operator (\( || \)).

**Parameters**

- **string1/string2**
  The two strings to concatenate together.

  Both \( \text{string1} \) and \( \text{string2} \) can be any of the datatypes \( \text{CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB} \).
Example

```sql
SELECT concat(concat(last_name, ''s job category is ''),
    job_id)
FROM employees
```

Returns 'Smith's job category is 4B'

Compatibility

This command is compatible with Oracle syntax and is provided for convenience.

cosh

Oracle-compliant function to return the hyperbolic cosine of a given number.

Synopsis

cosh(float8)

Description

This Oracle-compatible function returns the hyperbolic cosine of the floating 8 input number (float8).

Note: This function is available by default and can be accessed without running the Oracle Compatibility installer.

Parameters

`float8`

The input number.

Example

```sql
SELECT cosh(0.2)
FROM ClientDB;
```

Returns '1.02006675561908' (hyperbolic cosine of 0.2)

Compatibility

This command is compatible with Oracle syntax and is provided for convenience.

decode

Oracle-compliant function to transform a data value to a specified return value. This function is a way to implement a set of CASE statements.

Note: decode is converted into a reserved word in Greenplum Database. If you want to use the Postgres two-argument decode function that decodes binary strings previously encoded to ASCII-only representation, you must invoke it by using the full schema-qualified syntax, `pg_catalog.decode()`, or by enclosing the function name in quotes "decode" ()

Note: Greenplum's implementation of this function transforms decode into case.

This results in the following type of output:

```sql
gptest=# select decode(a, 1, 'A', 2, 'B', 'C') from decodetest;
```
This also means that if you deparse your view with `decode`, you will see `case` expression instead.

You should use the `case` function instead of `decode`.

**Synopsis**

```
decode(expression, value, return [,value, return]... [, default])
```

**Description**

The Oracle-compatible function `decode` searches for a value in an expression. If the value is found, the function returns the specified value.

*Note:* This function is available by default and can be accessed without running the Oracle Compatibility installer.

**Parameters**

- **expression**
  The expression to search.
- **value**
  The value to find in the expression.
- **return**
  What to return if expression matches value.
- **default**
  What to return if expression does not match any of the values.

Only one `expression` is passed to the function. Multiple `value/return` pairs can be passed.

The `default` parameter is optional. If `default` is not specified and if `expression` does not match any of the passed `value` parameters, `decode` returns `null`. The Greenplum implementation restricts `return` and `default` to be of the same data type. The `expression` and `value` can be different types if the data type of `value` can be converted into the data type of the `expression`. This is done implicitly. Otherwise, `decode` fails with an invalid input syntax error.

**Examples**

In the following code, `decode` searches for a value for `company_id` and returns a specified value for that company. If `company_id` not one of the listed values, the default value `Other` is returned.

```
SELECT decode(company_id, 1, 'EMC',
             2, 'Greenplum',
             'Other')
FROM suppliers;
```

The following code using `CASE` statements to produce the same result as the example using `decode`.

```
SELECT CASE company_id
```

WHEN IS NOT DISTINCT FROM 1 THEN 'EMC'
WHEN IS NOT DISTINCT FROM 2 THEN 'Greenplum'
ELSE 'Other'
END
FROM suppliers;

Notes
To assign a range of values to a single return value, either pass an expression for each value in the range, or pass an expression that evaluates identically for all values in the range. For example, if a fiscal year begins on August 1, the quarters are shown in the following table.

Table 85: Months and Quarters for Fiscal Year Beginning on August 1

<table>
<thead>
<tr>
<th>Range (Alpha)</th>
<th>Range (Numeric)</th>
<th>Quarter</th>
</tr>
</thead>
<tbody>
<tr>
<td>August — October</td>
<td>8 — 10</td>
<td>Q1</td>
</tr>
<tr>
<td>November — January</td>
<td>11 — 1</td>
<td>Q2</td>
</tr>
<tr>
<td>February — April</td>
<td>2 — 4</td>
<td>Q3</td>
</tr>
<tr>
<td>May — July</td>
<td>5 — 7</td>
<td>Q4</td>
</tr>
</tbody>
</table>

The table contains a numeric field `curr_month` that holds the numeric value of a month, 1 – 12. There are two ways to use `decode` to get the quarter:

- **Method 1** - Include 12 values in the `decode` function:

  ```sql
  SELECT decode(curr_month, 1, 'Q2',
                2, 'Q3',
                3, 'Q3',
                4, 'Q3',
                5, 'Q4',
                6, 'Q4',
                7, 'Q4',
                8, 'Q1',
                9, 'Q1',
               10, 'Q1',
               11, 'Q2',
               12, 'Q2')
  FROM suppliers;
  ```

- **Method 2** - Use an expression that defines a unique value to decode:

  ```sql
  SELECT decode((1+MOD(curr_month+4,12)/3)::int, 1, 'Q1',
                2, 'Q2',
                3, 'Q3',
                4, 'Q4',
  FROM suppliers;
  ```

Compatibility
This command is compatible with Oracle syntax and is provided for convenience.

See Also
PostgreSQL `decode` (not compatible with Oracle)
**dump**

Oracle-compliant function that returns a text value that includes the datatype code, the length in bytes, and the internal representation of the expression.

**Synopsis**

\[ \text{dump(expression [,integer])} \]

**Description**

This Oracle-compatible function returns a text value that includes the datatype code, the length in bytes, and the internal representation of the expression.

**Parameters**

- **expression**
  Any expression
- **integer**
  The number of characters to return

**Example**

- `dump('Tech')` returns 'Typ=96 Len=4: 84,101,99,104'
- `dump ('tech')` returns 'Typ-96 Len=4: 84,101,99,104'
- `dump('Tech', 10)` returns 'Typ=96 Len=4: 84,101,99,104'
- `dump('Tech', 16)` returns 'Typ=96 Len=4: 54,65,63,68'
- `dump('Tech', 1016)` returns 'Typ=96 Len=4 CharacterSet=US7ASCII: 54,65,63,68'
- `dump('Tech', 1017)` returns 'Typ=96 Len=4 CharacterSet=US7ASCII: T,e,c,h'

**Compatibility**

This command is compatible with Oracle syntax and is provided for convenience.

**instr**

Oracle-compliant function to return the location of a substring in a string.

**Synopsis**

\[ \text{instr(string, substring, [position[,occurrence]])} \]

**Description**

This Oracle-compatible function searches for a `substring` in a `string`. If found, it returns an integer indicating the position of the `substring` in the `string`, if not found, the function returns 0.

Optionally you can specify that the search starts at a given `position` in the string, and only return the nth `occurrence` of the `substring` in the `string`. 
instr calculates strings using characters as defined by the input character set. The value returned is of NUMBER datatype.

**Parameters**

**string**
- The string to search.

**substring**
- The substring to search for in string.
- Both string and substring can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB.

**position**
- The position is a nonzero integer in string where the search will start. If not specified, this defaults to 1. If this value is negative, the function counts backwards from the end of string then searches towards to beginning from the resulting position.

**occurrence**
- Occurrence is an integer indicating which occurrence of the substring should be searched for. The value of occurrence must be positive.
- Both position and occurrence must be of datatype NUMBER, or any datatype that can be implicitly converted to NUMBER, and must resolve to an integer. The default values of both position and occurrence are 1, meaning that the search begins at the first character of string for the first occurrence of substring. The return value is relative to the beginning of string, regardless of the value of position, and is expressed in characters.

**Examples**

```sql
SELECT instr('Greenplum', 'e')
FROM ClientDB;
```

Returns 3; the first occurrence of 'e'

```sql
SELECT instr('Greenplum', 'e',1,2)
FROM ClientDB;
```

Returns 4; the second occurrence of 'e'

**Compatibility**

This command is compatible with Oracle syntax and is provided for convenience.

*last_day*

Oracle-compliant function to return the last day in a given month.

**Synopsis**

```sql
last_day(date_expression)
```

**Description**

This Oracle-compatible function returns the last day of the month specified by a date_expression. The return type is always DATE, regardless of the datatype of date_expression.
Parameters

date_expression

The date value used to calculate the last day of the month. This can be any expression that can be implicitly converted to DATE.

Example

```
SELECT name, hiredate, last_day(hiredate) "Option Date"
FROM employees;
```

Returns the name, hiredate, and last_day of the month of hiredate labeled "Option Date."

Compatibility

This command is compatible with Oracle syntax and is provided for convenience.

**listagg**

Oracle-compliant function that aggregates text values into a string.

**Note:** This function is an overloaded function. There are two Oracle-compliant listagg functions, one that takes one argument, the text to be aggregated (see below), and one that takes two arguments, the text to be aggregated and a delimiter (see next page).

**Synopsis**

```
listagg(text)
```

**Description**

This Oracle-compatible function aggregates text values into a string.

**Parameters**

text

The text value to be aggregated into a string.

**Example**

```
SELECT listagg(t) FROM (VALUES('abc'), ('def')) as l(t)
```

Returns: abcdef

**Compatibility**

This command is compatible with Oracle syntax and is provided for convenience.

**listagg (2)**

Oracle-compliant function that aggregates text values into a string, separating each by the separator specified in a second argument.

**Note:** This function is an overloaded function. There are two Oracle-compliant listagg functions, one that takes one argument, the text to be aggregated (see previous page), and one that takes two arguments, the text to be aggregated and a delimiter (see below).
Synopsis

listagg(text, separator)

Description
This Oracle-compatible function aggregates text values into a string, separating each by the separator specified in a second argument (separator).

Parameters
- **text**: The text value to be aggregated into a string.
- **separator**: The separator by which to delimit the text values.

Example

```
SELECT oracompat.listagg(t, '.') FROM (VALUES('abc'), ('def')) as l(t)
```

Returns: abc.def

Compatibility
This command is compatible with Oracle syntax and is provided for convenience.

innvl

Oracle-compliant function that returns **true** if the argument is false or NULL, or **false**.

Synopsis

innvl(condition)

Description
This Oracle-compatible function takes as an argument a condition and returns **true** if the condition is false or NULL and **false** if the condition is true.

Parameters
- **condition**: Any condition that evaluates to **true**, **false**, or **NULL**.

Example

```
SELECT innvl(true)
```

Returns: false

```
SELECT innvl(NULL)
```

Returns: true

```
SELECT innvl(false)
```

Returns: false
Returns: true

```sql
SELECT (3=5)
```

Returns: true

**Compatibility**

This command is compatible with Oracle syntax and is provided for convenience.

### months_between

Oracle-compliant function to evaluate the number of months between two given dates.

#### Synopsis

```sql
months_between(date_expression1, date_expression2)
```

#### Description

This Oracle-compatible function returns the number of months between `date_expression1` and `date_expression2`.

If `date_expression1` is later than `date_expression2`, then the result is positive.

If `date_expression1` is earlier than `date_expression2`, then the result is negative.

If `date_expression1` and `date_expression2` are either the same days of the month or both last days of months, then the result is always an integer. Otherwise the function calculates the fractional portion of the month based on a 31-day month.

#### Parameters

- `date_expression1`, `date_expression2`
  
  The date values used to calculate the number of months. This can be any expression that can be implicitly converted to `DATE`.

#### Examples

```sql
SELECT months_between
  (to_date ('2003/07/01', 'yyyy/mm/dd'),
   to_date ('2003/03/14', 'yyyy/mm/dd'));
```

Returns the number of months between July 1, 2003 and March 14, 2014.

```sql
SELECT * FROM employees
where months_between(hire_date, leave_date) <12;
```

Returns the number of months between `hire_date` and `leave_date`.

#### Compatibility

This command is compatible with Oracle syntax and is provided for convenience.

### nanvl

Oracle-compliant function to substitute a value for a floating point number when a non-number value is encountered.
Synopsis

nanvl(float1, float2)

Description

This Oracle-compatible function evaluates a floating point number (float1) such as BINARY_FLOAT or BINARY_DOUBLE. If it is a non-number (not a number, NaN), the function returns float2. This function is most commonly used to convert non-number values into either NULL or 0.

Parameters

float1

The BINARY_FLOAT or BINARY_NUMBER to evaluate.

float2

The value to return if float1 is not a number.

float1 and float2 can be any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. The function determines the argument with the highest numeric precedence, implicitly converts the remaining arguments to that datatype, and returns that datatype.

Example

SELECT nanvl(binary1, 0)
FROM MyDB;

Returns 0 if the binary1 field contained a non-number value. Otherwise, it would return the binary1 value.

Compatibility

This command is compatible with Oracle syntax and is provided for convenience.

next_day

Oracle-compliant function to return the date of the next specified weekday after a date.

This section describes using this function with a string argument; see the following page for details about using this function with an integer argument.

Note: This function is an overloaded function. There are two Oracle-compliant next_day functions, one that takes a date and a day of the week as its arguments (see below), and one that takes a date and an integer as its arguments (see next page).

Synopsis

next_day(date_expression, day_of_the_week)

Description

This Oracle-compatible function returns the first day_of_the_week (Tuesday, Wednesday, etc.) to occur after a date_expression.

The weekday must be specified in English.

The case of the weekday is irrelevant.

The return type is always DATE, regardless of the datatype of date_expression.
Parameters

**date_expression**

The starting date. This can be any expression that can be implicitly converted to DATE.

**day_of_the_week**

A string containing the name of a day, in English; for example 'Tuesday'. day_of_the_week is case-insensitive.

Example

```
SELECT name, next_day(hiredate,"MONDAY") "Second Week Start"
FROM employees;
```

Returns the name and the date of the next Monday after hiredate labeled "Second Week Start".

Compatibility

This command is compatible with Oracle syntax and is provided for convenience.

**next_day (2)**

Oracle-compliant function to add a given number of days to a date and returns the date of the following day.

*Note:* This function is an overloaded function. There are two Oracle next_day functions, one that takes a date and a day of the week as its arguments (see previous page), and one that takes a date and an integer as its arguments (see below).

Synopsis

```
next_day(date_expression, days_to_add)
```

Description

This Oracle-compatible function adds the number of days_to_add to a date_expression and returns the date of the day after the result.

The return type is always DATE, regardless of the datatype of date_expression.

Parameters

**date_expression**

The starting date. This can be any expression that can be implicitly converted to DATE.

**days_to_add**

The number of days to be add to the date_expression. This is an integer or any value that can be implicitly converted to an integer. This parameter can be positive or negative.

Example

```
SELECT name, next_day(hiredate,90) "Benefits Eligibility Date"
FROM EMPLOYEES;
```

Returns the name and the date that is 90 days after hiredate labeled "Benefits Eligibility Date".
Compatibility
This command is compatible with Oracle syntax and is provided for convenience.

**nlssort**
Oracle-compliant function that sorts data according to a specific collation.

**Synopsis**
```plaintext
nlssort (variable, collation)
```

**Description**
This Oracle-compatible function sorts data according to a specific collation.

**Parameters**
- **variable**
  - The data to sort.
- **collation**
  - The collation type by which to sort.

**Example**
```sql
CREATE TABLE test (name text);
INSERT INTO test VALUES('Anne'), ('anne'), ('Bob'), ('bob');
SELECT * FROM test ORDER BY nlssort(name, 'en_US.UTF-8');
  anne
  Anne
  bob
  Bob

SELECT * FROM test ORDER BY nlssort(name, 'C');
  Anne
  Bob
  anne
  bob
```

In the first example, the UTF-8 collation rules are specified. This groups characters together regardless of case.

In the second example, ASCII (C) collation is specified. This sorts according to ASCII order. The result is that upper case characters are sorted ahead of lower case ones.

**Compatibility**
This command is compatible with Oracle syntax and is provided for convenience.

**nvl**
Oracle-compliant function to substitute a specified value when an expression evaluates to `null`.

**Note:** This function is analogous to the PostgreSQL `coalesce` function.

**Synopsis**
```plaintext
nvl(expression_to_evaluate, null_replacement_value)
```
**Description**

This Oracle-compatible function evaluates `expression_to_evaluate`. If it is null, the function returns `null_replacement_value`; otherwise, it returns `expression_to_evaluate`.

**Parameters**

- `expression_to_evaluate`
  The expression to evaluate for a null value.

- `null_replacement_value`
  The value to return if `expression_to_evaluate` is null.

Both `expression_to_evaluate` and `null_replacement_value` must be the same data type.

**Examples**

```sql
SELECT nvl(contact_name,'None')
FROM clients;
SELECT nvl(amount_past_due,0)
FROM txns;
SELECT nvl(nickname, firstname)
FROM contacts;
```

**Compatibility**

This command is compatible with Oracle syntax and is provided for convenience.

**nvl2**

Oracle-compliant function that returns alternate values for both null and non-null values.

**Synopsis**

```sql
nvl2(expression_to_evaluate, non_null_replacement_value, null_replacement_value)
```

**Description**

This Oracle-compatible function evaluates `expression_to_evaluate`. If it is not null, the function returns `non_null_replacement_value`; otherwise, it returns `null_replacement_value`.

**Parameters**

- `expression_to_evaluate`
  The expression to evaluate for a null value.

- `non_null_replacement_value`
  The value to return if `expression_to_evaluate` is not null.

- `null_replacement_value`
  The value to return if `expression_to_evaluate` is null.

**Example**

```sql
select nvl2(unit_number,'Multi Unit','Single Unit')
from clients;
```
Compatibility

This command is compatible with Oracle syntax and is provided for convenience.

See Also

decode

**oracle.substr**

This Oracle-compliant function extracts a portion of a string.

**Synopsis**

```
oracle.substr(string, [start [,char_count]])
```

**Description**

This Oracle-compatible function extract a portion of a string.

- If `start` is 0, it is evaluated as 1.
- If `start` is negative, the starting position is negative, the starting position is `start` characters moving backwards from the end of string.
- If `char_count` is not passed to the function, all characters from start to the end of string are returned.
- If `char_count` is less than 1, null is returned.
- If `start` or `char_count` is a number, but not an integer, the values are resolved to integers.

**Parameters**

- **string**
  The string from which to extract.
- **start**
  An integer specifying the starting position in the string.
- **char_count**
  An integer specifying the number of characters to extract.

**Example**

```
oracle.substr(name,1,15)
```

Returns the first 15 characters of `name`.

```
oracle.substr("Greenplum",-4,4)
```

Returns "plum".

```
oracle.substr(name,2)
```

Returns all characters of `name`, beginning with the second character.

**Compatibility**

PostgreSQL `substr` (not compatible with Oracle)
**reverse**

Oracle-compliant function to return the input string in reverse order.

**Synopsis**

```
reverse (string)
```

**Description**

This Oracle-compatible function returns the input string (`string`) in reverse order.

**Parameters**

- **string**
  
The input string.

**Example**

```
SELECT reverse('gnirts')
FROM ClientDB;
```

Returns 'string'

**Compatibility**

This command is compatible with Oracle syntax and is provided for convenience.

**round**

Oracle-compliant function to round a date to a specific unit of measure (day, week, etc.).

---

**Note:** This function is an overloaded function. It shares the same name with the Postgres `round` mathematical function that rounds numeric input to the nearest integer or optionally to the nearest `x` number of decimal places.

---

**Synopsis**

```
round (date_time_expression, [unit_of_measure])
```

**Description**

This Oracle-compatible function rounds a `date_time_expression` to the nearest `unit_of_measure` (day, week, etc.). If a `unit_of_measure` is not specified, the `date_time_expression` is rounded to the nearest day.

It operates according to the rules of the Gregorian calendar.

If the `date_time_expression` datatype is `TIMESTAMP`, the value returned is always of datatype `TIMESTAMP`.

If the `date_time_expression` datatype is `DATE`, the value returned is always of datatype `DATE`.

**Parameters**

- **date_time_expression**
  
The date to round. This can be any expression that can be implicitly converted to `DATE` or `TIMESTAMP`.

- **unit_of_measure**
The unit of measure to apply for rounding. If not specified, then the \textit{date\_time\_expression} is rounded to the nearest day. Valid parameters are:

\begin{table}[h]
\centering
\begin{tabular}{|l|l|l|}
\hline
\textbf{Unit} & \textbf{Valid parameters} & \textbf{Rounding Rule} \\
\hline
Year & SYYYY, YYYY, YEAR, SYEAR, YYY, YY, Y & Rounds up on July 1st \\
ISO Year & IYYY, IY, I & \\
Quarter & Q & Rounds up on the 16th day of the second month of the quarter \\
Month & MONTH, MON, MM, RM & Rounds up on the 16th day of the month \\
Week & WW & Same day of the week as the first day of the year \\
IW & IW & Same day of the week as the first day of the ISO year \\
W & W & Same day of the week as the first day of the month \\
Day & DDD, DD, J & Rounds to the nearest day \\
Start day of the week & DAY, DY, D & Rounds to the nearest start (sunday) day of the week \\
Hour & HH, HH12, HH24 & Rounds to the next hour \\
Minute & MI & Rounds to the next minute \\
\hline
\end{tabular}
\end{table}

Example

\begin{verbatim}
SELECT round(TO_DATE('27-OCT-00','DD-MON-YY'), 'YEAR')
FROM ClientDB;
\end{verbatim}

Returns '01-JAN-01' (27 Oct 00 rounded to the first day of the following year (\textit{YEAR}))

\begin{verbatim}
SELECT round('startdate','Q')
FROM ClientDB;
\end{verbatim}

Returns '01-JUL-92' (the \textit{startdate} rounded to the first day of the quarter (\textit{Q}))

Compatibility

This command is compatible with Oracle syntax and is provided for convenience.

See Also

PostgreSQL \textit{round} (not compatible with Oracle)

\textit{sinh}

Oracle-compliant function to return the hyperbolic sine of a given number.
Synopsis

\texttt{sinh(float8)}

Description

This Oracle-compatible function returns the hyperbolic sine of the floating 8 input number \texttt{(float8)}.

\textbf{Note:} This function is available by default and can be accessed without running the Oracle Compatibility installer.

Parameters

\texttt{float8}

The input number.

Example

\begin{verbatim}
SELECT sinh(3) 
FROM ClientDB;
\end{verbatim}

Returns \texttt{'10.0178749274099'} (hyperbolic sine of 3)

Compatibility

This command is compatible with Oracle syntax and is provided for convenience.

\textbf{tanh}

Oracle-compliant function to return the hyperbolic tangent of a given number.

Synopsis

\texttt{tanh(float8)}

Description

This Oracle-compatible function returns the hyperbolic tangent of the floating 8 input number \texttt{(float8)}.

\textbf{Note:}

This function is available by default and can be accessed without running the Oracle Compatibility installer.

Parameters

\texttt{float8}

The input number.

Example

\begin{verbatim}
SELECT tanh(3) 
FROM ClientDB;
\end{verbatim}

Returns \texttt{'0.99505475368673'} (hyperbolic tangent of 3)
**Compatibility**
This command is compatible with Oracle syntax and is provided for convenience.

**trunc**
Oracle-compliant function to truncate a date to a specific unit of measure (day, week, hour, etc.).

*Note:*
This function is an overloaded function. It shares the same name with the Postgres `trunc` and the Oracle `trunc` mathematical functions. Both of these truncate numeric input to the nearest integer or optionally to the nearest x number of decimal places.

**Synopsis**
```
trunc(date_time_expression, [unit_of_measure])
```

**Description**
This Oracle-compatible function truncates a *date_time_expression* to the nearest *unit_of_measure* (day, week, etc.). If a *unit_of_measure* is not specified, the *date_time_expression* is truncated to the nearest day. It operates according to the rules of the Gregorian calendar.

If the *date_time_expression* datatype is `TIMESTAMP`, the value returned is always of datatype `TIMESTAMP`, truncated to the hour/min level.

If the *date_time_expression* datatype is `DATE`, the value returned is always of datatype `DATE`.

**Parameters**
- **date_time_expression**
  The date to truncate. This can be any expression that can be implicitly converted to `DATE` or `TIMESTAMP`.

- **unit_of_measure**
  The unit of measure to apply for truncating. If not specified, then *date_time_expression* is truncated to the nearest day. Valid formats are:

**Table 87: Valid Format Parameters**

<table>
<thead>
<tr>
<th>Unit</th>
<th>Valid parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>SYYYYY, YYYY, YEAR, SYEAR, YYY, YY, Y</td>
</tr>
<tr>
<td>ISO Year</td>
<td>IYYYY, IY, I</td>
</tr>
<tr>
<td>Quarter</td>
<td>Q</td>
</tr>
<tr>
<td>Month</td>
<td>MONTH, MON, MM, RM</td>
</tr>
<tr>
<td>Week</td>
<td>WW</td>
</tr>
<tr>
<td>IW</td>
<td>IW</td>
</tr>
<tr>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>Day</td>
<td>DDD, DD, J</td>
</tr>
<tr>
<td>Start day of the week</td>
<td>DAY, DY, D</td>
</tr>
<tr>
<td>Hour</td>
<td>HH, HH12, HH24</td>
</tr>
</tbody>
</table>
### Examples

```
SELECT TRUNC(TO_DATE('27-OCT-92','DD-MON-YY'), 'YEAR')
FROM ClientDB;
```

Returns '01-JAN-92' (27 Oct 92 truncated to the first day of the year (YEAR))

```
SELECT TRUNC(startdate,'Q')
FROM ClientDB;
```

Returns '1992-07-01' (the startdate truncated to the first day of the quarter (Q), depending on the date_style setting)

### Compatibility

This command is compatible with Oracle syntax and is provided for convenience.

### See Also

PostgreSQL `trunc` (not compatible with Oracle)
dblink Functions

The dblink module is provided for making easy connections to other databases either on the same database host, or on a remote host.

dblink is intended for database users to perform short ad hoc queries in other databases. dblink is not intended as a replacement for external tables or for administrative tools such as gptransfer.

The following procedure shows the basic steps for configuring and using dblink in Greenplum Database. See dblink in the PostgreSQL documentation for more information about individual functions.

1. Begin by creating a sample table to query using the dblink functions. These commands create a small table in the postgres database, which you will later query from the gpadmin database using dblink:

   ```bash
   $ psql -d postgres
   psql (8.3.23)
   Type "help" for help.
   postgres=# CREATE TABLE testdblink (a int, b text) DISTRIBUTED BY (a);
   CREATE TABLE
   postgres=# INSERT INTO testdblink VALUES (1, 'Cheese');
   INSERT 0 1
   postgres=# INSERT INTO testdblink VALUES (2, 'Fish');
   INSERT 0 1
   postgres=# \q
   $ 
   ```

2. Log into a different database (gpadmin in this example) and install the dblink functions if they are not already available. You install the dblink functions using the $GPHOME/share/postgresql/contrib/dblink.sql script:

   ```bash
   $ psql -d gpadmin
   psql (8.3.23)
   Type "help" for help.
   gpadmin=# \i /usr/local/gpdb/share/postgresql/contrib/dblink.sql
   SET
   CREATE FUNCTION
   CREATE FUNCTION
   CREATE FUNCTION
   CREATE FUNCTION
   REVOKE
   REVOKE
   CREATE FUNCTION
   CREATE FUNCTION
   CREATE FUNCTION...
   ```

3. Use the dblink_connect function to create both implicit and named connections to other databases. The connection string that you provide should be a libpq-style keyword/value string. For example, to create a named connection to the postgres database on the local Greenplum Database system:

   ```bash
   gpadmin=# SELECT dblink_connect('mylocalconn', 'dbname=postgres');
   dblink_connect
   ----------------
   OK
   (1 row)
   ```
To make a connection to a remote database system, simply include host and port information in the connection string. For example, to create an implicit `dblink` connection to a remote system:

```sql
SELECT dblink_connect('host=remotehost port=5432 dbname=postgres');
```

4. Use the basic `dblink` function to query a database using a configured connection. Keep in mind that the `dblink` function returns a record type, so you must assign the columns returned in the `dblink` query. For example, the following command uses the named connection to query the table you created in Step 1:

```sql
SELECT * FROM dblink('mylocalconn', 'SELECT * FROM testdblink') AS dbltab(id int, product text);
```

```
 id | product  
----+---------  
 1  | Cheese   
 2  | Fish     
(2 rows)
```

In this release of Greenplum Database, statements that modify table data cannot use named or implicit `dblink` connections. Instead, you must provide the connection string directly in the `dblink` function. For example:

```sql
CREATE TABLE testdbllocal (a int, b text) DISTRIBUTED BY (a);
```

```sql
INSERT INTO testdbllocal select * FROM dblink('dbname=postgres', 'SELECT * FROM testdblink') AS dbltab(id int, product text);
```

```
INSERT 0 2
```
hstore Functions

The hstore module implements a data type for storing sets of (key,value) pairs within a single Greenplum Database data field. This can be useful in various scenarios, such as rows with many attributes that are rarely examined, or semi-structured data.

In the current implementation, neither the key nor the value string can exceed 65535 bytes in length; an error will be thrown if this limit is exceeded. These maximum lengths may change in future releases.

Installing hstore

Before you can use hstore data type and functions, run the installation script $GPHOME/share/postgresql/contrib/hstore.sql in each database where you want the ability to query other databases:

$ psql -d testdb -f $GPHOME/share/postgresql/contrib/hstore.sql

hstore External Representation

The text representation of an hstore value includes zero or more key => value items, separated by commas. For example:

k => v
foo => bar, baz => whatever
"l-a" => "anything at all"

The order of the items is not considered significant (and may not be reproduced on output). Whitespace between items or around the => sign is ignored. Use double quotes if a key or value includes whitespace, comma, = or >. To include a double quote or a backslash in a key or value, precede it with another backslash. (Keep in mind that depending on the setting of standard_conforming_strings, you may need to double backslashes in SQL literal strings.)

A value (but not a key) can be a SQL NULL. This is represented as

key => NULL

The NULL keyword is not case-sensitive. Again, use double quotes if you want the string null to be treated as an ordinary data value.

Currently, double quotes are always used to surround key and value strings on output, even when this is not strictly necessary.

hstore Operators and Functions

Table 88: hstore Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>hstore -&gt; text</td>
<td>get value for key (null if not present)</td>
<td>'a=&gt;x, b=&gt;y':hstore -&gt; 'a'</td>
<td>x</td>
</tr>
<tr>
<td>text =&gt; text</td>
<td>make single-item hstore</td>
<td>'a' =&gt; 'b'</td>
<td>&quot;a&quot;=&gt;&quot;b&quot;</td>
</tr>
<tr>
<td>hstore</td>
<td></td>
<td>hstore</td>
<td>concatenation</td>
</tr>
</tbody>
</table>
Operator | Description | Example | Result
--- | --- | --- | ---
hstore ? text | does hstore contain key? | 'a=>'1':hstore ? 'a' | t
hstore @> hstore | does left operand contain right? | 'a=>b, b=>1, c=>>NULL':hstore @> 'b=>1' | t
hstore @@ hstore | is left operand contained in right? | 'a=>c':hstore @@ 'a=>b, b=>1, c=>>NULL' | f

**Note:** The => operator is deprecated and may be removed in a future release. Use the hstore(text, text) function instead.

### Table 89: hstore Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Description</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| hstore(text, text) | hstore | make single-item hstore | hstore('a', 'b') | "a"=>"b"
| akeys(hstore) | text[] | get hstore's keys as array | akeys('a=>1,b=>2') | {a,b}
| skeys(hstore) | setof text | get hstore's keys as set | skeys('a=>1,b=>2') | a
 b |
| avals(hstore) | text[] | get hstore's values as array | avals('a=>1,b=>2') | {1,2}
| svals(hstore) | setof text | get hstore's values as set | svals('a=>1,b=>2') | 1
 2 |
| each(hstore) | setof (key text, value text) | get hstore's keys and values as set | select * from each('a=>1,b=>2') | key | value
-----|-------
a | 1
 b | 2 |
| exist(hstore, text) | boolean | does hstore contain key? | exist('a=>1','a') | t
| defined(hstore, text) | boolean | does hstore contain non-null value for key? | defined('a=>>NULL','a') | f
| delete(hstore, text) | hstore | delete any item matching key | delete('a=>1,b=>>2','b') | "a"=>"1"

### Indexes

hstore has index support for @> and ? operators. You can use the GiST index type. For example:

```sql
CREATE INDEX hidx ON testhstore USING GIST(h);
```
Examples

Add a key, or update an existing key with a new value:

```
UPDATE tab SET h = h || ('c' => '3');
```

Delete a key:

```
UPDATE tab SET h = delete(h, 'kl');
```

Statistics

The `hstore` type, because of its intrinsic liberality, could contain a lot of different keys. Checking for valid keys is the task of the application. Examples below demonstrate several techniques for checking keys and obtaining statistics.

Simple example:

```
SELECT * FROM each('aaa=>bq, b=>NULL, ""=>1');
```

Using a table:

```
SELECT (each(h)).key, (each(h)).value INTO stat FROM testhstore;
```

Online statistics:

```
SELECT key, count(*) FROM
  (SELECT (each(h)).key FROM testhstore) AS stat
GROUP BY key
ORDER BY count DESC, key;
```

<table>
<thead>
<tr>
<th>key</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>line</td>
<td>883</td>
</tr>
<tr>
<td>query</td>
<td>207</td>
</tr>
<tr>
<td>pos</td>
<td>203</td>
</tr>
<tr>
<td>node</td>
<td>202</td>
</tr>
<tr>
<td>space</td>
<td>197</td>
</tr>
<tr>
<td>status</td>
<td>195</td>
</tr>
<tr>
<td>public</td>
<td>194</td>
</tr>
<tr>
<td>title</td>
<td>190</td>
</tr>
<tr>
<td>org</td>
<td>189</td>
</tr>
</tbody>
</table>

.................
Chapter 15

Greenplum Database Reference Guide

Reference information for Greenplum Database systems including SQL commands, system catalogs, environment variables, server configuration parameters, character set support, datatypes, and Greenplum Database extensions.
SQL Command Reference

The following SQL commands are available in Greenplum Database:

- ABORT
- ALTER AGGREGATE
- ALTER CONVERSION
- ALTER DATABASE
- ALTER DOMAIN
- ALTER EXTERNAL TABLE
- ALTER FILESPACE
- 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 TYPE
- ALTER USER
- ALTER USER MAPPING*
- ALTER VIEW
- ANALYZE
- BEGIN
- CHECKPOINT
- CLOSE
- CLUSTER
- COMMENT
- COMMIT
- COPY
- CREATE AGGREGATE
- CREATE CAST
- CREATE CONVERSION
- CREATE DATABASE
- CREATE DOMAIN
- 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 TYPE
- CREATE USER
- CREATE USER MAPPING
- CREATE VIEW
- DEALLOCATE
- DECLARE
- DELETE
- DISCARD
- DO
- DROP AGGREGATE
- DROP CAST
- DROP CONVERSION
- DROP DATABASE
- DROP DOMAIN
- DROP EXTERNAL TABLE
- DROP FILESPACE
- 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 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 ROLE
- SET SESSION AUTHORIZATION
- SET TRANSACTION
- SHOW
- START TRANSACTION
- TRUNCATE
- UPDATE
- VACUUM
- VALUES

* Not implemented in 5.0

**SQL Syntax Summary**

**ABORT**

Aborts the current transaction.

```
ABORT [WORK | TRANSACTION]
```

See `ABORT` for more information.
ALTER AGGREGATE
Changes the definition of an aggregate function

```
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 CONVERSION
Changes the definition of a conversion.

```
ALTER CONVERSION name RENAME TO newname
ALTER CONVERSION name OWNER TO newowner
```

See `ALTER CONVERSION` for more information.

ALTER DATABASE
Changes the attributes of a database.

```
ALTER DATABASE name [ WITH CONNECTION LIMIT connlimit ]
ALTER DATABASE name SET parameter { TO | = } { value | DEFAULT }
ALTER DATABASE name RESET parameter
ALTER DATABASE name RENAME TO newname
ALTER DATABASE name OWNER TO new_owner
```

See `ALTER DATABASE` for more information.

ALTER DOMAIN
Changes the definition of a domain.

```
ALTER DOMAIN name { SET DEFAULT expression | DROP DEFAULT }
ALTER DOMAIN name { SET | DROP } NOT NULL
ALTER DOMAIN name ADD domain_constraint
ALTER DOMAIN name DROP CONSTRAINT constraint_name [RESTRICT | CASCADE]
ALTER DOMAIN name OWNER TO new_owner
ALTER DOMAIN name SET SCHEMA new_schema
```

See `ALTER DOMAIN` for more information.

ALTER EXTENSION
Change the definition of an extension that is registered in a Greenplum database.

```
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

Changes the definition of an external table.

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 [USING expression]
OWNER TO new_owner

See ALTER EXTERNAL TABLE for more information.

ALTER FILESPACE

Changes the definition of a filespace.

ALTER FILESPACE name RENAME TO newname
ALTER FILESPACE name OWNER TO newowner

See ALTER FILESPACE for more information.

**ALTER FUNCTION**

Changes the definition of a function.

```
ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ]
    action [, ...] [RESTRICT]
) RENAME TO new_name

ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ]
    OWNER TO new_owner
) SET SCHEMA new_schema
```

See ALTER FUNCTION for more information.

**ALTER GROUP**

Changes a role name or membership.

```
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**

Changes the definition of an index.

```
ALTER INDEX name RENAME TO new_name

ALTER INDEX name SET TABLESPACE tablespace_name

ALTER INDEX name SET ( FILLFACTOR = value )

ALTER INDEX name RESET ( FILLFACTOR )
```

See ALTER INDEX for more information.

**ALTER LANGUAGE**

Changes the name of a procedural language.

```
ALTER LANGUAGE name RENAME TO newname

ALTER LANGUAGE name OWNER TO new_owner
```

See ALTER LANGUAGE for more information.
**ALTER OPERATOR**
Changes the definition of an operator.

```
ALTER OPERATOR name ( {lefttype | NONE} , {righttype | NONE} )
OWNER TO newowner
```

See [ALTER OPERATOR](#) for more information.

**ALTER OPERATOR CLASS**
Changes the definition of an operator class.

```
ALTER OPERATOR CLASS name USING index_method RENAME TO newname
```

```
ALTER OPERATOR CLASS name USING index_method OWNER TO newowner
```

See [ALTER OPERATOR CLASS](#) for more information.

**ALTER OPERATOR FAMILY**
Changes the definition of an operator family.

```
ALTER OPERATOR FAMILY name USING index_method ADD
{  OPERATOR strategy_number operator_name ( op_type, op_type ) [ RECHECK ]
| 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 RENEAME TO newname
```

```
ALTER OPERATOR FAMILY name USING index_method OWNER TO newowner
```

See [ALTER OPERATOR FAMILY](#) for more information.

**ALTER PROTOCOL**
Changes the definition of a protocol.

```
ALTER PROTOCOL name RENAME TO newname
```

```
ALTER PROTOCOL name OWNER TO newowner
```

See [ALTER PROTOCOL](#) for more information.

**ALTER RESOURCE GROUP**
Changes the limits of a resource group.

```
ALTER RESOURCE GROUP name SET group_attribute value
```

See [ALTER RESOURCE GROUP](#) for more information.
**ALTER RESOURCE QUEUE**
Changes the limits of a resource queue.

```
ALTER RESOURCE QUEUE name WITH ( queue_attribute=value [, ... ] )
```

See **ALTER RESOURCE QUEUE** for more information.

**ALTER ROLE**
Changes a database role (user or group).

```
ALTER ROLE name RENAME TO newname

ALTER ROLE name SET config_parameter {TO | =} {value | DEFAULT}

ALTER ROLE name RESET config_parameter

ALTER ROLE name RESOURCE QUEUE {queue_name | NONE}

ALTER ROLE name RESOURCE GROUP {group_name | NONE}

ALTER ROLE name [ [WITH] option [ ... ] ]
```

See **ALTER ROLE** for more information.

**ALTER SCHEMA**
Changes the definition of a schema.

```
ALTER SCHEMA name RENAME TO newname

ALTER SCHEMA name OWNER TO newowner
```

See **ALTER SCHEMA** for more information.

**ALTER SEQUENCE**
Changes the definition of a sequence generator.

```
ALTER SEQUENCE name [INCREMENT [ BY ] increment]
[MINVALUE minvalue | NO MINVALUE]
[MAXVALUE maxvalue | NO MAXVALUE]
[RESTART [ WITH ] start]
[CACHE cache] [[ NO ] CYCLE]
[OWNED BY {table.column | NONE}]

ALTER SEQUENCE name RENAME TO new_name

ALTER SEQUENCE name SET SCHEMA new_schema
```

See **ALTER SEQUENCE** for more information.

**ALTER TABLE**
Changes the definition of a table.

```
ALTER TABLE [ONLY] name RENAME [COLUMN] column TO new_column

ALTER TABLE name RENAME TO new_name
```
ALTER TABLE name SET SCHEMA new_schema

ALTER TABLE [ONLY] name SET DISTRIBUTED BY (column, ...) | DISTRIBUTED RANDOMLY | WITH (REORGANIZE=true|false)

ALTER TABLE [ONLY] name action [, ... ]

ALTER TABLE name [ ALTER PARTITION { partition_name | FOR (RANK(number)) | FOR (value) } partition_action [...] ]

See ALTER TABLE for more information.

ALTER TABLESPACE
Changes the definition of a tablespace.

ALTER TABLESPACE name RENAME TO newname
ALTER TABLESPACE name OWNER TO newowner

See ALTER TABLESPACE for more information.

ALTER TYPE
Changes the definition of a data type.

ALTER TYPE name OWNER TO new_owner | SET SCHEMA new_schema

See ALTER TYPE for more information.

ALTER USER
Changes the definition of a database role (user).

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 VIEW
Changes the definition of a view.

ALTER VIEW name RENAME TO newname

See ALTER VIEW for more information.
**ANALYZE**

Collects statistics about a database.

```
ANALYZE [VERBOSE] [ROOTPARTITION [ALL] ]
 [table [ (column [, ...] ) ]]
```

See `ANALYZE` for more information.

**BEGIN**

Starts a transaction block.

```
BEGIN [WORK | TRANSACTION] [transaction_mode]
 [READ ONLY | READ WRITE]
```

See `BEGIN` for more information.

**CHECKPOINT**

Forces a transaction log checkpoint.

```
CHECKPOINT
```

See `CHECKPOINT` for more information.

**CLOSE**

Closes a cursor.

```
CLOSE cursor_name
```

See `CLOSE` for more information.

**CLUSTER**

Physically reorders a heap storage table on disk according to an index. Not a recommended operation in Greenplum Database.

```
CLUSTER indexname ON tablename
CLUSTER tablename
```

See `CLUSTER` for more information.

**COMMENT**

Defines or change the comment of an object.

```
COMMENT ON
 { TABLE object_name |
   COLUMN table_name.column_name |
   AGGREGATE agg_name (agg_type [, ...]) |
   CAST (sourcetype AS targettype) |
   CONSTRAINT constraint_name ON table_name |
   CONVERSION object_name |
   DATABASE object_name |
   DOMAIN object_name |
```
COPY table [(column [, ...])] FROM '{file' | PROGRAM 'command' | STDIN} [ [WITH] [ON SEGMENT] [BINARY] [OIDS] [HEADER] [DELIMITER [ AS ] 'delimiter'] [NULL [ AS ] 'null string'] [ESCAPE [ AS ] 'escape' | 'OFF'] [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF'] [CSV [QUOTE [ AS ] 'quote'] [FORCE NOT NULL column [, ...]] [FILL MISSING FIELDS] [LOG ERRORS] SEGMENT REJECT LIMIT count [ROWS | PERCENT] ]

COPY {table [(column [, ...])] | (query)} TO '{file' | PROGRAM 'command' | STDOUT} [ [WITH] [ON SEGMENT] [BINARY] [OIDS] [HEADER] [DELIMITER [ AS ] 'delimiter'] [NULL [ AS ] 'null string'] [ESCAPE [ AS ] 'escape' | 'OFF'] [CSV [QUOTE [ AS ] 'quote'] [FORCE QUOTE column [, ...]] [IGNORE EXTERNAL PARTITIONS] ]
See `COPY` for more information.

**CREATE AGGREGATE**
Defines a new aggregate function.

```sql
CREATE [ORDERED] AGGREGATE name (input_data_type [ , ... ])
( SFUNC = sfunc,
  STYPE = state_data_type
  [, PREFUNC = prefunc]
  [, FINALFUNC = ffunc]
  [, INITCOND = initial_condition]
  [, SORTOP = sort_operator] )
```

See `CREATE AGGREGATE` for more information.

**CREATE CAST**
Defines a new cast.

```sql
CREATE CAST (sourcetype AS targettype)
WITH FUNCTION funcname (argtypes)
[AS ASSIGNMENT | AS IMPLICIT]
CREATE CAST (sourcetype AS targettype) WITHOUT FUNCTION
[AS ASSIGNMENT | AS IMPLICIT]
```

See `CREATE CAST` for more information.

**CREATE CONVERSION**
Defines a new encoding conversion.

```sql
CREATE [DEFAULT] CONVERSION name FOR source_encoding TO
  dest_encoding FROM funcname
```

See `CREATE CONVERSION` for more information.

**CREATE DATABASE**
Creates a new database.

```sql
CREATE DATABASE name [ [WITH] [OWNER [=] dbowner]
[TEMPLATE [=] template]
[ENCODING [=] encoding]
[TABLESPACE [=] tablespace]
[CONNECTION LIMIT [=] conlimit ] ]
```

See `CREATE DATABASE` for more information.

**CREATE DOMAIN**
Defines a new domain.

```sql
CREATE DOMAIN name [AS] data_type [DEFAULT expression]
  [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
Defines a new external table.

CREATE [READABLE] EXTERNAL TABLE table_name
( column_name data_type [, ...] | LIKE other_table )
LOCATION ('file://seghost[ :port ]/path/file' [, ...])
| ('gpfs://filehost[ :port ]/file_pattern[ #transform = trans_name ]'
  [, ...])
| ('gpfs://filehost[ :port ]/file_pattern[ #transform = trans_name ]'
  [, ...])
| ('gpfs://hdfs_host[ :port ]/path/file')
| ('pxf://path-to-data?PROFILE[ &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] )]
| 'AVRO'
| 'PARQUET'
| 'CUSTOM' (Formatter=<formatter_specifications>)
[ ENCODING 'encoding' ]
[ [ LOG ERRORS ] SEGMENT REJECT LIMIT count
  [ROWS | PERCENT ] ]

CREATE [READABLE] EXTERNAL WEB TABLE table_name
( 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']

CREATE WRITABLE EXTERNAL 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]'
[, ...])
| ('gphdfs://hdfs_host[:port]/path')
FORMATT 'TEXT'
[ ENCODING 'encoding' ]
[ DISTRIBUTED BY (column, [ ... ]) | DISTRIBUTED RANDOMLY ]

CREATE WRITABLE EXTERNAL 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]
FORMATT 'TEXT'
[ ENCODING 'write_encoding' ]
[ DISTRIBUTED BY (column, [ ... ]) | DISTRIBUTED RANDOMLY ]

CREATE WRITABLE EXTERNAL WEB TABLE table_name
(column_name data_type [, ...] | LIKE other_table)
EXECUTE 'command' [ON ALL]
FORMATT 'TEXT'
[ ENCODING 'write_encoding' ]
[ DISTRIBUTED BY (column, [ ... ]) | DISTRIBUTED RANDOMLY ]

| '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] ))
| '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] ))
| '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 ] ]
[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, [ ... ] ) | DISTRIBUTED RANDOMLY ]

See CREATE EXTERNAL TABLE for more information.

CREATE FUNCTION
Defines a new function.

CREATE [OR REPLACE] FUNCTION name
( [ [argmode] [argname] argtype [ { DEFAULT | = } defexpr ] [, ... ] ] )
| TABLE [{ argname argtype | LIKE other table }
| [, ...]]}

{ LANGUAGE langname
| IMMUTABLE | STABLE | VOLATILE
| CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
| [EXTERNAL] SECURITY INVOKE | [EXTERNAL] SECURITY DEFINE
| COST execution_cost
| SET configuration_parameter { TO value | = value | FROM CURRENT }
| AS 'definition'
| AS 'obj_file', 'link_symbol' } ...
[ WITH ({ DESCRIPT = describe_function }
| [, ...] ) ]

See CREATE FUNCTION for more information.

CREATE GROUP
Defines a new database role.

CREATE GROUP name [[WITH] option [ ... ]]

See CREATE GROUP for more information.

CREATE INDEX
Defines a new index.

CREATE [UNIQUE] INDEX name ON table
[USING btree|bitmap|gist]
( { column | (expression) | opclass } [, ... ]
| WITH ( FILLFACTOR = value )
| TABLESPACE tablespace]
[WHERE predicate]

See CREATE INDEX for more information.
**CREATE LANGUAGE**
Defines a new procedural language.

```
CREATE [PROCEDURAL] LANGUAGE name

CREATE [TRUSTED] [PROCEDURAL] LANGUAGE name
    HANDLER call_handler [ INLINE inline_handler ]
    [VALIDATOR valfunction]
```

See *CREATE LANGUAGE* for more information.

**CREATE OPERATOR**
Defines a new operator.

```
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**
Defines a new operator class.

```
CREATE OPERATOR CLASS name [DEFAULT] FOR TYPE data_type
    USING index_method AS
    
    {  
    OPERATOR strategy_number op_name [(op_type, op_type)] [RECHECK]  
    | FUNCTION support_number funcname (argument_type [, ...] )  
    | STORAGE storage_type
    } [, ... ]
```

See *CREATE OPERATOR CLASS* for more information.

**CREATE OPERATOR FAMILY**
Defines a new operator family.

```
CREATE OPERATOR FAMILY name USING index_method
```

See *CREATE OPERATOR FAMILY* for more information.

**CREATE PROTOCOL**
Registers a custom data access protocol that can be specified when defining a Greenplum Database external table.

```
CREATE [TRUSTED] PROTOCOL name (  
    [readfunc='read_call_handler'] [, writefunc='write_call_handler']  
    [, validatorfunc='validate_handler' ])
```

See *CREATE PROTOCOL* for more information.
CREATE RESOURCE GROUP
Defines a new resource group.

```sql
CREATE RESOURCE GROUP name WITH (group_attribute=value [, ... ])
```

See `CREATE RESOURCE GROUP` for more information.

CREATE RESOURCE QUEUE
Defines a new resource queue.

```sql
CREATE RESOURCE QUEUE name WITH (queue_attribute=value [, ... ])
```

See `CREATE RESOURCE QUEUE` for more information.

CREATE ROLE
Defines a new database role (user or group).

```sql
CREATE ROLE name [[WITH] option [ ... ]]
```

See `CREATE ROLE` for more information.

CREATE RULE
Defines a new rewrite rule.

```sql
CREATE [OR REPLACE] RULE name AS ON event
TO table [WHERE condition]
DO [ALSO | INSTEAD] { NOTHING | command | (command; command ...
... ) }
```

See `CREATE RULE` for more information.

CREATE SCHEMA
Defines a new schema.

```sql
CREATE SCHEMA schema_name [AUTHORIZATION username]
[schema_element [ ... ]]
CREATE SCHEMA AUTHORIZATION rolename [schema_element [ ... ]]
```

See `CREATE SCHEMA` for more information.

CREATE SEQUENCE
Defines a new sequence generator.

```sql
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 TABLE

Defines a new table.

```
CREATE [[GLOBAL | LOCAL] {TEMPORARY | TEMP}] TABLE table_name (  
    ( column_name data_type [ DEFAULT default_expr ]  
    [column_constraint [ ... ] ]  
    [ ENCODING ( storage_directive [, ... ] ) ]  
)  
| table_constraint  
| LIKE other_table {{INCLUDING | EXCLUDING}  
    [DEFAULTS | CONSTRAINTS]} ...}  
[, ... ] )  
| INHERITS ( parent_table [, ... ] ) ]  
| WITH ( storage_parameter=value [, ... ] )  
| ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]  
| TABLESPACE tablespace ]  
| DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]  
| 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  
    [(...)  
    ) ]  
) ]  
)
```

See CREATE TABLE for more information.

CREATE TABLE AS

Defines a new table from the results of a query.

```
CREATE [[GLOBAL | LOCAL] {TEMPORARY | TEMP}] TABLE table_name (  
    ( column_name [, ... ] )  
)  
| WITH ( storage_parameter=value [, ... ] )  
| ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]  
| TABLESPACE tablespace ]  
| DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]
```

See CREATE TABLE AS for more information.

CREATE TABLESPACE

Defines a new tablespace.

```
CREATE TABLESPACE tablespace_name [OWNER username]  
    FILESPACE filespace_name
```

See CREATE TABLESPACE for more information.

CREATE TYPE

Defines a new data type.

```
CREATE TYPE name AS ( attribute_name data_type [, ... ] )
```
CREATE TYPE name AS ENUM ('label' [, ... ])


CREATE TYPE name

See CREATE TYPE for more information.

CREATE USER
Defines a new database role with the LOGIN privilege by default.

CREATE USER name [[WITH] option [ ... ]]

See CREATE USER for more information.

CREATE VIEW
Defines a new view.

CREATE [OR REPLACE] [TEMP | TEMPORARY] VIEW name [ ( column_name [, ... ] ) ] AS query

See CREATE VIEW for more information.

DEALLOCATE
Deallocates a prepared statement.

DEALLOCATE [PREPARE] name

See DEALLOCATE for more information.

DECLARE
Defines a cursor.

DECLARE name [BINARY] [INSENSITIVE] [NO SCROLL] CURSOR [[WITH | WITHOUT] HOLD] FOR query [FOR READ ONLY]

See DECLARE for more information.
DELETE
Deletes rows from a table.

```
DELETE FROM [ONLY] table [[AS] alias]
    [USING usinglist]
    [WHERE condition | WHERE CURRENT OF cursor_name ]
```

See `DELETE` for more information.

DISCARD
Discards the session state.

```
DISCARD { ALL | PLANS | TEMPORARY | TEMP }
```

See `DISCARD` for more information.

DROP AGGREGATE
Removes an aggregate function.

```
DROP AGGREGATE [IF EXISTS] name (type [, ...] ) [CASCADE | RESTRICT]
```

See `DROP AGGREGATE` for more information.

DO
Executes an anonymous code block as a transient anonymous function.

```
DO [ LANGUAGE lang_name ] code
```

See `DO` for more information.

DROP CAST
Removes a cast.

```
DROP CAST [IF EXISTS] (sourcetype AS targettype) [CASCADE | RESTRICT]
```

See `DROP CAST` for more information.

DROP CONVERSION
Removes a conversion.

```
DROP CONVERSION [IF EXISTS] name [CASCADE | RESTRICT]
```

See `DROP CONVERSION` for more information.

DROP DATABASE
Removes a database.

```
DROP DATABASE [IF EXISTS] name
```

See `DROP DATABASE` for more information.
**DROP DOMAIN**
Removes a domain.

```
DROP DOMAIN [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See *DROP DOMAIN* for more information.

**DROP EXTENSION**
Removes an extension from a Greenplum database.

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

See *DROP EXTENSION* for more information.

**DROP EXTERNAL TABLE**
Removes an external table definition.

```
DROP EXTERNAL [WEB] TABLE [IF EXISTS] name [CASCADE | RESTRICT]
```

See *DROP EXTERNAL TABLE* for more information.

**DROP FILESPACE**
Removes a filespace.

```
DROP FILESPACE [IF EXISTS] filespacename
```

See *DROP FILESPACE* for more information.

**DROP FUNCTION**
Removes a function.

```
DROP FUNCTION [IF EXISTS] name ( [ [argmode] [argname] argtype [, ...] ] ) [CASCADE | RESTRICT]
```

See *DROP FUNCTION* for more information.

**DROP GROUP**
Removes a database role.

```
DROP GROUP [IF EXISTS] name [, ...]
```

See *DROP GROUP* for more information.

**DROP INDEX**
Removes an index.

```
DROP INDEX [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See *DROP INDEX* for more information.
**DROP LANGUAGE**
Removes a procedural language.

```sql
DROP [PROCEDURAL] LANGUAGE [IF EXISTS] name [CASCADE | RESTRICT]
```

See *DROP LANGUAGE* for more information.

**DROP OPERATOR**
Removes an operator.

```sql
DROP OPERATOR [IF EXISTS] name ( {lefttype | NONE} ,
   {righttype | NONE} ) [CASCADE | RESTRICT]
```

See *DROP OPERATOR* for more information.

**DROP OPERATOR CLASS**
Removes an operator class.

```sql
DROP OPERATOR CLASS [IF EXISTS] name USING index_method [CASCADE | RESTRICT]
```

See *DROP OPERATOR CLASS* for more information.

**DROP OPERATOR FAMILY**
Removes an operator family.

```sql
DROP OPERATOR FAMILY [IF EXISTS] name USING index_method [CASCADE | RESTRICT]
```

See *DROP OPERATOR FAMILY* for more information.

**DROP OWNED**
Removes database objects owned by a database role.

```sql
DROP OWNED BY name [, ...] [CASCADE | RESTRICT]
```

See *DROP OWNED* for more information.

**DROP PROTOCOL**
Removes a external table data access protocol from a database.

```sql
DROP PROTOCOL [IF EXISTS] name
```

See *DROP PROTOCOL* for more information.

**DROP RESOURCE GROUP**
Removes a resource group.

```sql
DROP RESOURCE GROUP group_name
```

See *DROP RESOURCE GROUP* for more information.
**DROP RESOURCE QUEUE**
Removes a resource queue.

```
DROP RESOURCE QUEUE queue_name
```

See `DROP RESOURCE QUEUE` for more information.

**DROP ROLE**
Removes a database role.

```
DROP ROLE [IF EXISTS] name [, ...]
```

See `DROP ROLE` for more information.

**DROP RULE**
Removes a rewrite rule.

```
DROP RULE [IF EXISTS] name ON relation [CASCADE | RESTRICT]
```

See `DROP RULE` for more information.

**DROP SCHEMA**
Removes a schema.

```
DROP SCHEMA [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See `DROP SCHEMA` for more information.

**DROP SEQUENCE**
Removes a sequence.

```
DROP SEQUENCE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See `DROP SEQUENCE` for more information.

**DROP TABLE**
Removes a table.

```
DROP TABLE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See `DROP TABLE` for more information.

**DROP TABLESPACE**
Removes a tablespace.

```
DROP TABLESPACE [IF EXISTS] tablesapce_name
```

See `DROP TABLESPACE` for more information.
DROP TYPE
Removes a data type.

```
DROP TYPE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See *DROP TYPE* for more information.

DROP USER
Removes a database role.

```
DROP USER [IF EXISTS] name [, ...]
```

See *DROP USER* for more information.

DROP VIEW
Removes a view.

```
DROP VIEW [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See *DROP VIEW* for more information.

END
Commits the current transaction.

```
END [WORK | TRANSACTION]
```

See *END* for more information.

EXECUTE
Executes a prepared SQL statement.

```
EXECUTE name [ (parameter [, ...] ) ]
```

See *EXECUTE* for more information.

EXPLAIN
Shows the query plan of a statement.

```
EXPLAIN [ANALYZE] [VERBOSE] statement
```

See *EXPLAIN* for more information.

FETCH
Retrieves rows from a query using a cursor.

```
FETCH [ forward_direction { FROM | IN } ] cursorname
```

See *FETCH* for more information.
**GRANT**

Defines access privileges.

```
GRANT { {SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRIGGER | TRUNCATE } [, ...] | ALL [PRIVILEGES] } 
  ON [TABLE] tablename [, ...] 
  TO { rolename | PUBLIC} [, ...] [WITH GRANT OPTION]
```

```
GRANT { {USAGE | SELECT | UPDATE} [, ...] | ALL [PRIVILEGES] } 
  ON SEQUENCE sequencename [, ...] 
  TO { rolename | PUBLIC } [, ...] [WITH GRANT OPTION]
```

```
GRANT { {CREATE | CONNECT | TEMPORARY | TEMP} [, ...] | ALL [PRIVILEGES] } 
  ON DATABASE dbname [, ...] 
  TO { rolename | PUBLIC} [, ...] [WITH GRANT OPTION]
```

```
GRANT { EXECUTE | ALL [PRIVILEGES] } 
  ON FUNCTION funcname ( [ [argmode] [argname] argtype [, ...] ] [, ...] 
  TO { rolename | PUBLIC} [, ...] [WITH GRANT OPTION]
```

```
GRANT { USAGE | ALL [PRIVILEGES] } 
  ON LANGUAGE langname [, ...] 
  TO { rolename | PUBLIC} [, ...] [WITH GRANT OPTION]
```

```
GRANT { {CREATE | USAGE} [, ...] | ALL [PRIVILEGES] } 
  ON SCHEMA schemaname [, ...] 
  TO { rolename | PUBLIC} [, ...] [WITH GRANT OPTION]
```

```
GRANT { CREATE | ALL [PRIVILEGES] } 
  ON TABLESPACE tablespacename [, ...] 
  TO { rolename | 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**

Creates new rows in a table.

```
INSERT INTO table ( [ column [, ...] ] ) 
{ DEFAULT VALUES | VALUES ( {expression | DEFAULT} [, ...] ) 
 [, ...] | query}
```

See **INSERT** for more information.

**LOAD**

Loads or reloads a shared library file.

```
LOAD 'filename'
```

See **LOAD** for more information.
**LOCK**
Locks a table.

```
LOCK [TABLE] name [, ...] [IN lockmode MODE] [NOWAIT]
```

See `LOCK` for more information.

**MOVE**
Positions a cursor.

```
MOVE [ forward_direction {FROM | IN} ] cursorname
```

See `MOVE` for more information.

**PREPARE**
Prepare a statement for execution.

```
PREPARE name [ (datatype [, ...] ) ] AS statement
```

See `PREPARE` for more information.

**REASSIGN OWNED**
Changes the ownership of database objects owned by a database role.

```
REASSIGN OWNED BY old_role [, ...] TO new_role
```

See `REASSIGN OWNED` for more information.

**REINDEX**
Rebuilds indexes.

```
REINDEX {INDEX | TABLE | DATABASE | SYSTEM} name
```

See `REINDEX` for more information.

**RELEASE SAVEPOINT**
Destroys a previously defined savepoint.

```
RELEASE [SAVEPOINT] savepoint_name
```

See `RELEASE SAVEPOINT` for more information.

**RESET**
Restores the value of a system configuration parameter to the default value.

```
RESET configuration_parameter
RESET ALL
```

See `RESET` for more information.
**REVOKE**

Removes access privileges.

```sql
REVOKE [GRANT OPTION FOR] { {SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRIGGER | TRUNCATE } [, ...] | ALL [PRIVILEGES] } ON [TABLE] tablename [, ...]
FROM { rolename | PUBLIC} [, ...]
[CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { {USAGE | SELECT | UPDATE} [, ...] | ALL [PRIVILEGES] } ON SEQUENCE sequencename [, ...]
FROM { rolename | PUBLIC} [, ...]
[CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { {CREATE | CONNECT | TEMPORARY | TEMP} [, ...] | ALL [PRIVILEGES] } ON DATABASE dbname [, ...]
FROM { rolename | PUBLIC} [, ...]
[CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] {EXECUTE | ALL [PRIVILEGES]} ON FUNCTION funcname ( [[argmode] argname argtype [, ...]] ) [, ...]
FROM { rolename | PUBLIC} [, ...]
[CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] {USAGE | ALL [PRIVILEGES]} ON LANGUAGE langname [, ...]
FROM { rolename | PUBLIC} [, ...]
[CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { {CREATE | USAGE} [, ...] | ALL [PRIVILEGES] } ON SCHEMA schemaname [, ...]
FROM { rolename | PUBLIC} [, ...]
[CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { CREATE | ALL [PRIVILEGES] } ON TABLESPACE tablespacename [, ...]
FROM { rolename | PUBLIC} [, ...]
[CASCADE | RESTRICT]

REVOKE [ADMIN OPTION FOR] parent_role [, ...]
FROM member_role [, ...]
[CASCADE | RESTRICT]
```

See `REVOKE` for more information.

**ROLLBACK**

Aborts the current transaction.

```
ROLLBACK [WORK | TRANSACTION]
```

See `ROLLBACK` for more information.
ROLLBACK TO SAVEPOINT
Rolls back the current transaction to a savepoint.

```
ROLLBACK [WORK | TRANSACTION] TO [SAVEPOINT] savepoint_name
```

See [ROLLBACK TO SAVEPOINT](#) for more information.

SAVEPOINT
Defines a new savepoint within the current transaction.

```
SAVEPOINT savepoint_name
```

See [SAVEPOINT](#) for more information.

SELECT
Retrieves rows from a table or view.

```
[ WITH [ RECURSIVE\(^1\) ] 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_specification)]
    [{UNION | INTERSECT | EXCEPT} [ALL] select]
    [ORDER BY expression [ASC | DESC | USING operator] [NULLS {FIRST | LAST}]
    [, ...]]
    [LIMIT {count | ALL}]
    [OFFSET start]
    [FOR {UPDATE | SHARE} [OF table_name [, ...]] [NOWAIT] [...]]
```

See [SELECT](#) for more information.

SELECT INTO
Defines a new table from the results of a query.

```
[ WITH [ RECURSIVE\(^1\) ] with_query [, ...] ]
SELECT [ALL | DISTINCT [ON (expression [, ...])]]
    * | expression [AS output_name] [, ...]
    INTO [TEMPORARY | TEMP] [TABLE] new_table
    [FROM from_item [, ...]]
    [WHERE condition]
    [GROUP BY expression [, ...]]
    [HAVING condition [, ...]]
    [{UNION | INTERSECT | EXCEPT} [ALL] select]
    [ORDER BY expression [ASC | DESC | USING operator] [NULLS {FIRST | LAST}]
    [, ...]]
    [LIMIT {count | ALL}]
    [OFFSET start]
    [FOR {UPDATE | SHARE} [OF table_name [, ...]] [NOWAIT] [...]]
```

See [SELECT INTO](#) for more information.
**SET**

Changes the value of a Greenplum Database configuration parameter.

```sql
SET [SESSION | LOCAL] configuration_parameter {TO | =} value | 'value' | DEFAULT
SET [SESSION | LOCAL] TIME ZONE {timezone | LOCAL | DEFAULT}
```

See **SET** for more information.

**SET ROLE**

Sets the current role identifier of the current session.

```sql
SET [SESSION | LOCAL] ROLE rolename
SET [SESSION | LOCAL] ROLE NONE
RESET ROLE
```

See **SET ROLE** for more information.

**SET SESSION AUTHORIZATION**

Sets the session role identifier and the current role identifier of the current session.

```sql
SET [SESSION | LOCAL] SESSION AUTHORIZATION rolename
SET [SESSION | LOCAL] SESSION AUTHORIZATION DEFAULT
RESET SESSION AUTHORIZATION
```

See **SET SESSION AUTHORIZATION** for more information.

**SET TRANSACTION**

Sets the characteristics of the current transaction.

```sql
SET TRANSACTION [transaction_mode] [READ ONLY | READ WRITE]
SET SESSION CHARACTERISTICS AS TRANSACTION transaction_mode
[READ ONLY | READ WRITE]
```

See **SET TRANSACTION** for more information.

**SHOW**

Shows the value of a system configuration parameter.

```sql
SHOW configuration_parameter
SHOW ALL
```

See **SHOW** for more information.
**START TRANSACTION**

Starts a transaction block.

```
START TRANSACTION [SERIALIZABLE | READ COMMITTED | READ UNCOMMITTED]
[READ WRITE | READ ONLY]
```

See `START TRANSACTION` for more information.

**TRUNCATE**

Empties a table of all rows.

```
TRUNCATE [TABLE] name [, ...] [CASCADE | RESTRICT]
```

See `TRUNCATE` for more information.

**UPDATE**

Updates rows of a table.

```
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**

Garbage-collects and optionally analyzes a database.

```
VACUUM [FULL] [FREEZE] [VERBOSE] [table]
VACUUM [FULL] [FREEZE] [VERBOSE] ANALYZE
    [table [(column [, ... ])]]
```

See `VACUUM` for more information.

**VALUES**

Computes a set of rows.

```
VALUES ( expression [, ...] ) [, ...]
    [ORDER BY sort_expression [ASC | DESC | USING operator] [, ...]]
    [LIMIT {count | ALL}] [OFFSET start]
```

See `VALUES` for more information.

**ABORT**

Aborts the current transaction.

**Synopsis**

```
ABORT [WORK | TRANSACTION]
```
Description

`ABORT` rolls back the current transaction and causes all the updates made by the transaction to be discarded. This command is identical in behavior to the standard SQL command `ROLLBACK`, and is present only for historical reasons.

Parameters

`WORK`  
`TRANSACTION`  
Optional key words. They have no effect.

Notes

Use `COMMIT` to successfully terminate a transaction.

Issuing `ABORT` when not inside a transaction does no harm, but it will provoke a warning message.

Compatibility

This command is a Greenplum Database extension present for historical reasons. `ROLLBACK` is the equivalent standard SQL command.

See Also

`BEGIN`, `COMMIT`, `ROLLBACK`

**ALTER AGGREGATE**

Changes the definition of an aggregate function

Synopsis

```
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` changes the definition of an aggregate function.

You must own the aggregate function to use `ALTER AGGREGATE`. To change the schema of an aggregate function, you must also have `CREATE` privilege on the new schema. To alter the owner, you must also be a direct or indirect member of the new owning role, and that role must have `CREATE` privilege on the aggregate function's schema. (These restrictions enforce that altering the owner does not do anything you could not do by dropping and recreating the aggregate function. However, a superuser can alter ownership of any aggregate function anyway.)

Parameters

`name`  
The name (optionally schema-qualified) of an existing aggregate function.

`type`  
An input data type on which the aggregate function operates. To reference a zero-argument aggregate function, write `*` in place of the list of input data types.

`new_name`
The new name of the aggregate function.

`new_owner`

The new owner of the aggregate function.

`new_schema`

The new schema for the aggregate function.

**Examples**

To rename the aggregate function `myavg` for type `integer` to `my_average`:

```
ALTER AGGREGATE myavg(integer) RENAME TO my_average;
```

To change the owner of the aggregate function `myavg` for type `integer` to `joe`:

```
ALTER AGGREGATE myavg(integer) OWNER TO joe;
```

To move the aggregate function `myavg` for type `integer` into schema `myschema`:

```
ALTER AGGREGATE myavg(integer) SET SCHEMA myschema;
```

**Compatibility**

There is no `ALTER AGGREGATE` statement in the SQL standard.

**See Also**

`CREATE AGGREGATE`, `DROP AGGREGATE`

**ALTER CONVERSION**

Changes the definition of a conversion.

**Synopsis**

```
ALTER CONVERSION name RENAME TO newname
ALTER CONVERSION name OWNER TO newowner
```

**Description**

**ALTER CONVERSION** changes the definition of a conversion.

You must own the conversion to use `ALTER CONVERSION`. To alter the owner, you must also be a direct or indirect member of the new owning role, and that role must have `CREATE` privilege on the conversion's schema. (These restrictions enforce that altering the owner does not do anything you could not do by dropping and recreating the conversion. However, a superuser can alter ownership of any conversion anyway.)

**Parameters**

`name`

The name (optionally schema-qualified) of an existing conversion.

`newname`

The new name of the conversion.

`newowner`
The new owner of the conversion.

Examples

To rename the conversion `iso_8859_1_to_utf8` to `latin1_to_unicode`:

```sql
ALTER CONVERSION iso_8859_1_to_utf8 RENAME TO latin1_to_unicode;
```

To change the owner of the conversion `iso_8859_1_to_utf8` to `joe`:

```sql
ALTER CONVERSION iso_8859_1_to_utf8 OWNER TO joe;
```

Compatibility

There is no `ALTER CONVERSION` statement in the SQL standard.

See Also

`CREATE CONVERSION`, `DROP CONVERSION`

**ALTER DATABASE**

Changes the attributes of a database.

Synopsis

```sql
ALTER DATABASE name [ WITH CONNECTION LIMIT connlimit ]
ALTER DATABASE name SET parameter { TO | = } { value | DEFAULT }
ALTER DATABASE name RESET parameter
ALTER DATABASE name RENAME TO newname
ALTER DATABASE name OWNER TO new_owner
```

Description

`ALTER DATABASE` changes the attributes of a database.

The first form changes the allowed connection limit for a database. Only the database owner or a superuser can change this setting.

The second and third forms change the session default for a configuration parameter for a Greenplum database. Whenever a new session is subsequently started in that database, the specified value becomes the session default value. The database-specific default overrides whatever setting is present in the server configuration file (`postgresql.conf`). Only the database owner or a superuser can change the session defaults for a database. Certain parameters cannot be set this way, or can only be set by a superuser.

The fourth form changes the name of the database. Only the database owner or a superuser can rename a database; non-superuser owners must also have the `CREATEDB` privilege. You cannot rename the current database. Connect to a different database first.

The fifth form changes the owner of the database. To alter the owner, you must own the database and also be a direct or indirect member of the new owning role, and you must have the `CREATEDB` privilege. (Note that superusers have all these privileges automatically.)
Parameters

name
The name of the database whose attributes are to be altered.

con limit
The maximum number of concurrent connections possible. The default of -1 means there is no limitation.

parameter value
Set this database's session default for the specified configuration parameter to the given value. If value is DEFAULT or, equivalently, RESET is used, the database-specific setting is removed, so the system-wide default setting will be inherited in new sessions. Use RESET ALL to clear all database-specific settings. See Server Configuration Parameters for information about server parameters. for information about all user-settable configuration parameters.

newname
The new name of the database.

new_owner
The new owner of the database.

Notes
It is also possible to set a configuration parameter session default for a specific role (user) rather than to a database. Role-specific settings override database-specific ones if there is a conflict. See ALTER ROLE.

Examples
To set the default schema search path for the mydatabase database:

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

Compatibility
The ALTER DATABASE statement is a Greenplum Database extension.

See Also
CREATE DATABASE, DROP DATABASE, SET

ALTER DOMAIN
Changes the definition of a domain.

Synopsis

```
ALTER DOMAIN name { SET DEFAULT expression | DROP DEFAULT }

ALTER DOMAIN name { SET | DROP } NOT NULL

ALTER DOMAIN name ADD domain_constraint

ALTER DOMAIN name DROP CONSTRAINT constraint_name [RESTRICT | CASCADE]

ALTER DOMAIN name OWNER TO new_owner
```
Description

ALTER DOMAIN changes the definition of an existing domain. There are several sub-forms:

- **SET/DROP DEFAULT** — These forms set or remove the default value for a domain. Note that defaults only apply to subsequent INSERT commands. They do not affect rows already in a table using the domain.
- **SET/DROP NOT NULL** — These forms change whether a domain is marked to allow NULL values or to reject NULL values. You may only SET NOT NULL when the columns using the domain contain no null values.
- **ADD domain_constraint** — This form adds a new constraint to a domain using the same syntax as CREATE DOMAIN. This will only succeed if all columns using the domain satisfy the new constraint.
- **DROP CONSTRAINT** — This form drops constraints on a domain.
- **OWNER** — This form changes the owner of the domain to the specified user.
- **SET SCHEMA** — This form changes the schema of the domain. Any constraints associated with the domain are moved into the new schema as well.

You must own the domain to use ALTER DOMAIN. To change the schema of a domain, you must also have CREATE privilege on the new schema. To alter the owner, you must also be a direct or indirect member of the new owning role, and that role must have CREATE privilege on the domain's schema. (These restrictions enforce that altering the owner does not do anything you could not do by dropping and recreating the domain. However, a superuser can alter ownership of any domain anyway.)

Parameters

**name**

The name (optionally schema-qualified) of an existing domain to alter.

**domain_constraint**

New domain constraint for the domain.

**constraint_name**

Name of an existing constraint to drop.

**CASCADE**

Automatically drop objects that depend on the constraint.

**RESTRICT**

Refuse to drop the constraint if there are any dependent objects. This is the default behavior.

**new_owner**

The user name of the new owner of the domain.

**new_schema**

The new schema for the domain.

Examples

To add a NOT NULL constraint to a domain:

```
ALTER DOMAIN zipcode SET NOT NULL;
```

To remove a NOT NULL constraint from a domain:

```
ALTER DOMAIN zipcode DROP NOT NULL;
```
To add a check constraint to a domain:

```
ALTER DOMAIN zipcode ADD CONSTRAINT zipchk CHECK (char_length(VALUE) = 5);
```

To remove a check constraint from a domain:

```
ALTER DOMAIN zipcode DROP CONSTRAINT zipchk;
```

To move the domain into a different schema:

```
ALTER DOMAIN zipcode SET SCHEMA customers;
```

### Compatibility

`ALTER DOMAIN` conforms to the SQL standard, except for the `OWNER` and `SET SCHEMA` variants, which are Greenplum Database extensions.

### See Also

`CREATE DOMAIN`, `DROP DOMAIN`

### ALTER EXTENSION

**Synopsis**

```
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 [ , ... ]

Description

ALTER EXTENSION changes the definition of an installed extension. These are the subforms:

UPDATE

This form updates the extension to a newer version. The extension must supply a suitable update script (or series of scripts) that can modify the currently-installed version into the requested version.

SET SCHEMA

This form moves the extension member objects into another schema. The extension must be relocatable.

ADD member_object

This form adds an existing object to the extension. This is useful in extension update scripts. The added object is treated as a member of the extension. The object can only be dropped by dropping the extension.

DROP member_object

This form removes a member object from the extension. This is mainly useful in extension update scripts. The object is not dropped, only disassociated from the extension.

See Packaging Related Objects into an Extension for more information about these operations.

You must own the extension to use ALTER EXTENSION. The ADD and DROP forms also require ownership of the object that is being added or dropped.

Parameters

name

The name of an installed extension.

new_version

The new version of the extension. The new_version can be either an identifier or a string literal. If not specified, the command attempts to update to the default version in the extension control file.

new_schema

The new schema for the extension.

object_name
aggregate_name
function_name
operator_name

The name of an object to be added to or removed from the extension. Names of tables, aggregates, domains, foreign tables, functions, operators, operator classes, operator families, sequences, text search objects, types, and views can be schema-qualified.

source_type

The name of the source data type of the cast.
**target_type**
The name of the target data type of the cast.

**argmode**
The mode of a function or aggregate argument: IN, OUT, INOUT, or VARIADIC. The default is IN.
The command ignores the OUT arguments. Only the input arguments are required to determine the function identity. It is sufficient to list the IN, INOUT, and VARIADIC arguments.

**argname**
The name of a function or aggregate argument.
The command ignores argument names, since only the argument data types are required to determine the function identity.

**argtype**
The data type of a function or aggregate argument.

**left_type**
**right_type**
The data types (optionally schema-qualified) of the operator arguments. Specify NONE for the missing argument of a prefix or postfix operator.

**PROCEDURAL**
This is a noise word.

**type_name**
The name of the data type of the transform.

**lang_name**
The name of the language of the transform.

### Examples
To update the hstore extension to version 2.0:

```
ALTER EXTENSION hstore UPDATE TO '2.0';
```

To change the schema of the hstore extension to utils:

```
ALTER EXTENSION hstore SET SCHEMA utils;
```

To add an existing function to the hstore extension:

```
ALTER EXTENSION hstore ADD FUNCTION populate_record(anyelement, hstore);
```

### Compatibility
ALTER EXTENSION is a Greenplum Database extension.

### See Also
CREATE EXTENSION, DROP EXTENSION
**ALTER EXTERNAL TABLE**

Changes the definition of an external table.

**Synopsis**

```
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 [USING expression]`
- `OWNER TO new_owner`

**Description**

`ALTER EXTERNAL TABLE` changes the definition of an existing external table. These are the supported `ALTER EXTERNAL TABLE` actions:

- **ADD COLUMN** — Adds a new column to the external table definition.
- **DROP COLUMN** — Drops a column from the external table definition. If you drop readable external table columns, it only changes the table definition in Greenplum Database. The `CASCADE` keyword is required if anything outside the table depends on the column, such as a view that references the column.
- **ALTER COLUMN TYPE** — Changes the data type of a table column.
- **OWNER** — Changes the owner of the external table to the specified user.

Use the `ALTER TABLE` command to perform these actions on an external table.

- Set (change) the table schema.
- Rename the table.
- Rename a table column.

You must own the external table to use `ALTER EXTERNAL TABLE`. To change the schema of an external table, you must also have `CREATE` privilege on the new schema. To alter the owner, you must also be a direct or indirect member of the new owning role, and that role must have `CREATE` privilege on the external table’s schema. A superuser has these privileges automatically.

Changes to the external table definition with either `ALTER EXTERNAL TABLE` or `ALTER TABLE` do not affect the external data.

The `ALTER EXTERNAL TABLE` and `ALTER TABLE` commands cannot modify the type external table (read, write, web), the table `FORMAT` information, or the location of the external data. To modify this information, you must drop and recreate the external table definition.

**Parameters**

*name*

The name (possibly schema-qualified) of an existing external table definition to alter.

*column*

Name of an existing column.

*new_column*

Name of a new column.

*USING expression*
Optional if an implicit or assignment cast exists from the old column data type to new data type. The clause is required if there is no implicit or assignment cast.

The **USING** clause does not affect the external data.

**type**

Data type of the new column, or new data type for an existing column.

**new_owner**

The role name of the new owner of the external table.

**CASCADE**

Automatically drop objects that depend on the dropped column, such as a view that references the column.

**RESTRICT**

Refuse to drop the column or constraint if there are any dependent objects. This is the default behavior.

**Examples**

Add a new column to an external table definition:

```
ALTER EXTERNAL TABLE ext_expenses ADD COLUMN manager text;
```

Change the owner of an external table:

```
ALTER EXTERNAL TABLE ext_data OWNER TO jojo;
```

Change the data type of an external table:

```
ALTER EXTERNAL TABLE ext_leads ALTER COLUMN acct_code TYPE integer USING acct_code::integer
```

When altering the column data type from `text` to `integer`, the **USING** clause is required but does not affect the external data.

**Compatibility**

`ALTER EXTERNAL TABLE` is a Greenplum Database extension. There is no `ALTER EXTERNAL TABLE` statement in the SQL standard or regular PostgreSQL.

**See Also**

`CREATE EXTERNAL TABLE`, `DROP EXTERNAL TABLE`, `ALTER TABLE`
You must own the filespace to use `ALTER FILESPACE`. To alter the owner, you must also be a direct or indirect member of the new owning role (note that superusers have these privileges automatically).

**Parameters**

`name`

The name of an existing filespace.

`newname`

The new name of the filespace. The new name cannot begin with `pg_` or `gp_` (reserved for system filespaces).

`newowner`

The new owner of the filespace.

**Examples**

Rename filespace `myfs` to `fast_ssd`:

```
ALTER FILESPACE myfs RENAME TO fast_ssd;
```

Change the owner of tablespace `myfs`:

```
ALTER FILESPACE myfs OWNER TO dba;
```

**Compatibility**

There is no `ALTER FILESPACE` statement in the SQL standard or in PostgreSQL.

**See Also**

`DROP FILESPACE`, `gpfilespace` in the *Greenplum Database Utility Guide*

**ALTER FUNCTION**

Changes the definition of a function.

**Synopsis**

```
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}
{[EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER}
COST execution_cost
SET configuration_parameter { TO | = } { value | DEFAULT }
SET configuration_parameter FROM CURRENT
Description

ALTER FUNCTION changes the definition of a function.

You must own the function to use ALTER FUNCTION. To change a function’s schema, you must also have CREATE privilege on the new schema. To alter the owner, you must also be a direct or indirect member of the new owning role, and that role must have CREATE privilege on the function's schema. (These restrictions enforce that altering the owner does not do anything you could not do by dropping and recreating the function. However, a superuser can alter ownership of any function anyway.)

Parameters

name

The name (optionally schema-qualified) of an existing function.

argmode

The mode of an argument: either IN, OUT, INOUT, or VARIADIC. If omitted, the default is IN. Note that ALTER FUNCTION does not actually pay any attention to OUT arguments, since only the input arguments are needed to determine the function's identity. So it is sufficient to list the IN, INOUT, and VARIADIC arguments.

argname

The name of an argument. Note that ALTER FUNCTION does not actually pay any attention to argument names, since only the argument data types are needed to determine the function's identity.

argtype

The data type(s) of the function's arguments (optionally schema-qualified), if any.

new_name

The new name of the function.

new_owner

The new owner of the function. Note that if the function is marked SECURITY DEFINER, it will subsequently execute as the new owner.

new_schema

The new schema for the function.

CALLED ON NULL INPUT

RETURNS NULL ON NULL INPUT

STRICT

CALLED ON NULL INPUT changes the function so that it will be invoked when some or all of its arguments are null. RETURNS NULL ON NULL INPUT or STRICT changes the function so that it is not invoked if any of its arguments are null; instead, a null result is assumed automatically. See CREATE FUNCTION for more information.

IMMUTABLE

STABLE

VOLATILE

Change the volatility of the function to the specified setting. See CREATE FUNCTION for details.

[ EXTERNAL ] SECURITY INVOKER

[ EXTERNAL ] SECURITY DEFINER
Change whether the function is a security definer or not. The key word `EXTERNAL` is ignored for SQL conformance. See `CREATE FUNCTION` for more information about this capability.

**COST** `execution_cost`

Change the estimated execution cost of the function. See `CREATE FUNCTION` for more information.

**configuration_parameter** `value`

Set or change the value of a configuration parameter when the function is called. If `value` is `DEFAULT` or, equivalently, `RESET` is used, the function-local setting is removed, and the function executes with the value present in its environment. Use `RESET ALL` to clear all function-local settings. `SET FROM CURRENT` applies the session's current value of the parameter when the function is entered.

**RESTRICT**

Ignored for conformance with the SQL standard.

**Notes**

Greenplum Database has limitations on the use of functions defined as `STABLE` or `VOLATILE`. See `CREATE FUNCTION` for more information.

**Examples**

To rename the function `sqrt` for type `integer` to `square_root`:

```
ALTER FUNCTION sqrt(integer) RENAME TO square_root;
```

To change the owner of the function `sqrt` for type `integer` to `joe`:

```
ALTER FUNCTION sqrt(integer) OWNER TO joe;
```

To change the `schema` of the function `sqrt` for type `integer` to `math`:

```
ALTER FUNCTION sqrt(integer) SET SCHEMA math;
```

To adjust the search path that is automatically set for a function:

```
ALTER FUNCTION check_password(text) RESET search_path;
```

**Compatibility**

This statement is partially compatible with the `ALTER FUNCTION` statement in the SQL standard. The standard allows more properties of a function to be modified, but does not provide the ability to rename a function, make a function a security definer, or change the owner, schema, or volatility of a function. The standard also requires the `RESTRICT` key word, which is optional in Greenplum Database.

**See Also**

`CREATE FUNCTION`, `DROP FUNCTION`
Synopsis

ALTER GROUP groupname ADD USER username [, ... ]
ALTER GROUP groupname DROP USER username [, ... ]
ALTER GROUP groupname RENAME TO newname

Description

ALTER GROUP changes the attributes of a user group. This is an obsolete command, though still accepted for backwards compatibility, because users and groups are superseded by the more general concept of roles. See ALTER ROLE for more information.

The first two variants add users to a group or remove them from a group. Any role can play the part of groupname or username. The preferred method for accomplishing these tasks is to use GRANT and REVOKE.

Parameters

groupname
  The name of the group (role) to modify.
username
  Users (roles) that are to be added to or removed from the group. The users (roles) must already exist.
newname
  The new name of the group (role).

Examples

To add users to a group:

ALTER GROUP staff ADD USER karl, john;

To remove a user from a group:

ALTER GROUP workers DROP USER beth;

Compatibility

There is no ALTER GROUP statement in the SQL standard.

See Also

ALTER ROLE, GRANT, REVOKE

ALTER INDEX

Changes the definition of an index.

Synopsis

ALTER INDEX name RENAME TO new_name
ALTER INDEX name SET TABLESPACE tablespace_name
ALTER INDEX name SET ( FILLFACTOR = value )
ALTER INDEX name RESET ( FILLFACTOR )

Description
ALTER INDEX changes the definition of an existing index. There are several subforms:

- **RENAME** — Changes the name of the index. There is no effect on the stored data.
- **SET TABLESPACE** — Changes the index's tablespace to the specified tablespace and moves the data file(s) associated with the index to the new tablespace. See also `CREATE TABLESPACE`.
- **SET FILLFACTOR** — Changes the index-method-specific storage parameters for the index. The built-in index methods all accept a single parameter: `FILLFACTOR`. The fillfactor for an index is a percentage that determines how full the index method will try to pack index pages. Index contents will not be modified immediately by this command. Use `REINDEX` to rebuild the index to get the desired effects.
- **RESET FILLFACTOR** — Resets `FILLFACTOR` to the default. As with `SET`, a `REINDEX` may be needed to update the index entirely.

Parameters

- **name**
  The name (optionally schema-qualified) of an existing index to alter.

- **new_name**
  New name for the index.

- **tablespace_name**
  The tablespace to which the index will be moved.

- **FILLFACTOR**
  The fillfactor for an index is a percentage that determines how full the index method will try to pack index pages. For B-trees, leaf pages are filled to this percentage during initial index build, and also when extending the index at the right (largest key values). If pages subsequently become completely full, they will be split, leading to gradual degradation in the index's efficiency.

  B-trees use a default fillfactor of 90, but any value from 10 to 100 can be selected. If the table is static then fillfactor 100 is best to minimize the index's physical size, but for heavily updated tables a smaller fillfactor is better to minimize the need for page splits. The other index methods use fillfactor in different but roughly analogous ways; the default fillfactor varies between methods.

Notes

These operations are also possible using `ALTER TABLE`.

Changing any part of a system catalog index is not permitted.

Examples

To rename an existing index:

```
ALTER INDEX distributors RENAME TO suppliers;
```

To move an index to a different tablespace:

```
ALTER INDEX distributors SET TABLESPACE fasttablespace;
```

To change an index's fill factor (assuming that the index method supports it):

```
ALTER INDEX distributors SET (fillfactor = 75);
```
REINDEX INDEX distributors;

Compatibility
ALTER INDEX is a Greenplum Database extension.

See Also
CREATE INDEX, REINDEX, ALTER TABLE

ALTER LANGUAGE
Changes the name of a procedural language.

Synopsis
```
ALTER LANGUAGE name RENAME TO newname
ALTER LANGUAGE name OWNER TO new_owner
```

Description
ALTER LANGUAGE changes the definition of a procedural language for a specific database. Definition changes supported include renaming the language or assigning a new owner. You must be superuser or the owner of the language to use ALTER LANGUAGE.

Parameters
- **name**
  Name of a language.
- **newname**
  The new name of the language.
- **new_owner**
  The new owner of the language.

Compatibility
There is no ALTER LANGUAGE statement in the SQL standard.

See Also
CREATE LANGUAGE, DROP LANGUAGE

ALTER OPERATOR
Changes the definition of an operator.

Synopsis
```
ALTER OPERATOR name ( {lefttype | NONE} , {righttype | NONE} )
OWNER TO newowner
```

Description
ALTER OPERATOR changes the definition of an operator. The only currently available functionality is to change the owner of the operator.
You must own the operator to use `ALTER OPERATOR`. To alter the owner, you must also be a direct or indirect member of the new owning role, and that role must have `CREATE` privilege on the operator’s schema. (These restrictions enforce that altering the owner does not do anything you could not do by dropping and recreating the operator. However, a superuser can alter ownership of any operator anyway.)

**Parameters**

- **name**
  The name (optionally schema-qualified) of an existing operator.

- **lefttype**
  The data type of the operator’s left operand; write `NONE` if the operator has no left operand.

- **righttype**
  The data type of the operator’s right operand; write `NONE` if the operator has no right operand.

- **newowner**
  The new owner of the operator.

**Examples**

Change the owner of a custom operator `a @@ b` for type `text`:

```
ALTER OPERATOR @@ (text, text) OWNER TO joe;
```

**Compatibility**

There is no `ALTEROPERATOR` statement in the SQL standard.

**See Also**

`CREATE OPERATOR`, `DROP OPERATOR`

## ALTER OPERATOR CLASS

Changes the definition of an operator class.

**Synopsis**

```
ALTER OPERATOR CLASS name USING index_method RENAME TO newname

ALTER OPERATOR CLASS name USING index_method OWNER TO newowner
```

**Description**

`ALTER OPERATOR CLASS` changes the definition of an operator class.

You must own the operator class to use `ALTER OPERATOR CLASS`. To alter the owner, you must also be a direct or indirect member of the new owning role, and that role must have `CREATE` privilege on the operator class’s schema. (These restrictions enforce that altering the owner does not do anything you could not do by dropping and recreating the operator class. However, a superuser can alter ownership of any operator class anyway.)

**Parameters**

- **name**
  The name (optionally schema-qualified) of an existing operator class.
**index_method**

The name of the index method this operator class is for.

**newname**

The new name of the operator class.

**newowner**

The new owner of the operator class.

**Compatibility**

There is no ALTER OPERATOR CLASS statement in the SQL standard.

**See Also**

CREATE OPERATOR CLASS, DROP OPERATOR CLASS

**ALTER OPERATOR FAMILY**

Changes the definition of an operator family.

**Synopsis**

```sql
ALTER OPERATOR FAMILY name USING index_method ADD
   { OPERATOR strategy_number operator_name ( op_type, op_type ) [ RECHECK ]
      | 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 newname
ALTER OPERATOR FAMILY name USING index_method OWNER TO newowner
```

**Description**

ALTER OPERATOR FAMILY changes the definition of an operator family. You can add operators and support functions to the family, remove them from the family, or change the family's name or owner.

When operators and support functions are added to a family with ALTER OPERATOR FAMILY, they are not part of any specific operator class within the family, but are just "loose" within the family. This indicates that these operators and functions are compatible with the family's semantics, but are not required for correct functioning of any specific index. (Operators and functions that are so required should be declared as part of an operator class, instead; see CREATE OPERATOR CLASS.) You can drop loose members of a family from the family at any time, but members of an operator class cannot be dropped without dropping the whole class and any indexes that depend on it. Typically, single-data-type operators and functions are part of operator classes because they are needed to support an index on that specific data type, while cross-data-type operators and functions are made loose members of the family.

You must be a superuser to use ALTER OPERATOR FAMILY. (This restriction is made because an erroneous operator family definition could confuse or even crash the server.)

ALTER OPERATOR FAMILY does not presently check whether the operator family definition includes all the operators and functions required by the index method, nor whether the operators and functions form a self-consistent set. It is the user's responsibility to define a valid operator family.

OPERATOR and FUNCTION clauses can appear in any order.
Parameters

name
The name (optionally schema-qualified) of an existing operator family.

index_method
The name of the index method this operator family is for.

strategy_number
The index method's strategy number for an operator associated with the operator family.

operator_name
The name (optionally schema-qualified) of an operator associated with the operator family.

op_type
In an OPERATOR clause, the operand data type(s) of the operator, or NONE to signify a
left-unary or right-unary operator. Unlike the comparable syntax in CREATE OPERATOR
CLASS, the operand data types must always be specified. In an ADD FUNCTION clause,
the operand data type(s) the function is intended to support, if different from the input
data type(s) of the function. For B-tree and hash indexes it is not necessary to specify
op_type since the function's input data type(s) are always the correct ones to use. For GIN
and GiST indexes it is necessary to specify the input data type the function is to be used
with. In a DROP FUNCTION clause, the operand data type(s) the function is intended to
support must be specified.

RECHECK
If present, the index is "lossy" for this operator, and so the rows retrieved using the index
must be rechecked to verify that they actually satisfy the qualification clause involving this
operator.

support_number
The index method's support procedure number for a function associated with the operator
family.

funcname
The name (optionally schema-qualified) of a function that is an index method support
procedure for the operator family.

argument_types
The parameter data type(s) of the function.

newname
The new name of the operator family.

tenewowner
The new owner of the operator family.

Compatibility
There is no ALTER OPERATOR FAMILY statement in the SQL standard.

Notes
Notice that the DROP syntax only specifies the "slot" in the operator family, by strategy or support number
and input data type(s). The name of the operator or function occupying the slot is not mentioned. Also, for
DROP FUNCTION the type(s) to specify are the input data type(s) the function is intended to support; for
GIN and GiST indexes this might have nothing to do with the actual input argument types of the function.

Because the index machinery does not check access permissions on functions before using them,
including a function or operator in an operator family is tantamount to granting public execute permission
on it. This is usually not an issue for the sorts of functions that are useful in an operator family.
The operators should not be defined by SQL functions. A SQL function is likely to be inlined into the calling query, which will prevent the optimizer from recognizing that the query matches an index.

Examples

The following example command adds cross-data-type operators and support functions to an operator family that already contains B-tree operator classes for data types int4 and int2:

```
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);
```

To remove these entries:

```
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);
```

See Also

`CREATE OPERATOR FAMILY`, `DROP OPERATOR FAMILY`, `ALTER OPERATOR CLASS`, `CREATE OPERATOR CLASS`, `DROP OPERATOR CLASS`

**ALTER PROTOCOL**

Changes the definition of a protocol.

**Synopsis**

```
ALTER PROTOCOL name RENAME TO newname

ALTER PROTOCOL name OWNER TO newowner
```
**Description**

`ALTER PROTOCOL` changes the definition of a protocol. Only the protocol name or owner can be altered. You must own the protocol to use `ALTER PROTOCOL`. To alter the owner, you must also be a direct or indirect member of the new owning role, and that role must have `CREATE` privilege on schema of the conversion.

These restrictions are in place to ensure that altering the owner only makes changes that could be made by dropping and recreating the protocol. Note that a superuser can alter ownership of any protocol.

**Parameters**

- **name**
  The name (optionally schema-qualified) of an existing protocol.

- **newname**
  The new name of the protocol.

- **newowner**
  The new owner of the protocol.

**Examples**

To rename the conversion `GPDBauth` to `GPDB_authentication`:

```
ALTER PROTOCOL GPDBauth RENAME TO GPDB_authentication;
```

To change the owner of the conversion `GPDB_authentication` to `joe`:

```
ALTER PROTOCOL GPDB_authentication OWNER TO joe;
```

**Compatibility**

There is no `ALTER PROTOCOL` statement in the SQL standard.

**See Also**

`CREATE EXTERNAL TABLE`, `CREATE PROTOCOL`
**Description**

`ALTER RESOURCE GROUP` changes the limits of a resource group. Only a superuser can alter a resource group.

You can set or reset the concurrency limit of a resource group to control the maximum number of active concurrent statements in that group. You can also reset the memory or CPU rate limit of a resource group to control the amount of memory or CPU resources that all queries submitted through the group can consume on each segment host.

The new resource limit is immediately applied if current resource usage is less than or equal to the new value and there are no running transactions in the resource group. If current resource usage exceeds the new limit value, or there are running transactions in other resource groups holding some of the resource, Greenplum Database will defer the new limit assignment until resource usage is within the range of the new value.

You can alter one limit type in a single `ALTER RESOURCE GROUP` call.

**Parameters**

- **name**
  The name of the resource group to alter.

- **CONCURRENCY integer**
  The maximum number of concurrent transactions, including active and idle transactions, that are permitted for this resource group. Any transactions submitted after the `CONCURRENCY` value limit is reached are queued. When a running transaction completes, the earliest queued transaction is executed.
  
  The `CONCURRENCY` value must be an integer in the range `[0 .. max_connections]`. The default `CONCURRENCY` value is 20.
  
  **Note:** You cannot set the `CONCURRENCY` value for the `admin_group` to zero (0).

- **CPU_RATE_LIMIT integer**
  The percentage of CPU resources to allocate to this resource group. The minimum CPU percentage for a resource group is 1. The maximum is 100. The sum of the `CPU_RATE_LIMIT` values of all resource groups defined in the Greenplum Database cluster must not exceed 100.

- **MEMORY_LIMIT integer**
  The percentage of memory resources to allocate to this resource group. The minimum memory percentage for a resource group is 1. The maximum is 100. The sum of the `MEMORY_LIMIT` values of all resource groups defined in the Greenplum Database cluster must not exceed 100.

- **MEMORY_SHARED_QUOTA integer**
  The percentage of memory resources to share among transactions in the resource group. The minimum memory shared quota percentage for a resource group is 0. The maximum is 100. The default `MEMORY_SHARED_QUOTA` value is 20.

- **MEMORY_SPILL_RATIO integer**
  The memory usage threshold for memory-intensive operators in a transaction issued in the resource group. The minimum memory spill ratio percentage for a resource group is 0. The maximum is 100. The default `MEMORY_SPILL_RATIO` value is 20.

**Notes**

Use `CREATE ROLE` or `ALTER ROLE` to assign a specific resource group to a role (user).

You cannot submit an `ALTER RESOURCE GROUP` command in an explicit transaction or sub-transaction.
Examples
Change the active transaction limit for a resource group:

```
ALTER RESOURCE GROUP rgroup1 SET CONCURRENCY 13;
```
Update the CPU limit for a resource group:

```
ALTER RESOURCE GROUP rgroup2 SET CPU_RATE_LIMIT 45;
```
Update the memory limit for a resource group:

```
ALTER RESOURCE GROUP rgroup3 SET MEMORY_LIMIT 30;
```
Increase the memory spill ratio for a resource group from the default:

```
ALTER RESOURCE GROUP rgroup4 SET MEMORY_SPILL_RATIO 25;
```

Compatibility
The `ALTER RESOURCE GROUP` statement is a Greenplum Database extension. This command does not exist in standard PostgreSQL.

See Also
`CREATE RESOURCE GROUP`, `DROP RESOURCE GROUP`, `CREATE ROLE`, `ALTER ROLE`

**ALTER RESOURCE QUEUE**
Changes the limits of a resource queue.

**Synopsis**

```
ALTER RESOURCE QUEUE name WITH ( queue_attribute=value [, ... ] )
```

where `queue_attribute` is:

- `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 [, ... ] )
```

where `queue_attribute` is:

- `ACTIVE_STATEMENTS`
- `MEMORY_LIMIT`
- `MAX_COST`
- `COST_OVERCOMMIT`
- `MIN_COST`

**Note:** A resource queue must have either an `ACTIVE_STATEMENTS` or a `MAX_COST` value. Do not remove both these `queue_attribute` from a resource queue.
Description

`ALTER RESOURCE QUEUE` changes the limits of a resource queue. Only a superuser can alter a resource queue. A resource queue must have either an `ACTIVE_STATEMENTS` or a `MAX_COST` value (or it can have both). You can also set or reset priority for a resource queue to control the relative share of available CPU resources used by queries associated with the queue, or memory limit of a resource queue to control the amount of memory that all queries submitted through the queue can consume on a segment host.

`ALTER RESOURCE QUEUE WITHOUT` removes the specified limits on a resource that were previously set. A resource queue must have either an `ACTIVE_STATEMENTS` or a `MAX_COST` value. Do not remove both these queue_attributes from a resource queue.

Parameters

```plaintext
name
```
The name of the resource queue whose limits are to be altered.

```plaintext
ACTIVE_STATEMENTS integer
```
The number of active statements submitted from users in this resource queue allowed on the system at any one time. The value for `ACTIVE_STATEMENTS` should be an integer greater than 0. To reset `ACTIVE_STATEMENTS` to have no limit, enter a value of -1.

```plaintext
MEMORY_LIMIT 'memory_units'
```
Sets the total memory quota for all statements submitted from users in this resource queue. Memory units can be specified in kB, MB or GB. The minimum memory quota for a resource queue is 10MB. There is no maximum; however the upper boundary at query execution time is limited by the physical memory of a segment host. The default value is no limit (-1).

```plaintext
MAX_COST float
```
The total query optimizer cost of statements submitted from users in this resource queue allowed on the system at any one time. The value for `MAX_COST` is specified as a floating point number (for example 100.0) or can also be specified as an exponent (for example 1e+2). To reset `MAX_COST` to have no limit, enter a value of -1.0.

```plaintext
COST_OVERCOMMIT boolean
```
If a resource queue is limited based on query cost, then the administrator can allow cost overcommit (`COST_OVERCOMMIT=TRUE`, the default). This means that a query that exceeds the allowed cost threshold will be allowed to run but only when the system is idle. If `COST_OVERCOMMIT=False` is specified, queries that exceed the cost limit will always be rejected and never allowed to run.

```plaintext
MIN_COST float
```
Queries with a cost under this limit will not be queued and run immediately. Cost is measured in units of disk page fetches; 1.0 equals one sequential disk page read. The value for `MIN_COST` is specified as a floating point number (for example 100.0) or can also be specified as an exponent (for example 1e+2). To reset `MIN_COST` to have no limit, enter a value of -1.0.

```plaintext
PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX}
```
Sets the priority of queries associated with a resource queue. Queries or statements in queues with higher priority levels will receive a larger share of available CPU resources in case of contention. Queries in low-priority queues may be delayed while higher priority queries are executed.

Notes

Use `CREATE ROLE` or `ALTER ROLE` to add a role (user) to a resource queue.
**Examples**

Change the active query limit for a resource queue:

```sql
ALTER RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=20);
```

Change the memory limit for a resource queue:

```sql
ALTER RESOURCE QUEUE myqueue WITH (MEMORY_LIMIT='2GB');
```

Reset the maximum and minimum query cost limit for a resource queue to no limit:

```sql
ALTER RESOURCE QUEUE myqueue WITH (MAX_COST=-1.0,
     MIN_COST=-1.0);
```

Reset the query cost limit for a resource queue to $3^{10}$ (or 30000000000.0) and do not allow overcommit:

```sql
ALTER RESOURCE QUEUE myqueue WITH (MAX_COST=3e+10,
     COST_OVERCOMMIT=FALSE);
```

Reset the priority of queries associated with a resource queue to the minimum level:

```sql
ALTER RESOURCE QUEUE myqueue WITH (PRIORITY=MIN);
```

Remove the `MAX_COST` and `MEMORY_LIMIT` limits from a resource queue:

```sql
ALTER RESOURCE QUEUE myqueue WITHOUT (MAX_COST, MEMORY_LIMIT);
```

**Compatibility**

The `ALTER RESOURCE QUEUE` statement is a Greenplum Database extension. This command does not exist in standard PostgreSQL.

**See Also**

`CREATE RESOURCE QUEUE`, `DROP RESOURCE QUEUE`, `CREATE ROLE`, `ALTER ROLE`

---

**ALTER ROLE**

Changes a database role (user or group).

**Synopsis**

```sql
ALTER ROLE name RENAME TO newname

ALTER ROLE name SET config_parameter {TO | =} {value | DEFAULT}

ALTER ROLE name RESET config_parameter

ALTER ROLE name RESOURCE QUEUE {queue_name | NONE}

ALTER ROLE name RESOURCE GROUP {group_name | NONE}

ALTER ROLE name [ [WITH] option [ ... ] ]
```

where `option` can be:

```
SUPERUSER | NOSUPERUSER
| CREATEDB | NOCREATEDB
```
ALTER ROLE changes the attributes of a Greenplum Database role. There are several variants of this command:

- **RENAME** — Changes the name of the role. Database superusers can rename any role. Roles having `CREATEROLE` privilege can rename non-superuser roles. The current session user cannot be renamed (connect as a different user to rename a role). Because MD5-encrypted passwords use the role name as cryptographic salt, renaming a role clears its password if the password is MD5-encrypted.

- **SET | RESET** — Changes a role’s session default for a specified configuration parameter. Whenever the role subsequently starts a new session, the specified value becomes the session default, overriding whatever setting is present in server configuration file (`postgresql.conf`). For a role without `LOGIN` privilege, session defaults have no effect. Ordinary roles can change their own session defaults. Superusers can change anyone’s session defaults. Roles having `CREATEROLE` privilege can change defaults for non-superuser roles. See the Greenplum Database Administrator Guide for information about all user-settable configuration parameters.

- **RESOURCE QUEUE** — Assigns the role to a resource queue. The role would then be subject to the limits assigned to the resource queue when issuing queries. Specify `NONE` to assign the role to the default resource queue. A role can only belong to one resource queue. For a role without `LOGIN` privilege, resource queues have no effect. See CREATE RESOURCE QUEUE for more information.

- **RESOURCE GROUP** — Assigns a resource group to the role. The role would then be subject to the concurrent transaction, memory, and CPU limits configured for the resource group. You can assign a single resource group to one or more roles. See CREATE RESOURCE GROUP for additional information.

  **Warning:** Resource groups are an experimental feature and are not intended for use in a production environment. Experimental features are subject to change without notice in future releases.

- **WITH option** — Changes many of the role attributes that can be specified in `CREATE ROLE`. Attributes not mentioned in the command retain their previous settings. Database superusers can change any of these settings for any role. Roles having `CREATEROLE` privilege can change any of these settings, but only for non-superuser roles. Ordinary roles can only change their own password.

**Parameters**

- **name**
  
The name of the role whose attributes are to be altered.

- **newname**
  
The new name of the role.

- **config_parameter=value**
  
  Set this role’s session default for the specified configuration parameter to the given value. If value is `DEFAULT` or if `RESET` is used, the role-specific variable setting is removed,
so the role will inherit the system-wide default setting in new sessions. Use \texttt{RESET ALL} to clear all role-specific settings. See \texttt{SET} and \texttt{Server Configuration Parameters} for information about user-settable configuration parameters.

\textbf{group\_name}

The name of the resource group to assign to this role. Specifying the \texttt{group\_name NONE} removes the role's current resource group assignment and assigns a default resource group based on the role's capability. \texttt{SUPERUSER} roles are assigned the \texttt{admin\_group resource group}, while the \texttt{default\_group resource group} is assigned to non-admin roles.

\textbf{queue\_name}

The name of the resource queue to which the user-level role is to be assigned. Only roles with \texttt{LOGIN} privilege can be assigned to a resource queue. To unassign a role from a resource queue and put it in the default resource queue, specify \texttt{NONE}. A role can only belong to one resource queue.

\begin{itemize}
\item \texttt{SUPERUSER} | \texttt{NOSUPERUSER}
\item \texttt{CREATEDB} | \texttt{NOCREATEDB}
\item \texttt{CREATEROLE} | \texttt{NOCREATEROLE}
\item \texttt{CREATEUSER} | \texttt{NOCREATEUSER}
\end{itemize}

\texttt{CREATEUSER} and \texttt{NOCREATEUSER} are obsolete, but still accepted, spellings of \texttt{SUPERUSER} and \texttt{NOSUPERUSER}. Note that they are not equivalent to the \texttt{CREATEROLE} and \texttt{NOCREATEROLE} clauses.

\texttt{CREATEEXTTABLE} | \texttt{NOCREATEEXTTABLE} [\{\texttt{attribute=}'value'\}]

If \texttt{CREATEEXTTABLE} is specified, the role being defined is allowed to create external tables. The default \texttt{type} is \texttt{readable} and the default \texttt{protocol} is \texttt{gpfdist} if not specified. \texttt{NOCREATEEXTTABLE} (the default) denies the role the ability to create external tables. Note that external tables that use the \texttt{file} or \texttt{execute} protocols can only be created by superusers.

\begin{itemize}
\item \texttt{INHERIT} | \texttt{NOINHERIT}
\item \texttt{LOGIN} | \texttt{NOLOGIN}
\item \texttt{CONNECTION LIMIT} conlimit
\item \texttt{PASSWORD} password
\item \texttt{ENCRYPTED} | \texttt{UNENCRYPTED}
\item \texttt{VALID UNTIL} 'timestamp'
\end{itemize}

These clauses alter role attributes originally set by \texttt{CREATE ROLE}.

\texttt{DENY deny\_point}

\texttt{DENY BETWEEN deny\_point AND deny\_point}

The \texttt{DENY} and \texttt{DENY BETWEEN} keywords set time-based constraints that are enforced at login. \texttt{DENY} sets a day or a day and time to deny access. \texttt{DENY BETWEEN} sets an interval during which access is denied. Both use the parameter \texttt{deny\_point} that has following format:

\begin{verbatim}
DAY day [ TIME 'time' ]
\end{verbatim}

The two parts of the \texttt{deny\_point} parameter use the following formats:

For day:

\begin{verbatim}
{'Sunday' | 'Monday' | 'Tuesday' |'Wednesday' | 'Thursday' | 'Friday' | 'Saturday' | 0-6 }
\end{verbatim}

For time:

\begin{verbatim}
{ 00-23 : 00-59 | 01-12 : 00-59 { AM | PM} }
\end{verbatim}
The `DENY BETWEEN` clause uses two `deny_point` parameters.

```sql
DENY BETWEEN deny_point AND deny_point
```

For more information about time-based constraints and examples, see "Managing Roles and Privileges" in the Greenplum Database Administrator Guide.

### DROP DENY FOR `deny_point`

The `DROP DENY FOR` clause removes a time-based constraint from the role. It uses the `deny_point` parameter described above.

For more information about time-based constraints and examples, see "Managing Roles and Privileges" in the Greenplum Database Administrator Guide.

## Notes

Use `GRANT` and `REVOKE` for adding and removing role memberships.

Caution must be exercised when specifying an unencrypted password with this command. The password will be transmitted to the server in clear text, and it might also be logged in the client's command history or the server log. The `psql` command-line client contains a meta-command `\password` that can be used to safely change a role's password.

It is also possible to tie a session default to a specific database rather than to a role. Role-specific settings override database-specific ones if there is a conflict. See `ALTER DATABASE`.

## Examples

Change the password for a role:

```sql
ALTER ROLE daria WITH PASSWORD 'passwd123';
```

Change a password expiration date:

```sql
ALTER ROLE scott VALID UNTIL 'May 4 12:00:00 2015 +1';
```

Make a password valid forever:

```sql
ALTER ROLE luke VALID UNTIL 'infinity';
```

Give a role the ability to create other roles and new databases:

```sql
ALTER ROLE joelle CREATEROLE CREATEDB;
```

Give a role a non-default setting of the `maintenance_work_mem` parameter:

```sql
ALTER ROLE admin SET maintenance_work_mem = 100000;
```

Assign a role to a resource queue:

```sql
ALTER ROLE sammy RESOURCE QUEUE poweruser;
```

Give a role permission to create writable external tables:

```sql
ALTER ROLE load CREATEEXTTABLE (type='writable');
```

Alter a role so it does not allow login access on Sundays:

```sql
ALTER ROLE user3 DENY DAY 'Sunday';
```
Alter a role to remove the constraint that does not allow login access on Sundays:

```
ALTER ROLE user3 DROP DENY FOR DAY 'Sunday';
```

Assign a new resource group to a role:

```
ALTER ROLE parttime_user RESOURCE GROUP rg_light;
```

**Compatibility**
The `ALTER ROLE` statement is a Greenplum Database extension.

**See Also**
`CREATE ROLE`, `DROP ROLE`, `SET`, `CREATE RESOURCE GROUP`, `CREATE RESOURCE QUEUE`, `GRANT`, `REVOKE`

### ALTER SCHEMA

Changes the definition of a schema.

**Synopsis**

```
ALTER SCHEMA name RENAME TO newname
ALTER SCHEMA name OWNER TO newowner
```

**Description**

`ALTER SCHEMA` changes the definition of a schema.

You must own the schema to use `ALTER SCHEMA`. To rename a schema you must also have the `CREATE` privilege for the database. To alter the owner, you must also be a direct or indirect member of the new owning role, and you must have the `CREATE` privilege for the database. Note that superusers have all these privileges automatically.

**Parameters**

- **name**
  The name of an existing schema.

- **newname**
  The new name of the schema. The new name cannot begin with `pg_`, as such names are reserved for system schemas.

- **newowner**
  The new owner of the schema.

**Compatibility**

There is no `ALTER SCHEMA` statement in the SQL standard.

**See Also**

`CREATE SCHEMA`, `DROP SCHEMA`
**ALTER SEQUENCE**

Changes the definition of a sequence generator.

**Synopsis**

```sql
ALTER SEQUENCE name [INCREMENT [ BY ] increment]
[MINVALUE minvalue | NO MINVALUE]
[MAXVALUE maxvalue | NO MAXVALUE]
[RESTART [ WITH ] start]
[CACHE cache] [[ NO ] CYCLE]
[OWNED BY {table.column | NONE}]
```

```sql
ALTER SEQUENCE name RENAME TO new_name

ALTER SEQUENCE name SET SCHEMA new_schema
```

**Description**

`ALTER SEQUENCE` changes the parameters of an existing sequence generator. Any parameters not specifically set in the `ALTER SEQUENCE` command retain their prior settings.

You must own the sequence to use `ALTER SEQUENCE`. To change a sequence's schema, you must also have `CREATE` privilege on the new schema. Note that superusers have all these privileges automatically.

**Parameters**

**name**

The name (optionally schema-qualified) of a sequence to be altered.

**increment**

The clause `INCREMENT BY increment` is optional. A positive value will make an ascending sequence, a negative one a descending sequence. If unspecified, the old increment value will be maintained.

**minvalue**

NO MINVALUE

The optional clause `MINVALUE minvalue` determines the minimum value a sequence can generate. If `NO MINVALUE` is specified, the defaults of 1 and -263-1 for ascending and descending sequences, respectively, will be used. If neither option is specified, the current minimum value will be maintained.

**maxvalue**

NO MAXVALUE

The optional clause `MAXVALUE maxvalue` determines the maximum value for the sequence. If `NO MAXVALUE` is specified, the defaults are 263-1 and -1 for ascending and descending sequences, respectively, will be used. If neither option is specified, the current maximum value will be maintained.

**start**

The optional clause `RESTART WITH start` changes the current value of the sequence. Altering the sequence in this manner is equivalent to calling the `setval(sequence, start_val, is_called)` function with `is_called = false`. The first `nextval()` call after you alter the sequence start value does not increment the sequence and returns `start`.

**cache**
The clause `CACHE cache` enables sequence numbers to be preallocated and stored in memory for faster access. The minimum value is 1 (only one value can be generated at a time, i.e., no cache). If unspecified, the old cache value will be maintained.

**CYCLE**

The optional `CYCLE` key word may be used to enable the sequence to wrap around when the `maxvalue` or `minvalue` has been reached by an ascending or descending sequence. If the limit is reached, the next number generated will be the respective `minvalue` or `maxvalue`.

**NO CYCLE**

If the optional `NO CYCLE` key word is specified, any calls to `nextval()` after the sequence has reached its maximum value will return an error. If neither `CYCLE` or `NO CYCLE` are specified, the old cycle behavior will be maintained.

**OWNED BY table.column**

The `OWNED BY` option causes the sequence to be associated with a specific table column, such that if that column (or its whole table) is dropped, the sequence will be automatically dropped as well. If specified, this association replaces any previously specified association for the sequence. The specified table must have the same owner and be in the same schema as the sequence. Specifying `OWNED BY NONE` removes any existing table column association.

```
new_name
```

The new name for the sequence.

```
new_schema
```

The new schema for the sequence.

**Notes**

To avoid blocking of concurrent transactions that obtain numbers from the same sequence, `ALTER SEQUENCE`'s effects on the sequence generation parameters are never rolled back; those changes take effect immediately and are not reversible. However, the `OWNED BY`, `RENAME TO`, and `SET SCHEMA` clauses are ordinary catalog updates and can be rolled back.

`ALTER SEQUENCE` will not immediately affect `nextval()` results in sessions, other than the current one, that have preallocated (cached) sequence values. They will use up all cached values prior to noticing the changed sequence generation parameters. The current session will be affected immediately.

Some variants of `ALTER TABLE` can be used with sequences as well. For example, to rename a sequence use `ALTER TABLE RENAME`.

**Examples**

Restart a sequence called `serial` at 105:

```
ALTER SEQUENCE serial RESTART WITH 105;
```

**Compatibility**

`ALTER SEQUENCE` conforms to the SQL standard, except for the `OWNED BY`, `RENAME TO`, and `SET SCHEMA` clauses, which are Greenplum Database extensions.

**See Also**

`CREATE SEQUENCE`, `DROP SEQUENCE`, `ALTER TABLE`
**ALTER TABLE**

Changes the definition of a table.

**Synopsis**

```sql
ALTER TABLE [ONLY] name RENAME [COLUMN] column TO new_column

ALTER TABLE name RENAME TO new_name

ALTER TABLE name SET SCHEMA new_schema

ALTER TABLE [ONLY] name SET DISTRIBUTED BY (column, [ ... ] )
| DISTRIBUTED RANDOMLY
| WITH (REORGANIZE=true|false)

ALTER TABLE [ONLY] name action [, ... ]

ALTER TABLE name
[ ALTER PARTITION { partition_name | FOR (RANK(number))
| FOR (value) } partition_action [...] ]

partition_action
```

where **action** is one of:

```sql
ADD [COLUMN] column_name type
[columConstraint [ ... ]]
DROP [COLUMN] column [RESTRICT | CASCADE]
ALTER [COLUMN] column TYPE type [USING expression]
ALTER [COLUMN] column SET DEFAULT expression
ALTER [COLUMN] column DROP DEFAULT
ALTER [COLUMN] column (SET | DROP ) NOT NULL
ALTER [COLUMN] column SET STATISTICS integer
ADD table_constraint
DROP CONSTRAINT constraint_name [RESTRICT | CASCADE]
DISABLE TRIGGER [trigger_name | ALL | USER]
ENABLE TRIGGER [trigger_name | ALL | USER]
CLUSTER ON index_name
SET WITHOUT CLUSTER
SET WITHOUT OIDS
SET (FILLFACTOR = value)
RESET (FILLFACTOR)
INHERIT parent_table
NO INHERIT parent_table
OWNER TO new_owner
SET TABLESPACE new_tablespace
```

where **partition_action** is one of:

```sql
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:

APPENDONLY={TRUE|FALSE}
BLOCKSIZE={8192-2097152}
ORIENTATION={COLUMN|ROW}
COMPRESSTYPE={ZLIB|QUICKLZ|RLE_TYPE|NONE}
COMPRESSLEVEL={0-9}
FILLFACTOR={10-100}
OIDS=[=TRUE|FALSE]

Description
ALTER TABLE changes the definition of an existing table. There are several subforms:
• **ADD COLUMN** — Adds a new column to the table, using the same syntax as `CREATE TABLE`. When adding a column to an append-optimized table a **DEFAULT** clause is required.

• **DROP COLUMN** — Drops a column from a table. Note that if you drop table columns that are being used as the Greenplum Database distribution key, the distribution policy for the table will be changed to **DISTRIBUTED RANDOMLY**. Indexes and table constraints involving the column will be automatically dropped as well. You will need to say **CASCADE** if anything outside the table depends on the column (such as views).

• **ALTER COLUMN TYPE** — Changes the data type of a column of a table. Note that you cannot alter column data types that are being used as distribution or partitioning keys. Indexes and simple table constraints involving the column will be automatically converted to use the new column type by reparsing the originally supplied expression. The optional **USING** clause specifies how to compute the new column value from the old. If omitted, the default conversion is the same as an assignment cast from old data type to new. A **USING** clause must be provided if there is no implicit or assignment cast from old to new type.

• **SET/DROP DEFAULT** — Sets or removes the default value for a column. The default values only apply to subsequent **INSERT** commands. They do not cause rows already in the table to change. Defaults may also be created for views, in which case they are inserted into statements on the view before the view’s **ON INSERT** rule is applied.

• **SET/DROP NOT NULL** — Changes whether a column is marked to allow null values or to reject null values. You can only use **SET NOT NULL** when the column contains no null values.

• **SET STATISTICS** — Sets the per-column statistics-gathering target for subsequent **ANALYZE** operations. The target can be set in the range 0 to 1000, or set to -1 to revert to using the system default statistics target (**default_statistics_target**).

• **ADD table_constraint** — Adds a new constraint to a table (not just a partition) using the same syntax as `CREATE TABLE`.

• **DROP CONSTRAINT** — Drops the specified constraint on a table.

• **DISABLE/ENABLE TRIGGER** — Disables or enables trigger(s) belonging to the table. A disabled trigger is still known to the system, but is not executed when its triggering event occurs. For a deferred trigger, the enable status is checked when the event occurs, not when the trigger function is actually executed. One may disable or enable a single trigger specified by name, or all triggers on the table, or only user-created triggers. Disabling or enabling constraint triggers requires superuser privileges.

  **Note:** Triggers are not supported in Greenplum Database. Triggers in general have very limited functionality due to the parallelism of Greenplum Database.

• **CLUSTER ON/SET WITHOUT CLUSTER** — Selects or removes the default index for future **CLUSTER** operations. It does not actually re-cluster the table. Note that **CLUSTER** is not the recommended way to physically reorder a table in Greenplum Database because it takes so long. It is better to recreate the table with **CREATE TABLE AS** and order it by the index column(s).

  **Note:** **CLUSTER ON** is not supported on append-optimized tables.

• **SET WITHOUT OIDS** — Removes the OID system column from the table. Note that there is no variant of **ALTER TABLE** that allows OIDs to be restored to a table once they have been removed.

• **SET ( FILLFACTOR = value ) / RESET (FILLFACTOR)** — Changes the fillfactor for the table. The fillfactor for a table is a percentage between 10 and 100. 100 (complete packing) is the default. When a smaller fillfactor is specified, **INSERT** operations pack table pages only to the indicated percentage; the remaining space on each page is reserved for updating rows on that page. This gives **UPDATE** a chance to place the updated copy of a row on the same page as the original, which is more efficient than placing it on a different page. For a table whose entries are never updated, complete packing is the best choice, but in heavily updated tables smaller fillfactors are appropriate. Note that the table contents will not be modified immediately by this command. You will need to rewrite the table to get the desired effects.

• **SET DISTRIBUTED** — Changes the distribution policy of a table. Changes to a hash distribution policy will cause the table data to be physically redistributed on disk, which can be resource intensive.

• **INHERIT parent_table / NO INHERIT parent_table** — Adds or removes the target table as a child of the specified parent table. Queries against the parent will include records of its child table. To be
added as a child, the target table must already contain all the same columns as the parent (it could
have additional columns, too). The columns must have matching data types, and if they have NOT NULL
constraints in the parent then they must also have NOT NULL constraints in the child. There must also
be matching child-table constraints for all CHECK constraints of the parent.

- **OWNER** — Changes the owner of the table, sequence, or view to the specified user.
- **SET TABLESPACE** — Changes the table's tablespace to the specified tablespace and moves the
data file(s) associated with the table to the new tablespace. Indexes on the table, if any, are not moved;
but they can be moved separately with additional SET TABLESPACE commands. See also CREATE
TABLESPACE. If changing the tablespace of a partitioned table, all child table partitions will also be
moved to the new tablespace.
- **RENAME** — Changes the name of a table (or an index, sequence, or view) or the name of an individual
column in a table. There is no effect on the stored data. Note that Greenplum Database distribution key
columns cannot be renamed.
- **SET SCHEMA** — Moves the table into another schema. Associated indexes, constraints, and
sequences owned by table columns are moved as well.
- **ALTER PARTITION | DROP PARTITION | RENAME PARTITION | TRUNCATE PARTITION | ADD
PARTITION | SPLIT PARTITION | EXCHANGE PARTITION | SET SUBPARTITION TEMPLATE** —
Changes the structure of a partitioned table. In most cases, you must go through the parent table to
alter one of its child table partitions.

**Note:** If you add a partition to a table that has subpartition encodings, the new partition inherits the
storage directives for the subpartitions. For more information about the precedence of compression
settings, see “Using Compression” in the Greenplum Database Administrator Guide.

You must own the table to use **ALTER TABLE**. To change the schema of a table, you must also have
CREATE privilege on the new schema. To add the table as a new child of a parent table, you must own the
parent table as well. To alter the owner, you must also be a direct or indirect member of the new owning
role, and that role must have CREATE privilege on the table's schema. A superuser has these privileges
automatically.

**Note:** Memory usage increases significantly when a table has many partitions, if a table has
compression, or if the blocksize for a table is large. If the number of relations associated with
the table is large, this condition can force an operation on the table to use more memory. For
example, if the table is a CO table and has a large number of columns, each column is a relation.
An operation like **ALTER TABLE ALTER COLUMN** opens all the columns in the table allocates
associated buffers. If a CO table has 40 columns and 100 partitions, and the columns are
compressed and the blocksize is 2 MB (with a system factor of 3), the system attempts to allocate
24 GB, that is (40 \times 100) \times (2 \times 3) MB or 24 GB.

**Parameters**

**ONLY**

Only perform the operation on the table name specified. If the **ONLY** keyword is not
used, the operation will be performed on the named table and any child table partitions
associated with that table.

**name**

The name (possibly schema-qualified) of an existing table to alter. If **ONLY** is specified, only
that table is altered. If **ONLY** is not specified, the table and all its descendant tables (if any)
are updated.

**Note:** Constraints can only be added to an entire table, not to a partition.
Because of that restriction, the **name** parameter can only contain a table
name, not a partition name.

**column**

Name of a new or existing column. Note that Greenplum Database distribution key
columns must be treated with special care. Altering or dropping these columns can change
the distribution policy for the table.
new_column
New name for an existing column.

type
Data type of the new column, or new data type for an existing column. If changing the
data type of a Greenplum distribution key column, you are only allowed to change it to a
compatible type (for example, text to varchar is OK, but text to int is not).

table_constraint
New table constraint for the table. Note that foreign key constraints are currently not
supported in Greenplum Database. Also a table is only allowed one unique constraint and
the uniqueness must be within the Greenplum Database distribution key.

constraint_name
Name of an existing constraint to drop.

CASCADE
Automatically drop objects that depend on the dropped column or constraint (for example,
views referencing the column).

RESTRICT
Refuse to drop the column or constraint if there are any dependent objects. This is the
default behavior.

trigger_name
Name of a single trigger to disable or enable. Note that Greenplum Database does not
support triggers.

ALL
Disable or enable all triggers belonging to the table including constraint related triggers.
This requires superuser privilege.

USER
Disable or enable all user-created triggers belonging to the table.

index_name
The index name on which the table should be marked for clustering. Note that CLUSTER is
not the recommended way to physically reorder a table in Greenplum Database because it
takes so long. It is better to recreate the table with CREATE TABLE AS and order it by the
index column(s).

FILLFACTOR
Set the fillfactor percentage for a table.

value
The new value for the FILLFACTOR parameter, which is a percentage between 10 and
100. 100 is the default.

DISTRIBUTED BY (column) | DISTRIBUTED RANDOMLY
Specifies the distribution policy for a table. Changing a hash distribution policy will cause
the table data to be physically redistributed on disk, which can be resource intensive. If you
declare the same hash distribution policy or change from hash to random distribution, data
will not be redistributed unless you declare SET WITH (REORGANIZE=true).

REORGANIZE=true|false
Use `REORGANIZE=true` when the hash distribution policy has not changed or when you have changed from a hash to a random distribution, and you want to redistribute the data anyways.

**parent_table**
A parent table to associate or de-associate with this table.

**new_owner**
The role name of the new owner of the table.

**new_tablespace**
The name of the tablespace to which the table will be moved.

**new_schema**
The name of the schema to which the table will be moved.

**parent_table_name**
When altering a partitioned table, the name of the top-level parent table.

**ALTER [DEFAULT] PARTITION**
If altering a partition deeper than the first level of partitions, the `ALTER PARTITION` clause is used to specify which subpartition in the hierarchy you want to alter.

**DROP [DEFAULT] PARTITION**
Drops the specified partition. If the partition has subpartitions, the subpartitions are automatically dropped as well.

**TRUNCATE [DEFAULT] PARTITION**
Truncates the specified partition. If the partition has subpartitions, the subpartitions are automatically truncated as well.

**RENAME [DEFAULT] PARTITION**
Changes the partition name of a partition (not the relation name). Partitioned tables are created using the naming convention: `<parentname>_<level>_prt_<partition_name>`.

**ADD DEFAULT PARTITION**
Adds a default partition to an existing partition design. When data does not match to an existing partition, it is inserted into the default partition. Partition designs that do not have a default partition will reject incoming rows that do not match to an existing partition. Default partitions must be given a name.

**ADD PARTITION**

- `partition_element` - Using the existing partition type of the table (range or list), defines the boundaries of new partition you are adding.
- `name` - A name for this new partition.

**VALUES** - For list partitions, defines the value(s) that the partition will contain.

**START** - For range partitions, defines the starting range value for the partition. By default, start values are **INCLUSIVE**. For example, if you declared a start date of '2016-01-01', then the partition would contain all dates greater than or equal to '2016-01-01'. Typically the data type of the **START** expression is the same type as the partition key column. If that is not the case, then you must explicitly cast to the intended data type.

**END** - For range partitions, defines the ending range value for the partition. By default, end values are **EXCLUSIVE**. For example, if you declared an end date of '2016-02-01', then the partition would contain all dates less than but not equal to '2016-02-01'. Typically the data type of the **END** expression is the same type as the partition key column. If that is not the case, then you must explicitly cast to the intended data type.
WITH - Sets the table storage options for a partition. For example, you may want older partitions to be append-optimized tables and newer partitions to be regular heap tables. See CREATE TABLE for a description of the storage options.

TABLESPACE - The name of the tablespace in which the partition is to be created.

subpartition_spec - Only allowed on partition designs that were created without a subpartition template. Declares a subpartition specification for the new partition you are adding. If the partitioned table was originally defined using a subpartition template, then the template will be used to generate the subpartitions automatically.

EXCHANGE [DEFAULT] PARTITION

Exchanges another table into the partition hierarchy into the place of an existing partition. In a multi-level partition design, you can only exchange the lowest level partitions (those that contain data).

The Greenplum Database server configuration parameter gp_enable_exchange_default_partition controls availability of the EXCHANGE DEFAULT PARTITION clause. The default value for the parameter is off. The clause is not available and Greenplum Database returns an error if the clause is specified in an ALTER TABLE command.

For information about the parameter, see Server Configuration Parameters.

Warning: Before you exchange the default partition, you must ensure the data in the table to be exchanged, the new default partition, is valid for the default partition. For example, the data in the new default partition must not contain data that would be valid in other leaf child partitions of the partitioned table. Otherwise, queries against the partitioned table with the exchanged default partition that are executed by GPORCA might return incorrect results.

WITH TABLE table_name - The name of the table you are swapping into the partition design. You can exchange a table where the table data is stored in the database. For example, the table is created with the CREATE TABLE command.

With the EXCHANGE PARTITION clause, you can also exchange a readable external table (created with the CREATE EXTERNAL TABLE command) into the partition hierarchy in the place of an existing leaf child partition. If you specify a readable external table, you must also specify the WITHOUT VALIDATION clause to skip table validation against the CHECK constraint of the partition you are exchanging.

Exchanging a leaf child partition with an external table is not supported in these cases:

- The partitioned table is created with the SUBPARTITION clause or if a partition has a subpartition.
- The partitioned table contains a column with a check constraint or a NOT NULL constraint.

WITH | WITHOUT VALIDATION - Validates that the data in the table matches the CHECK constraint of the partition you are exchanging. The default is to validate the data against the CHECK constraint.

Warning: If you specify the WITHOUT VALIDATION clause, you must ensure that the data in table that you are exchanging for an existing child leaf partition is valid against the CHECK constraints on the partition. Otherwise, queries against the partitioned table might return incorrect results.

SET SUBPARTITION TEMPLATE

Modifies the subpartition template for an existing partition. After a new subpartition template is set, all new partitions added will have the new subpartition design (existing partitions are not modified).

SPLIT DEFAULT PARTITION
Splits a default partition. Only a range partition can be split, not a list partition. In a multi-level partition design, you can only split the lowest level default partitions (those that contain data). Splitting a default partition creates a new partition containing the values specified and leaves the default partition containing any values that do not match to an existing partition.

**AT** - For list partitioned tables, specifies a single list value that should be used as the criteria for the split.

**START** - For range partitioned tables, specifies a starting value for the new partition.

**END** - For range partitioned tables, specifies an ending value for the new partition.

**INTO** - Allows you to specify a name for the new partition. When using the **INTO** clause to split a default partition, the second partition name specified should always be that of the existing default partition. If you do not know the name of the default partition, you can look it up using the `pg_partitions` view.

---

**SPLIT PARTITION**

Splits an existing partition into two partitions. Only a range partition can be split, not a list partition. In a multi-level partition design, you can only split the lowest level partitions (those that contain data).

**AT** - Specifies a single value that should be used as the criteria for the split. The partition will be divided into two new partitions with the split value specified being the starting range for the **latter** partition.

**INTO** - Allows you to specify names for the two new partitions created by the split.

```sql
partition_name
```

The given name of a partition.

```sql
FOR (RANK(number))
```

For range partitions, the rank of the partition in the range.

```sql
FOR ('value')
```

Specifies a partition by declaring a value that falls within the partition boundary specification. If the value declared with **FOR** matches to both a partition and one of its subpartitions (for example, if the value is a date and the table is partitioned by month and then by day), then **FOR** will operate on the first level where a match is found (for example, the monthly partition). If your intent is to operate on a subpartition, you must declare so as follows: ```ALTER TABLE name ALTER PARTITION FOR ('2016-10-01') DROP PARTITION FOR ('2016-10-01');```

---

**Notes**

The table name specified in the **ALTER TABLE** command cannot be the name of a partition within a table.

Take special care when altering or dropping columns that are part of the Greenplum Database distribution key as this can change the distribution policy for the table.

Greenplum Database does not currently support foreign key constraints. For a unique constraint to be enforced in Greenplum Database, the table must be hash-distributed (not **DISTRIBUTED RANDOMLY**), and all of the distribution key columns must be the same as the initial columns of the unique constraint columns.

Adding a **CHECK** or **NOT NULL** constraint requires scanning the table to verify that existing rows meet the constraint.

When a column is added with **ADD COLUMN**, all existing rows in the table are initialized with the column's default value, or **NULL** if no **DEFAULT** clause is specified. (Note that adding a column to an append-optimized table without specifying a default value is not allowed.) Adding a column with a non-null default
or changing the type of an existing column will require the entire table to be rewritten. This may take a significant amount of time for a large table; and it will temporarily require double the disk space.

You can specify multiple changes in a single `ALTER TABLE` command, which will be done in a single pass over the table.

The `DROP COLUMN` form does not physically remove the column, but simply makes it invisible to SQL operations. Subsequent insert and update operations in the table will store a null value for the column. Thus, dropping a column is quick but it will not immediately reduce the on-disk size of your table, as the space occupied by the dropped column is not reclaimed. The space will be reclaimed over time as existing rows are updated.

The fact that `ALTER TYPE` requires rewriting the whole table is sometimes an advantage, because the rewriting process eliminates any dead space in the table. For example, to reclaim the space occupied by a dropped column immediately, the fastest way is:

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

where `anycol` is any remaining table column and `sametype` is the same type that column already has. This results in no semantically-visible change in the table, but the command forces rewriting, which gets rid of no-longer-useful data.

If a table is partitioned or has any descendant tables, it is not permitted to add, rename, or change the type of a column in the parent table without doing the same to the descendants. This ensures that the descendants always have columns matching the parent.

To see the structure of a partitioned table, you can use the view `pg_partitions`. This view can help identify the particular partitions you may want to alter.

A recursive `DROP COLUMN` operation will remove a descendant table’s column only if the descendant does not inherit that column from any other parents and never had an independent definition of the column. A nonrecursive `DROP COLUMN` operation never removes any descendant columns, but instead marks them as independently defined rather than inherited.

The `TRIGGER`, `CLUSTER`, `OWNER`, and `TABLESPACE` actions never recurse to descendant tables; that is, they always act as though `ONLY` were specified. Adding a constraint can recurse only for `CHECK` constraints.

These `ALTER PARTITION` operations are supported if no data is changed on a partitioned table that contains a leaf child partition that has been exchanged to use an external table. Otherwise, an error is returned.

- Adding or dropping a column.
- Changing the data type of column.

These `ALTER PARTITION` operations are not supported for a partitioned table that contains a leaf child partition that has been exchanged to use an external table:

- 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.

Changing any part of a system catalog table is not permitted.

**Examples**

Add a column to a table:

```
ALTER TABLE distributors ADD COLUMN address varchar(30);
```
Rename an existing column:

```sql
ALTER TABLE distributors RENAME COLUMN address TO city;
```

Rename an existing table:

```sql
ALTER TABLE distributors RENAME TO suppliers;
```

Add a not-null constraint to a column:

```sql
ALTER TABLE distributors ALTER COLUMN street SET NOT NULL;
```

Add a check constraint to a table:

```sql
ALTER TABLE distributors ADD CONSTRAINT zipchk CHECK (char_length(zipcode) = 5);
```

Move a table to a different schema:

```sql
ALTER TABLE myschema.distributors SET SCHEMA yourschema;
```

Add a new partition to a partitioned table:

```sql
ALTER TABLE sales ADD PARTITION
START (date '2017-02-01') INCLUSIVE
END (date '2017-03-01') EXCLUSIVE;
```

Add a default partition to an existing partition design:

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

Rename a partition:

```sql
ALTER TABLE sales RENAME PARTITION FOR ('2016-01-01') TO
jan08;
```

Drop the first (oldest) partition in a range sequence:

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

Exchange a table into your partition design:

```sql
ALTER TABLE sales EXCHANGE PARTITION FOR ('2016-01-01') WITH
TABLE jan08;
```

Split the default partition (where the existing default partition's name is `other`) to add a new monthly partition for January 2017:

```sql
ALTER TABLE sales SPLIT DEFAULT PARTITION
START ('2017-01-01') INCLUSIVE
END ('2017-02-01') EXCLUSIVE
INTO (PARTITION jan09, PARTITION other);
```

Split a monthly partition into two with the first partition containing dates January 1-15 and the second partition containing dates January 16-31:

```sql
ALTER TABLE sales SPLIT PARTITION FOR ('2016-01-01')
AT ('2016-01-16')
INTO (PARTITION jan081to15, PARTITION jan0816to31);

Compatibility
The ADD, DROP, and SET DEFAULT forms conform with the SQL standard. The other forms are Greenplum Database extensions of the SQL standard. Also, the ability to specify more than one manipulation in a single ALTER TABLE command is an extension.

ALTER TABLE DROP COLUMN can be used to drop the only column of a table, leaving a zero-column table. This is an extension of SQL, which disallows zero-column tables.

See Also
CREATE TABLE, DROP TABLE

ALTER TABLESPACE
Changes the definition of a tablespace.

Synopsis

ALTER TABLESPACE name RENAME TO newname

ALTER TABLESPACE name OWNER TO newowner

Description
ALTER TABLESPACE changes the definition of a tablespace.

You must own the tablespace to use ALTER TABLESPACE. To alter the owner, you must also be a direct or indirect member of the new owning role. (Note that superusers have these privileges automatically.)

Parameters

name
The name of an existing tablespace.

newname
The new name of the tablespace. The new name cannot begin with pg_ or gp_ (reserved for system tablespaces).

newowner
The new owner of the tablespace.

Examples

Rename tablespace index_space to fast_raid:

ALTER TABLESPACE index_space RENAME TO fast_raid;

Change the owner of tablespace index_space:

ALTER TABLESPACE index_space OWNER TO mary;

Compatibility
There is no ALTER TABLESPACE statement in the SQL standard.
ALTER TYPE
Changes the definition of a data type.

Synopsis

```
ALTER TYPE name
    OWNER TO new_owner | SET SCHEMA new_schema
```

Description

ALTER TYPE changes the definition of an existing type. You can change the owner and the schema of a type.

You must own the type to use ALTER TYPE. To change the schema of a type, you must also have CREATE privilege on the new schema. To alter the owner, you must also be a direct or indirect member of the new owning role, and that role must have CREATE privilege on the type's schema. (These restrictions enforce that altering the owner does not do anything that could be done by dropping and recreating the type. However, a superuser can alter ownership of any type.)

Parameters

- `name` The name (optionally schema-qualified) of an existing type to alter.
- `new_owner` The user name of the new owner of the type.
- `new_schema` The new schema for the type.

Examples

To change the owner of the user-defined type `email` to `joe`:

```
ALTER TYPE email OWNER TO joe;
```

To change the schema of the user-defined type `email` to `customers`:

```
ALTER TYPE email SET SCHEMA customers;
```

Compatibility

There is no ALTER TYPE statement in the SQL standard.

See Also

CREATE TYPE, DROP TYPE

ALTER USER
Changes the definition of a database role (user).
## Synopsis

```sql
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 [ ... ] ]
```

where `option` can be:

- 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
  - CONNECTION LIMIT conlimit
  - [ENCRYPTED | UNENCRYPTED] PASSWORD 'password'
  - VALID UNTIL 'timestamp'
  - [ DENY deny_point ]
  - [ DENY BETWEEN deny_point AND deny_point]
  - [ DROP DENY FOR deny_point ]
```

## Description

`ALTER USER` is an alias for `ALTER ROLE`. See `ALTER ROLE` for more information.

## Compatibility

The `ALTER USER` statement is a Greenplum Database extension. The SQL standard leaves the definition of users to the implementation.

## See Also

`ALTER ROLE`

### `ALTER VIEW`

Changes the definition of a view.

## Synopsis

```sql
ALTER VIEW name RENAME TO newname
```

## Description

`ALTER VIEW` changes the definition of a view. The only currently available functionality is to rename the view. To execute this command you must be the owner of the view.
Parameters

name

The (optionally schema-qualified) name of an existing filespace.

newname

The new name of the view.

Notes

Some variants of ALTER TABLE can be used with views as well; for example, to rename a view, it is also possible to use ALTER TABLE RENAME. To change the schema or owner of a view, you currently must use ALTER TABLE.

Examples

Rename the view myview to newview:

ALTER VIEW myview RENAME TO newview;

Change the owner of tablespace myfs:

ALTER FILESPACE myfs OWNER TO dba;

Compatibility

ALTER VIEW is a Greenplum Database extension of the SQL standard.

See Also

CREATE VIEW, DROP VIEW in the Greenplum Database Utility Guide

ANALYZE

Collects statistics about a database.

Synopsis

ANALYZE [VERBOSE] [ROOTPARTITION [ALL] ]
[ table [ (column [, ...] ) ] ]

Description

ANALYZE collects statistics about the contents of tables in the database, and stores the results in the system table pg_statistic. Subsequently, Greenplum Database uses these statistics to help determine the most efficient execution plans for queries.

With no parameter, ANALYZE collects statistics for every table in the current database. You can specify a table name to collect statistics for a single table. You can specify a set of column names, in which case the statistics only for those columns are collected.

ANALYZE does not collect statistics on external tables.

Important: If you intend to execute queries on partitioned tables with GPORCA enabled (the default), then you must collect statistics on the root partition of the partitioned table with the ANALYZE ROOTPARTITION command. For information about GPORCA, see “Querying Data” in the Greenplum Database Administrator Guide.
**Note:** You can also use the Greenplum Database utility `analyzedb` to update table statistics. The `analyzedb` utility can update statistics for multiple tables concurrently. The utility can also check table statistics and update statistics only if the statistics are not current or do not exist. For information about the utility, see the *Greenplum Database Utility Guide*.

**Parameters**

**ROOTPARTITION [ALL]**

Collect statistics only on the root partition of partitioned tables based on the data in the partitioned table. Statistics are not collected on the leaf child partitions, the data is only sampled. When you specify `ROOTPARTITION`, you must specify either `ALL` or the name of a partitioned table.

If you specify `ALL` with `ROOTPARTITION`, Greenplum Database collects statistics for the root partition of all partitioned tables in the database. If there are no partitioned tables in the database, a message stating that there are no partitioned tables is returned. For tables that are not partitioned tables, statistics are not collected.

If you specify a table name with `ROOTPARTITION` and the table is not a partitioned table, no statistics are collected for the table and a warning message is returned.

The `ROOTPARTITION` clause is not valid with `VACUUM ANALYZE`. The command `VACUUM ANALYZE ROOTPARTITION` returns an error.

The time to run `ANALYZE ROOTPARTITION` is similar to the time to analyze a non-partitioned table with the same data since `ANALYZE ROOTPARTITION` only samples the leaf child partition data.

For the partitioned table `sales_curr_yr`, this example command collects statistics only on the root partition of the partitioned table. `ANALYZE ROOTPARTITION sales_curr_yr;`

This example `ANALYZE` command collects statistics on the root partition of all the partitioned tables in the database.

```
ANALYZE ROOTPARTITION ALL;
```

**VERBOSE**

Enables display of progress messages. Enables display of progress messages. When specified, `ANALYZE` emits this information

- The table that is being processed.
- The query that is executed to generate the sample table.
- The column for which statistics is being computed.
- The queries that are issued to collect the different statistics for a single column.
- The statistics that are generated.

**table**

The name (possibly schema-qualified) of a specific table to analyze. Defaults to all tables in the current database.

**column**

The name of a specific column to analyze. Defaults to all columns.

**Notes**

It is a good idea to run `ANALYZE` periodically, or just after making major changes in the contents of a table. Accurate statistics helps Greenplum Database choose the most appropriate query plan, and thereby improve the speed of query processing. A common strategy is to run `VACUUM` and `ANALYZE` once a day during a low-usage time of day.
ANALYZE requires SHARE UPDATE EXCLUSIVE lock on the target table. This lock conflicts with these locks: SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE.

For a partitioned table, specifying which portion of the table to analyze, the root partition or subpartitions (leaf child tables) can be useful if the partitioned table has large number of partitions that have been analyzed and only a few leaf child tables have changed.

- When you run ANALYZE on the root partitioned table, statistics are collected for all the leaf child tables (the lowest-level tables in the hierarchy of child tables created by Greenplum Database for use by the partitioned table).
- When you run ANALYZE on a leaf child table, statistics are collected only for that leaf child table. When you run ANALYZE on a child table that is not a leaf child table, statistics are not collected.

For example, you can create a partitioned table with partitions for the years 2006 to 2016 and subpartitions for each month in each year. If you run ANALYZE on the child table for the year 2013 no statistics are collected. If you run ANALYZE on the leaf child table for March of 2013, statistics are collected only for that leaf child table.

**Note:** When you create a partitioned table with the `CREATE TABLE` command, Greenplum Database creates the table that you specify (the root partition or parent table), and also creates a hierarchy of tables based on the partition hierarchy that you specified (the child tables). Partitioned tables, child tables and their inheritance level relationships are tracked in the system view `pg_partitions`.

For a partitioned table that contains a leaf child partition that has been exchanged to use an external table, ANALYZE does not collect statistics for the external table partition:

- If `ANALYZE [ROOTPARTITION]` is run, external table partitions are not sampled and root table statistics do not include external table partition.
- If `ANALYZE` is run on an external table partition, the partition is not analyzed.
- If the `VERBOSE` clause is specified, an informational message is displayed: skipping external table.

The Greenplum Database server configuration parameter `optimizer_analyze_root_partition` affects when statistics are collected on the root partition of a partitioned table. If the parameter is enabled, statistics are also collected on the root partition when you run `ANALYZE` (without the `ROOTPARTITION` keyword) and specify the root partition.

The statistics collected by ANALYZE usually include a list of some of the most common values in each column and a histogram showing the approximate data distribution in each column. One or both of these may be omitted if `ANALYZE` deems them uninteresting (for example, in a unique-key column, there are no common values) or if the column data type does not support the appropriate operators.

For large tables, ANALYZE takes a random sample of the table contents, rather than examining every row. This allows even very large tables to be analyzed in a small amount of time. Note, however, that the statistics are only approximate, and will change slightly each time ANALYZE is run, even if the actual table contents did not change. This may result in small changes in the planner's estimated costs shown by EXPLAIN. In rare situations, this non-determinism will cause the query optimizer to choose a different query plan between runs of ANALYZE. To avoid this, raise the amount of statistics collected by ANALYZE by adjusting the `default_statistics_target` configuration parameter, or on a column-by-column basis by setting the per-column statistics target with `ALTER TABLE ... ALTER COLUMN ... SET STATISTICS` (see ALTER TABLE). The target value sets the maximum number of entries in the most-common-value list and the maximum number of bins in the histogram. The default target value is 10, but this can be adjusted up or down to trade off accuracy of planner estimates against the time taken for ANALYZE and the amount of space occupied in `pg_statistic`. In particular, setting the statistics target to zero disables collection of statistics for that column. It may be useful to do that for columns that are never used as part of the `WHERE`, `GROUP BY`, or `ORDER BY` clauses of queries, since the planner will have no use for statistics on such columns.
The largest statistics target among the columns being analyzed determines the number of table rows sampled to prepare the statistics. Increasing the target causes a proportional increase in the time and space needed to do \texttt{ANALYZE}.

When Greenplum Database performs an \texttt{ANALYZE} operation to collect statistics for a table and detects that all the sampled table data pages are empty (do not contain valid data), Greenplum Database displays a message that a \texttt{VACUUM FULL} operation should be performed. If the sampled pages are empty, the table statistics will be inaccurate. Pages become empty after a large number of changes to the table, for example, deleting a large number of rows. A \texttt{VACUUM FULL} operation removes the empty pages and allows an \texttt{ANALYZE} operation to collect accurate statistics.

If there are no statistics for the table, the server configuration parameter \texttt{gp_enable_relsize_collection} controls whether the legacy query optimizer uses a default statistics file or estimates the size of a table using the \texttt{pg_relation_size} function. By default, the legacy optimizer uses the default statistics file to estimate the number of rows if statistics are not available.

### Examples

Collect statistics for the table \texttt{mytable}:

```
ANALYZE mytable;
```

### Compatibility

There is no \texttt{ANALYZE} statement in the SQL standard.

### See Also

\texttt{ALTER TABLE}, \texttt{EXPLAIN}, \texttt{VACUUM}, \texttt{analyzedb} utility in the \textit{Greenplum Database Utility Guide}.

**BEGIN**

Starts a transaction block.

### Synopsis

```
BEGIN [WORK | TRANSACTION] [transaction_mode]
   [READ ONLY | READ WRITE]
```

where \texttt{transaction_mode} is one of:

- \texttt{ISOLATION LEVEL} | \{\texttt{SERIALIZABLE} | \texttt{READ COMMITTED} | \texttt{READ UNCOMMITTED}\}

### Description

\texttt{BEGIN} initiates a transaction block, that is, all statements after a \texttt{BEGIN} command will be executed in a single transaction until an explicit \texttt{COMMIT} or \texttt{ROLLBACK} is given. By default (without \texttt{BEGIN}), Greenplum Database executes transactions in autocommit mode, that is, each statement is executed in its own transaction and a commit is implicitly performed at the end of the statement (if execution was successful, otherwise a rollback is done).

Statements are executed more quickly in a transaction block, because transaction start/commit requires significant CPU and disk activity. Execution of multiple statements inside a transaction is also useful to ensure consistency when making several related changes: other sessions will be unable to see the intermediate states wherein not all the related updates have been done.

If the isolation level or read/write mode is specified, the new transaction has those characteristics, as if \texttt{SET TRANSACTION} was executed.
Parameters
WORK
TRANSACTION
Optional key words. They have no effect.

SERIALIZABLE
READ COMMITTED
READ UNCOMMITTED
The SQL standard defines four transaction isolation levels: READ COMMITTED, READ UNCOMMITTED, SERIALIZABLE, and REPEATABLE READ. The default behavior is that a statement can only see rows committed before it began (READ COMMITTED). In Greenplum Database READ UNCOMMITTED is treated the same as READ COMMITTED. REPEATABLE READ is not supported; use SERIALIZABLE if this behavior is required. SERIALIZABLE is the strictest transaction isolation. This level emulates serial transaction execution, as if transactions had been executed one after another, serially, rather than concurrently. Applications using this level must be prepared to retry transactions due to serialization failures.

READ WRITE
READ ONLY
Determines whether the transaction is read/write or read-only. Read/write is the default. When a transaction is read-only, the following SQL commands are disallowed: INSERT, UPDATE, DELETE, and COPY FROM if the table they would write to is not a temporary table; all CREATE, ALTER, and DROP commands; GRANT, REVOKE, TRUNCATE; and EXPLAIN ANALYZE and EXECUTE if the command they would execute is among those listed.

Notes
START TRANSACTION has the same functionality as BEGIN.
Use COMMIT or ROLLBACK to terminate a transaction block.
Issuing BEGIN when already inside a transaction block will provoke a warning message. The state of the transaction is not affected. To nest transactions within a transaction block, use savepoints (see SAVEPOINT).

Examples
To begin a transaction block:

BEGIN;

To begin a transaction block with the serializable isolation level:

BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Compatibility
BEGIN is a Greenplum Database language extension. It is equivalent to the SQL-standard command START TRANSACTION.
Incidently, the BEGIN key word is used for a different purpose in embedded SQL. You are advised to be careful about the transaction semantics when porting database applications.

See Also
COMMIT, ROLLBACK, START TRANSACTION, SAVEPOINT
**CHECKPOINT**

Forces a transaction log checkpoint.

**Synopsis**

CHECKPOINT

**Description**

Write-Ahead Logging (WAL) puts a checkpoint in the transaction log every so often. The automatic checkpoint interval is set per Greenplum Database segment instance by the server configuration parameters `checkpoint_segments` and `checkpoint_timeout`. The **CHECKPOINT** command forces an immediate checkpoint when the command is issued, without waiting for a scheduled checkpoint.

A checkpoint is a point in the transaction log sequence at which all data files have been updated to reflect the information in the log. All data files will be flushed to disk.

Only superusers may call **CHECKPOINT**. The command is not intended for use during normal operation.

**Compatibility**

The **CHECKPOINT** command is a Greenplum Database language extension.

**CLOSE**

Closes a cursor.

**Synopsis**

CLOSE cursor_name

**Description**

**CLOSE** frees the resources associated with an open cursor. After the cursor is closed, no subsequent operations are allowed on it. A cursor should be closed when it is no longer needed.

Every non-holdable open cursor is implicitly closed when a transaction is terminated by **COMMIT** or **ROLLBACK**. A holdable cursor is implicitly closed if the transaction that created it aborts via **ROLLBACK**. If the creating transaction successfully commits, the holdable cursor remains open until an explicit **CLOSE** is executed, or the client disconnects.

**Parameters**

cursor_name

The name of an open cursor to close.

**Notes**

Greenplum Database does not have an explicit **OPEN** cursor statement. A cursor is considered open when it is declared. Use the **DECLARE** statement to declare (and open) a cursor.

You can see all available cursors by querying the `pg_cursors` system view.
Examples
Close the cursor portala:

CLOSE portala;

Compatibility
CLOSE is fully conforming with the SQL standard.

See Also
DECLARE, FETCH, MOVE

CLUSTER
Physically reorders a heap storage table on disk according to an index. Not a recommended operation in Greenplum Database.

Synopsis

```
CLUSTER indexname ON tablename
CLUSTER tablename
CLUSTER
```

Description
CLUSTER orders a heap storage table based on an index. CLUSTER is not supported on append-optimized storage tables. Clustering an index means that the records are physically ordered on disk according to the index information. If the records you need are distributed randomly on disk, then the database has to seek across the disk to get the records requested. If those records are stored more closely together, then the fetching from disk is more sequential. A good example for a clustered index is on a date column where the data is ordered sequentially by date. A query against a specific date range will result in an ordered fetch from the disk, which leverages faster sequential access.

Clustering is a one-time operation: when the table is subsequently updated, the changes are not clustered. That is, no attempt is made to store new or updated rows according to their index order. If one wishes, one can periodically recluster by issuing the command again.

When a table is clustered using this command, Greenplum Database remembers on which index it was clustered. The form CLUSTER tablename reclusters the table on the same index that it was clustered before. CLUSTER without any parameter reclusters all previously clustered tables in the current database that the calling user owns, or all tables if called by a superuser. This form of CLUSTER cannot be executed inside a transaction block.

When a table is being clustered, an ACCESS EXCLUSIVE lock is acquired on it. This prevents any other database operations (both reads and writes) from operating on the table until the CLUSTER is finished.

Parameters

- **indexname**
  - The name of an index.

- **tablename**
  - The name (optionally schema-qualified) of a table.
Notes

In cases where you are accessing single rows randomly within a table, the actual order of the data in
the table is unimportant. However, if you tend to access some data more than others, and there is an
index that groups them together, you will benefit from using CLUSTER. If you are requesting a range of
indexed values from a table, or a single indexed value that has multiple rows that match, CLUSTER will help
because once the index identifies the table page for the first row that matches, all other rows that match
are probably already on the same table page, and so you save disk accesses and speed up the query.

During the cluster operation, a temporary copy of the table is created that contains the table data in
the index order. Temporary copies of each index on the table are created as well. Therefore, you need free
space on disk at least equal to the sum of the table size and the index sizes.

Because the query optimizer records statistics about the ordering of tables, it is advisable to run ANALYZE
on the newly clustered table. Otherwise, the planner may make poor choices of query plans.

There is another way to cluster data. The CLUSTER command reorders the original table by scanning it
using the index you specify. This can be slow on large tables because the rows are fetched from the table
in index order, and if the table is disordered, the entries are on random pages, so there is one disk page
retrieved for every row moved. (Greenplum Database has a cache, but the majority of a big table will not fit
in the cache.) The other way to cluster a table is to use a statement such as:

```
CREATE TABLE newtable AS SELECT * FROM table ORDER BY column;
```

This uses the Greenplum Database sorting code to produce the desired order, which is usually much faster
than an index scan for disordered data. Then you drop the old table, use ALTER TABLE ... RENAME to
rename newtable to the old name, and recreate the table's indexes. The big disadvantage of this approach
is that it does not preserve OIDs, constraints, granted privileges, and other ancillary properties of the
table — all such items must be manually recreated. Another disadvantage is that this way requires a sort
temporary file about the same size as the table itself, so peak disk usage is about three times the table size
instead of twice the table size.

Note: CLUSTER is not supported with append-optimized tables.

Examples

Cluster the table employees on the basis of its index emp_ind:

```
CLUSTER emp_ind ON emp;
```

Cluster a large table by recreating it and loading it in the correct index order:

```
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;
```

Compatibility

There is no CLUSTER statement in the SQL standard.

See Also

CREATE TABLE AS, CREATE INDEX

COMMENT

Defines or change the comment of an object.
Synopsis

```
COMMENT ON
{ TABLE object_name |
  COLUMN table_name.column_name |
  AGGREGATE agg_name (agg_type [, ...]) |
  CAST (sourcetype AS targettype) |
  CONSTRAINT constraint_name ON table_name |
  CONVERSION object_name |
  DATABASE object_name |
  DOMAIN object_name |
  FILESPACE object_name |
  FUNCTION func_name (argmode argname argtype [, ...]) |
  INDEX object_name |
  LARGE OBJECT large_object_oid |
  OPERATOR op (leftoperand_type, rightoperand_type) |
  OPERATOR CLASS object_name USING index_method |
  [PROCEDURAL] LANGUAGE object_name |
  RESOURCE QUEUE object_name |
  ROLE object_name |
  RULE rule_name ON table_name |
  SCHEMA object_name |
  SEQUENCE object_name |
  TABLESPACE object_name |
  TRIGGER trigger_name ON table_name |
  TYPE object_name |
  VIEW object_name }
IS 'text'
```

Description

`COMMENT` stores a comment about a database object. To modify a comment, issue a new `COMMENT` command for the same object. Only one comment string is stored for each object. To remove a comment, write `NULL` in place of the text string. Comments are automatically dropped when the object is dropped.

Comments can be easily retrieved with the `psql` meta-commands `\dd`, `\d+`, and `\l+`. Other user interfaces to retrieve comments can be built atop the same built-in functions that `psql` uses, namely `obj_description`, `col_description`, and `shobj_description`.

Parameters

- `object_name`
- `table_name.column_name`
- `agg_name`
- `constraint_name`
- `func_name`
- `op`
- `rule_name`
- `trigger_name`

  The name of the object to be commented. Names of tables, aggregates, domains, functions, indexes, operators, operator classes, sequences, types, and views may be schema-qualified.

  **Note:** Greenplum Database does not support triggers.

- `agg_type`

  An input data type on which the aggregate function operates. To reference a zero-argument aggregate function, write `*` in place of the list of input data types.

- `sourcetype`

  The name of the source data type of the cast.
**targettype**

The name of the target data type of the cast.

**argmode**

The mode of a function argument: either IN, OUT, INOUT, or VARIADIC. If omitted, the default is IN. Note that COMMENT ON FUNCTION does not actually pay any attention to OUT arguments, since only the input arguments are needed to determine the function’s identity. So it is sufficient to list the IN, INOUT, and VARIADIC arguments.

**argname**

The name of a function argument. Note that COMMENT ON FUNCTION does not actually pay any attention to argument names, since only the argument data types are needed to determine the function’s identity.

**argtype**

The data type(s) of the function’s arguments (optionally schema-qualified), if any.

**large_object_oid**

The OID of the large object.

**PROCEDURAL**

This is a noise word.

**text**

The new comment, written as a string literal; or NULL to drop the comment.

**Notes**

There is presently no security mechanism for comments: any user connected to a database can see all the comments for objects in that database (although only superusers can change comments for objects that they do not own). For shared objects such as databases, roles, and tablespaces comments are stored globally and any user connected to any database can see all the comments for shared objects. Therefore, do not put security-critical information in comments.

**Examples**

Attach a comment to the table mytable:

```sql
COMMENT ON TABLE mytable IS 'This is my table.';
```

Remove it again:

```sql
COMMENT ON TABLE mytable IS NULL;
```

**Compatibility**

There is no COMMENT statement in the SQL standard.

**COMMIT**

Commits the current transaction.

**Synopsis**

```sql
COMMIT [WORK | TRANSACTION]
```
Description

COMMIT commits the current transaction. All changes made by the transaction become visible to others and are guaranteed to be durable if a crash occurs.

Parameters

<table>
<thead>
<tr>
<th>WORK</th>
<th>TRANSACTION</th>
</tr>
</thead>
</table>

Optional key words. They have no effect.

Notes

Use ROLLBACK to abort a transaction.

Issuing COMMIT when not inside a transaction does no harm, but it will provoke a warning message.

Examples

To commit the current transaction and make all changes permanent:

```sql
COMMIT;
```

Compatibility

The SQL standard only specifies the two forms COMMIT and COMMIT WORK. Otherwise, this command is fully conforming.

See Also

BEGIN, END, START TRANSACTION, ROLLBACK

COPY

Copies data between a file and a table.

Synopsis

```sql
COPY table [(column [, ...])] FROM ('file' | PROGRAM 'command' | STDIN) [
  [WITH]
  [ON SEGMENT]
  [BINARY]
  [OIDS]
  [HEADER]
  [DELIMITER [ AS ] 'delimiter']
  [NULL [ AS ] 'null string']
  [ESCAPE [ AS ] 'escape' | 'OFF']
  [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
  [CSV [QUOTE [ AS ] 'quote']
    [FORCE NOT NULL column [, ...]]
  [FILL MISSING FIELDS]
  [[LOG ERRORS]
   SEGMENT REJECT LIMIT count [ROWS | PERCENT] ]
] TO {'file' | PROGRAM 'command' | STDOUT}
  [
  [WITH]
  [ON SEGMENT]
  [BINARY]
  [OIDS]
```
COPY moves data between Greenplum Database tables and standard file-system files. COPY TO copies the contents of a table to a file (or multiple files based on the segment ID if copying ON SEGMENT), while COPY FROM copies data from a file to a table (appending the data to whatever is in the table already). COPY TO can also copy the results of a SELECT query.

If a list of columns is specified, COPY will only copy the data in the specified columns to or from the file. If there are any columns in the table that are not in the column list, COPY FROM will insert the default values for those columns.

COPY with a file name instructs the Greenplum Database master host to directly read from or write to a file. The file must be accessible to the master host and the name must be specified from the viewpoint of the master host.

When COPY is used with the ON SEGMENT clause, the COPY TO causes segments to create individual segment-oriented files, which remain on the segment hosts. The file argument for ON SEGMENT takes the string literal <SEGID> (required) and uses either the absolute path or the <SEG_DATA_DIR> string literal. When the COPY operation is run, the segment IDs and the paths of the segment data directories are substituted for the string literal values.

The ON SEGMENT clause allows you to copy table data to files on segment hosts for use in operations such as migrating data between clusters or performing a backup. Segment data created by the ON SEGMENT clause can be restored by tools such as gpfdist, which is useful for high speed data loading.

Warning: Use of the ON SEGMENT clause is recommended for expert users only.

When STDIN or STDOUT is specified, data is transmitted via the connection between the client and the master. STDIN and STDOUT cannot be used with the ON SEGMENT clause.

If SEGMENT REJECT LIMIT is used, then a COPY FROM operation will operate in single row error isolation mode. In this release, single row error isolation mode only applies to rows in the input file with format errors — for example, extra or missing attributes, attributes of a wrong data type, or invalid client encoding sequences. Constraint errors such as violation of a NOT NULL, CHECK, or UNIQUE constraint will still be handled in 'all-or-nothing' input mode. The user can specify the number of error rows acceptable (on a per-segment basis), after which the entire COPY FROM operation will be aborted and no rows will be loaded. The count of error rows is per-segment, not per entire load operation. If the per-segment reject limit is not reached, then all rows not containing an error will be loaded and any error rows discarded. To keep error rows for further examination, specify the LOG ERRORS clause to capture error log information. The error information and the row is stored internally in Greenplum Database.

Outputs

On successful completion, a COPY command returns a command tag of the form, where count is the number of rows copied:

COPY count

If running a COPY FROM command in single row error isolation mode, the following notice message will be returned if any rows were not loaded due to format errors, where count is the number of rows rejected:

NOTICE: Rejected count badly formatted rows.
Parameters

table
The name (optionally schema-qualified) of an existing table.

column
An optional list of columns to be copied. If no column list is specified, all columns of the table will be copied.

When copying in text format, the default, a row of data in a column of type `bytea` can be up to 256MB.

query
A `SELECT` or `VALUES` command whose results are to be copied. Note that parentheses are required around the query.

file
The absolute path name of the input or output file.

PROGRAM 'command'
Specify a command to execute. The `command` must be specified from the viewpoint of the Greenplum Database master host system, and must be executable by the Greenplum Database administrator user (`gpadmin`). The `COPY FROM` command reads the input from the standard output of the command, and for the `COPY TO` command, the output is written to the standard input of the command.

The `command` is invoked by a shell. When passing arguments to the shell, strip or escape any special characters that have a special meaning for the shell. For security reasons, it is best to use a fixed command string, or at least avoid passing any user input in the string.

When `ON SEGMENT` is specified, the command must be executable on all Greenplum Database primary segment hosts by the Greenplum Database administrator user (`gpadmin`). The command is executed by each Greenplum segment instance. The `<SEGID>` is required in the `command`.

See the ON SEGMENT clause for information about command syntax requirements and the data that is copied when the clause is specified.

STDIN
Specifies that input comes from the client application. The `ON SEGMENT` clause is not supported with `STDIN`.

STDOUT
Specifies that output goes to the client application. The `ON SEGMENT` clause is not supported with `STDOUT`.

ON SEGMENT
Specify individual, segment data files on the segment hosts. Each file contains the table data that is managed by the primary segment instance. For example, when copying data to files from a table with a `COPY TO...ON SEGMENT` command, the command creates a file on the segment host for each segment instance on the host. Each file contains the table data that is managed by the segment instance.

The `COPY` command does not copy data from or to mirror segment instances and segment data files.

The keywords `STDIN` and `STDOUT` are not supported with `ON SEGMENT`.

The `<SEG_DATA_DIR>` and `<SEGID>` string literals are used to specify an absolute path and file name with the following syntax:

```
COPY table [TO|FROM] 'SEG_DATA_DIR'/gpdumpname<SEGID>_suffix' ON SEGMENT;
```
The string literal representing the absolute path of the segment instance data directory for ON SEGMENT copying. The angle brackets (< and >) are part of the string literal used to specify the path. COPY replaces the string literal with the segment path(s) when COPY is run. An absolute path can be used in place of the <SEG_DATA_DIR> string literal.

The string literal representing the content ID number of the segment instance to be copied when copying ON SEGMENT. <SEGID> is a required part of the file name when ON SEGMENT is specified. The angle brackets are part of the string literal used to specify the file name.

With COPY TO, the string literal is replaced by the content ID of the segment instance when the COPY command is run.

With COPY FROM, specify the segment instance content ID in the name of the file and place that file on the segment instance host. There must be a file for each primary segment instance on each host. When the COPY FROM command is run, the data is copied from the file to the segment instance.

When the PROGRAM command clause is specified, the <SEGID> string literal is required in the command, the <SEG_DATA_DIR> string literal is optional. See Examples.

For a COPY FROM...ON SEGMENT command, the table distribution policy is checked when data is copied into the table. By default, an error is returned if a data row violates the table distribution policy. You can disable the distribution policy check with the server configuration parameter gp_enable_segment_copy_checking. See Notes.

Causes all data to be stored or read in binary format rather than as text. You cannot specify the DELIMITER, NULL, or CSV options in binary mode. See Binary Format.

When copying in binary format, a row of data can be up to 1GB.

Specifies copying the OID for each row. (An error is raised if OIDS is specified for a table that does not have OIDs, or in the case of copying a query.)

delimiter

The single ASCII character that separates columns within each row (line) of the file. The default is a tab character in text mode, a comma in CSV mode.

null string

The string that represents a null value. The default is \N (backslash-N) in text mode, and a empty value with no quotes in CSV mode. You might prefer an empty string even in text mode for cases where you don't want to distinguish nulls from empty strings. When using COPY FROM, any data item that matches this string will be stored as a null value, so you should make sure that you use the same string as you used with COPY TO.

escape

Specifies the single character that is used for C escape sequences (such as \n,\t,\100, and so on) and for quoting data characters that might otherwise be taken as row or column delimiters. Make sure to choose an escape character that is not used anywhere in your actual column data. The default escape character is \ (backslash) for text files or " (double quote) for CSV files, however it is possible to specify any other character to represent an escape. It is also possible to disable escaping on text-formatted files by specifying the value 'OFF' as the escape value. This is very useful for data such as web log data that has many embedded backslashes that are not intended to be escapes.

NEWLINE
Specifies the newline used in your data files — LF (Line feed, 0x0A), CR (Carriage return, 0x0D), or CRLF (Carriage return plus line feed, 0x0D 0x0A). If not specified, a Greenplum Database segment will detect the newline type by looking at the first row of data it receives and using the first newline type encountered.

**CSV**

Selects Comma Separated Value (CSV) mode. See [CSV Format](#).

**HEADER**

Specifies that a file contains a header line with the names of each column in the file. On output, the first line contains the column names from the table, and on input, the first line is ignored.

**quote**

Specifies the quotation character in CSV mode. The default is double-quote.

**FORCE QUOTE**

In CSV COPY TO mode, forces quoting to be used for all non-NULL values in each specified column. NULL output is never quoted.

**FORCE NOT NULL**

In CSV COPY FROM mode, process each specified column as though it were quoted and hence not a NULL value. For the default null string in CSV mode (nothing between two delimiters), this causes missing values to be evaluated as zero-length strings.

**FILL MISSING FIELDS**

In COPY FROM more for both TEXT and CSV, specifying FILL MISSING FIELDS will set missing trailing field values to NULL (instead of reporting an error) when a row of data has missing data fields at the end of a line or row. Blank rows, fields with a NOT NULL constraint, and trailing delimiters on a line will still report an error.

**LOG ERRORS**

This is an optional clause that can precede a SEGMENT REJECT LIMIT clause to capture error log information about rows with formatting errors.

Error log information is stored internally and is accessed with the Greenplum Database built-in SQL function `gp_read_error_log()`.

See [Notes](#) for information about the error log information and built-in functions for viewing and managing error log information.

**SEGMENT REJECT LIMIT count [ROWS | PERCENT]**

Runs a COPY FROM operation in single row error isolation mode. If the input rows have format errors they will be discarded provided that the reject limit count is not reached on any Greenplum Database segment instance during the load operation. The reject limit count can be specified as number of rows (the default) or percentage of total rows (1-100). If PERCENT is used, each segment starts calculating the bad row percentage only after the number of rows specified by the parameter `gp_reject_percent_threshold` has been processed. The default for `gp_reject_percent_threshold` is 300 rows. Constraint errors such as violation of a NOT NULL, CHECK, or UNIQUE constraint will still be handled in ‘all-or-nothing’ input mode. If the limit is not reached, all good rows will be loaded and any error rows discarded.

**Note:** Greenplum Database limits the initial number of rows that can contain formatting errors if the SEGMENT REJECT LIMIT is not triggered first or is not specified. If the first 1000 rows are rejected, the COPY operation is stopped and rolled back.

The limit for the number of initial rejected rows can be changed with the Greenplum Database server configuration parameter
gp_initial_bad_row_limit. See Server Configuration Parameters for information about the parameter.

**IGNORE EXTERNAL PARTITIONS**

When copying data from partitioned tables, data are not copied from leaf child partitions that are external tables. A message is added to the log file when data are not copied.

If this clause is not specified and Greenplum Database attempts to copy data from a leaf child partition that is an external table, an error is returned.

See the next section “Notes” for information about specifying an SQL query to copy data from leaf child partitions that are external tables.

**Notes**

COPY can only be used with tables, not with external tables or views. However, you can write COPY (SELECT * FROM viewname) TO ...

When the ON SEGMENT clause is specified, the COPY command does not support specifying a SELECT statement in the COPY TO command. For example, this command is not supported.

COPY (SELECT * FROM testtbl) TO '/tmp/mytst<SEGID>' ON SEGMENT

To copy data from a partitioned table with a leaf child partition that is an external table, use an SQL query to copy the data. For example, if the table my_sales contains a with a leaf child partition that is an external table, this command COPY my_sales TO stdout returns an error. This command sends the data to stdout:

COPY (SELECT * from my_sales ) TO stdout

The BINARY key word causes all data to be stored/read as binary format rather than as text. It is somewhat faster than the normal text mode, but a binary-format file is less portable across machine architectures and Greenplum Database versions. Also, you cannot run COPY FROM in single row error isolation mode if the data is in binary format.

You must have SELECT privilege on the table whose values are read by COPY TO, and insert privilege on the table into which values are inserted by COPY FROM.

Files named in a COPY command are read or written directly by the database server, not by the client application. Therefore, they must reside on or be accessible to the Greenplum Database master host machine, not the client. They must be accessible to and readable or writable by the Greenplum Database system user (the user ID the server runs as), not the client. COPY naming a file is only allowed to database superusers, since it allows reading or writing any file that the server has privileges to access.

COPY FROM will invoke any triggers and check constraints on the destination table. However, it will not invoke rewrite rules. Note that in this release, violations of constraints are not evaluated for single row error isolation mode.

COPY input and output is affected by DateStyle. To ensure portability to other Greenplum Database installations that might use non-default DateStyle settings, DateStyle should be set to ISO before using COPY TO.

When copying XML data from a file in text mode, the server configuration parameter xmloption affects the validation of the XML data that is copied. If the value is content (the default), XML data is validated as an XML content fragment. If the parameter value is document, XML data is validated as an XML document. If the XML data is not valid, COPY returns an error.

By default, COPY stops operation at the first error. This should not lead to problems in the event of a COPY TO, but the target table will already have received earlier rows in a COPY FROM. These rows will not be visible or accessible, but they still occupy disk space. This may amount to a considerable amount of wasted disk space if the failure happened well into a large COPY FROM operation. You may wish to invoke
VACUUM to recover the wasted space. Another option would be to use single row error isolation mode to filter out error rows while still loading good rows.

When a COPY FROM...ON SEGMENT command is run, the server configuration parameter gp_enable_segment_copy_checking controls whether the table distribution policy (from the table DISTRIBUTED clause) is checked when data is copied into the table. The default is to check the distribution policy. An error is returned if the row of data violates the distribution policy for the segment instance. For a partitioned table, if the distribution policy of the child leaf partitioned table is not the same as the root table, an error is returned for all data. For information about the parameter, see Server Configuration Parameters.

Data from a table that is generated by a COPY TO...ON SEGMENT command can be used to restore table data with COPY FROM...ON SEGMENT. However, data restored to the segments is distributed according to the table distribution policy at the time the files were generated with the COPY TO command. The COPY command might return table distribution policy errors, if you attempt to restore table data and the table distribution policy was changed after the COPY FROM...ON SEGMENT was run.

Note: If you run COPY FROM...ON SEGMENT and the server configuration parameter gp_enable_segment_copy_checking is false, manual redistribution of table data might be required. See the ALTER TABLE clause WITH REORGANIZE.

When you specify the LOG ERRORS clause, Greenplum Database captures errors that occur while reading the external table data. You can view and manage the captured error log data.

• Use the built-in SQL function gp_read_error_log('table_name'). It requires SELECT privilege on table_name. This example displays the error log information for data loaded into table ext_expenses with a COPY command:

```sql
SELECT * from gp_read_error_log('ext_expenses');
```

For information about the error log format, see Viewing Bad Rows in the Error Log in the Greenplum Database Administrator Guide.

The function returns FALSE if table_name does not exist.

• If error log data exists for the specified table, the new error log data is appended to existing error log data. The error log information is not replicated to mirror segments.

• Use the built-in SQL function gp_truncate_error_log('table_name') to delete the error log data for table_name. It requires the table owner privilege. This example deletes the error log information captured when moving data into the table ext_expenses:

```sql
SELECT gp_truncate_error_log('ext_expenses');
```

The function returns FALSE if table_name does not exist.

Specify the * wildcard character to delete error log information for existing tables in the current database. Specify the string *. * to delete all database error log information, including error log information that was not deleted due to previous database issues. If * is specified, database owner privilege is required. If *. * is specified, operating system super-user privilege is required.

When a Greenplum Database user who is not a superuser runs a COPY command, the command can be controlled by a resource queue. The resource queue must be configured with the ACTIVE_STATEMENTS parameter that specifies a maximum limit on the number of queries that can be executed by roles assigned to that queue. Greenplum Database does not apply a cost value or memory value to a COPY command, resource queues with only cost or memory limits do not affect the running of COPY commands.

A non-superuser can run only these types of COPY commands:

• COPY FROM command where the source is stdin
• COPY TO command where the destination is stdout
For information about resource queues, see “Resource Management with Resource Queues” in the Greenplum Database Administrator Guide.

File Formats
File formats supported by COPY.

Text Format
When COPY is used without the BINARY or CSV options, the data read or written is a text file with one line per table row. Columns in a row are separated by the delimiter character (tab by default). The column values themselves are strings generated by the output function, or acceptable to the input function, of each attribute’s data type. The specified null string is used in place of columns that are null. COPY FROM will raise an error if any line of the input file contains more or fewer columns than are expected. If OIDS is specified, the OID is read or written as the first column, preceding the user data columns.

The data file has two reserved characters that have special meaning to COPY:
- The designated delimiter character (tab by default), which is used to separate fields in the data file.
- A UNIX-style line feed (\n or 0x0a), which is used to designate a new row in the data file. It is strongly recommended that applications generating COPY data convert data line feeds to UNIX-style line feeds rather than Microsoft Windows style carriage return line feeds (\r\n or 0x0a 0x0d).

If your data contains either of these characters, you must escape the character so COPY treats it as data and not as a field separator or new row.

By default, the escape character is a \ (backslash) for text-formatted files and a " (double quote) for csv-formatted files. If you want to use a different escape character, you can do so using the ESCAPE AS clause. Make sure to choose an escape character that is not used anywhere in your data file as an actual data value. You can also disable escaping in text-formatted files by using ESCAPE 'OFF'.

For example, suppose you have a table with three columns and you want to load the following three fields using COPY.
- percentage sign = %
- vertical bar = |
- backslash = \n
Your designated delimiter character is | (pipe character), and your designated escape character is * (asterisk). The formatted row in your data file would look like this:

```
percentage sign = % | vertical bar = *| | backslash = \n```

Notice how the pipe character that is part of the data has been escaped using the asterisk character (*). Also notice that we do not need to escape the backslash since we are using an alternative escape character.

The following characters must be preceded by the escape character if they appear as part of a column value: the escape character itself, newline, carriage return, and the current delimiter character. You can specify a different escape character using the ESCAPE AS clause.

CSV Format
This format is used for importing and exporting the Comma Separated Value (CSV) file format used by many other programs, such as spreadsheets. Instead of the escaping used by Greenplum Database standard text mode, it produces and recognizes the common CSV escaping mechanism.

The values in each record are separated by the DELIMITER character. If the value contains the delimiter character, the QUOTE character, the ESCAPE character (which is double quote by default), the NULL string, a carriage return, or line feed character, then the whole value is prefixed and suffixed by the QUOTE character. You can also use FORCE QUOTE to force quotes when outputting non-NULL values in specific columns.
The CSV format has no standard way to distinguish a NULL value from an empty string. Greenplum Database COPY handles this by quoting. A NULL is output as the NULL string and is not quoted, while a data value matching the NULL string is quoted. Therefore, using the default settings, a NULL is written as an unquoted empty string, while an empty string is written with double quotes ("""). Reading values follows similar rules. You can use FORCE NOT NULL to prevent NULL input comparisons for specific columns.

Because backslash is not a special character in the CSV format, \
, the end-of-data marker, could also appear as a data value. To avoid any misinterpretation, a \
 data value appearing as a lone entry on a line is automatically quoted on output, and on input, if quoted, is not interpreted as the end-of-data marker. If you are loading a file created by another application that has a single unquoted column and might have a value of \
, you might need to quote that value in the input file.

**Note:** In CSV mode, all characters are significant. A quoted value surrounded by white space, or any characters other than DELIMITER, will include those characters. This can cause errors if you import data from a system that pads CSV lines with white space out to some fixed width. If such a situation arises you might need to preprocess the CSV file to remove the trailing white space, before importing the data into Greenplum Database.

CSV mode will both recognize and produce CSV files with quoted values containing embedded carriage returns and line feeds. Thus the files are not strictly one line per table row like text-mode files.

**Note:** Many programs produce strange and occasionally perverse CSV files, so the file format is more a convention than a standard. Thus you might encounter some files that cannot be imported using this mechanism, and COPY might produce files that other programs cannot process.

**Binary Format**

The BINARY format consists of a file header, zero or more tuples containing the row data, and a file trailer. Headers and data are in network byte order.

- **File Header** — The file header consists of 15 bytes of fixed fields, followed by a variable-length header extension area. The fixed fields are:
  - **Signature** — 11-byte sequence PGCOPY\n\377\r\n0 — note that the zero byte is a required part of the signature. (The signature is designed to allow easy identification of files that have been munged by a non-8-bit-clean transfer. This signature will be changed by end-of-line-translation filters, dropped zero bytes, dropped high bits, or parity changes.)
  - **Flags field** — 32-bit integer bit mask to denote important aspects of the file format. Bits are numbered from 0 (LSB) to 31 (MSB). Note that this field is stored in network byte order (most significant byte first), as are all the integer fields used in the file format. Bits 16-31 are reserved to denote critical file format issues; a reader should abort if it finds an unexpected bit set in this range. Bits 0-15 are reserved to signal backwards-compatible format issues; a reader should simply ignore any unexpected bits set in this range. Currently only one flag is defined, and the rest must be zero (Bit 16: 1 if data has OIDs, 0 if not).
  - **Header extension area length** — 32-bit integer, length in bytes of remainder of header, not including self. Currently, this is zero, and the first tuple follows immediately. Future changes to the format might allow additional data to be present in the header. A reader should silently skip over any header extension data it does not know what to do with. The header extension area is envisioned to contain a sequence of self-identifying chunks. The flags field is not intended to tell readers what is in the extension area. Specific design of header extension contents is left for a later release.
  - **Tuples** — Each tuple begins with a 16-bit integer count of the number of fields in the tuple. (Presently, all tuples in a table will have the same count, but that might not always be true.) Then, repeated for each field in the tuple, there is a 32-bit length word followed by that many bytes of field data. (The length word does not include itself, and can be zero.) As a special case, -1 indicates a NULL field value. No value bytes follow in the NULL case.

There is no alignment padding or any other extra data between fields.
Presently, all data values in a COPY BINARY file are assumed to be in binary format (format code one). It is anticipated that a future extension may add a header field that allows per-column format codes to be specified.

If OIDs are included in the file, the OID field immediately follows the field-count word. It is a normal field except that it is not included in the field-count. In particular it has a length word — this will allow handling of 4-byte vs. 8-byte OIDs without too much pain, and will allow OIDs to be shown as null if that ever proves desirable.

- **File Trailer** — The file trailer consists of a 16-bit integer word containing −1. This is easily distinguished from a tuple's field-count word. A reader should report an error if a field-count word is neither −1 nor the expected number of columns. This provides an extra check against somehow getting out of sync with the data.

**Examples**

Copy a table to the client using the vertical bar (|) as the field delimiter:

```sql
COPY country TO STDOUT WITH DELIMITER '|';
```

Copy data from a file into the `country` table:

```sql
COPY country FROM '/home/usr1/sql/country_data';
```

Copy into a file just the countries whose names start with 'A':

```sql
COPY (SELECT * FROM country WHERE country_name LIKE 'A%') TO
'/home/usr1/sql/a_list_countries.copy';
```

Copy data from a file into the `sales` table using single row error isolation mode and log errors:

```sql
COPY sales FROM '/home/usr1/sql/sales_data' LOG ERRORS
    SEGMENT REJECT LIMIT 10 ROWS;
```

To copy segment data for later use, use the `ON SEGMENT` clause. Use of the `COPY TO ON SEGMENT` command takes the form:

```sql
COPY table TO '<SEG_DATA_DIR>/gpdumpname<SEGID>_suffix' ON SEGMENT;
```

The `<SEGID>` is required. However, you can substitute an absolute path for the `<SEG_DATA_DIR>` string literal in the path.

When you pass in the string literal `<SEG DATA_DIR>` and `<SEGID>` to COPY, COPY will fill in the appropriate values when the operation is run.

For example, if you have `mytable` with the segments and mirror segments like this:

<table>
<thead>
<tr>
<th>contentid</th>
<th>dbid</th>
<th>file segment location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>/home/usr1/data1/gpsegdir0</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>/home/usr1/data_mirror1/gpsegdir0</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>/home/usr1/data2/gpsegdir1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>/home/usr1/data_mirror2/gpsegdir1</td>
</tr>
</tbody>
</table>

running the command:

```sql
COPY mytable TO '<SEG_DATA_DIR>/gpbackup<SEGID>.txt' ON SEGMENT;
```

would result in the following files:

```
/home/usr1/data1/gpsegdir0/gpbackup0.txt
```
The content ID in the first column is the identifier inserted into the file path (for example, `gpsegdir0/gpbackup0.txt`, above) Files are created on the segment hosts, rather than on the master, as they would be in a standard COPY operation. No data files are created for the mirror segments when using ON SEGMENT copying.

If an absolute path is specified, instead of `<SEG_DATA_DIR>`, such as in the statement

```
COPY mytable TO '/tmp/gpdir/gpbackup_<SEGID>.txt' ON SEGMENT;
```

files would be placed in `/tmp/gpdir` on every segment. The `gpfdist` tool can also be used to restore data files generated with `COPY` TO with the ON SEGMENT option if redistribution is necessary.

**Note:** Tools such as `gpfdist` can be used to restore data. The backup/restore tools will not work with files that were manually generated with `COPY` TO ON SEGMENT.

This example copies the data from the `lineitem` table and uses the PROGRAM clause to add the data to the `/tmp/lineitem_program.csv` file with `cat` utility. The file is placed on the Greenplum Database master.

```
COPY LINEITEM TO PROGRAM 'cat > /tmp/lineitem.csv' CSV;
```

This example uses the PROGRAM and ON SEGMENT clauses to copy data to files on the segment hosts. On the segment hosts, the `COPY` command replaces `<SEGID>` with the segment content ID to create a file for each segment instance on the segment host.

```
COPY LINEITEM TO PROGRAM 'cat > /tmp/lineitem_program<SEGID>.csv' ON SEGMENT CSV;
```

This example uses the PROGRAM and ON SEGMENT clauses to copy data from files on the segment hosts. The `COPY` command replaces `<SEGID>` with the segment content ID when copying data from the files. On the segment hosts, there must be a file for each segment instance where the file name contains the segment content ID on the segment host.

```
COPY LINEITEM_4 FROM PROGRAM 'cat /tmp/lineitem_program<SEGID>.csv' ON SEGMENT CSV;
```

**Compatibility**

There is no `COPY` statement in the SQL standard.

**See Also**

`CREATE EXTERNAL TABLE`

**CREATE AGGREGATE**

Defines a new aggregate function.

**Synopsis**

```
CREATE [ORDERED] AGGREGATE name (input_data_type [ , ... ])
  ( SFUNC = sfunc,
    STYPE = state_data_type
    [, PREFUNC = prefunc]
    [, FINALFUNC = ffunc]
    [, INITCOND = initial_condition]
```
CREATE AGGREGATE defines a new aggregate function. Some basic and commonly-used aggregate functions such as `count`, `min`, `max`, `sum`, `avg` and so on are already provided in Greenplum Database. If one defines new types or needs an aggregate function not already provided, then CREATE AGGREGATE can be used to provide the desired features.

An aggregate function is identified by its name and input data types. Two aggregate functions in the same schema can have the same name if they operate on different input types. The name and input data types of an aggregate function must also be distinct from the name and input data types of every ordinary function in the same schema.

An aggregate function is made from one, two or three ordinary functions (all of which must be IMMUTABLE functions):

- A state transition function `sfunc`
- An optional preliminary segment-level calculation function `prefunc`
- An optional final calculation function `ffunc`

These functions are used as follows:

\[
\text{sfunc( internal-state, next-data-values )} \rightarrow \text{next-internal-state} \\
\text{prefunc( internal-state, internal-state )} \rightarrow \text{next-internal-state} \\
\text{ffunc( internal-state )} \rightarrow \text{aggregate-value}
\]

You can specify `PREFUNC` as method for optimizing aggregate execution. By specifying `PREFUNC`, the aggregate can be executed in parallel on segments first and then on the master. When a two-level execution is performed, `SFUNC` is executed on the segments to generate partial aggregate results, and `PREFUNC` is executed on the master to aggregate the partial results from segments. If single-level aggregation is performed, all the rows are sent to the master and `SFUNC` is applied to the rows.

Single-level aggregation and two-level aggregation are equivalent execution strategies. Either type of aggregation can be implemented in a query plan. When you implement the functions `prefunc` and `sfunc`, you must ensure that the invocation of `sfunc` on the segment instances followed by `prefunc` on the master produce the same result as single-level aggregation that sends all the rows to the master and then applies only the `sfunc` to the rows.

Greenplum Database creates a temporary variable of data type `stype` to hold the current internal state of the aggregate function. At each input row, the aggregate argument values are calculated and the state transition function is invoked with the current state value and the new argument values to calculate a new internal state value. After all the rows have been processed, the final function is invoked once to calculate the aggregate return value. If there is no final function then the ending state value is returned as-is.

An aggregate function can provide an optional initial condition, an initial value for the internal state value. This is specified and stored in the database as a value of type text, but it must be a valid external representation of a constant of the state value data type. If it is not supplied then the state value starts out `NULL`.

If the state transition function is declared `STRICT`, then it cannot be called with `NULL` inputs. With such a transition function, aggregate execution behaves as follows. Rows with any null input values are ignored (the function is not called and the previous state value is retained). If the initial state value is `NULL`, then at the first row with all non-null input values, the first argument value replaces the state value, and the transition function is invoked at subsequent rows with all non-null input values. This is useful for implementing aggregates like `max`. Note that this behavior is only available when `state_data_type` is the same as the first `input_data_type`. When these types are different, you must supply a non-null initial condition or use a nonstrict transition function.
If the state transition function is not declared `STRICT`, then it will be called unconditionally at each input row, and must deal with `NULL` inputs and `NULL` transition values for itself. This allows the aggregate author to have full control over the aggregate handling of `NULL` values.

If the final function is declared `STRICT`, then it will not be called when the ending state value is `NULL`; instead a `NULL` result will be returned automatically. (This is the normal behavior of `STRICT` functions.) In any case the final function has the option of returning a `NULL` value. For example, the final function for `avg` returns `NULL` when it sees there were zero input rows.

Single argument aggregate functions, such as `min` or `max`, can sometimes be optimized by looking into an index instead of scanning every input row. If this aggregate can be so optimized, indicate it by specifying a sort operator. The basic requirement is that the aggregate must yield the first element in the sort ordering induced by the operator; in other words:

```sql
SELECT agg(col) FROM tab;
```

must be equivalent to:

```sql
SELECT col FROM tab ORDER BY col USING sortop LIMIT 1;
```

Further assumptions are that the aggregate function ignores `NULL` inputs, and that it delivers a `NULL` result if and only if there were no non-null inputs. Ordinarily, a data type’s `<` operator is the proper sort operator for `MIN`, and `>` is the proper sort operator for `MAX`. Note that the optimization will never actually take effect unless the specified operator is the "less than" or "greater than" strategy member of a B-tree index operator class.

**Ordered Aggregates**

If the optional qualification `ORDERED` appears, the created aggregate function is an ordered aggregate. In this case, the preliminary aggregation function, `prefunc` cannot be specified.

An ordered aggregate is called with the following syntax.

```sql
name ( arg [, ... ] [ORDER BY sortspec [, ...]])
```

If the optional `ORDER BY` is omitted, a system-defined ordering is used. The transition function `sfunc` of an ordered aggregate function is called on its input arguments in the specified order and on a single segment. There is a new column `aggordered` in the `pg_aggregate` table to indicate the aggregate function is defined as an ordered aggregate.

**Parameters**

- `name` The name (optionally schema-qualified) of the aggregate function to create.

- `input_data_type` An input data type on which this aggregate function operates. To create a zero-argument aggregate function, write `*` in place of the list of input data types. An example of such an aggregate is `count(*)`.

- `sfunc` The name of the state transition function to be called for each input row. For an N-argument aggregate function, the `sfunc` must take N+1 arguments, the first being of type `state_data_type` and the rest matching the declared input data types of the aggregate. The function must return a value of type `state_data_type`. This function takes the current state value and the current input data values, and returns the next state value.

- `state_data_type` The data type for the aggregate state value.

- `prefunc`
The name of a preliminary aggregation function. This is a function of two arguments, both of type `state_data_type`. It must return a value of `state_data_type`. A preliminary function takes two transition state values and returns a new transition state value representing the combined aggregation. In Greenplum Database, if the result of the aggregate function is computed in a segmented fashion, the preliminary aggregation function is invoked on the individual internal states in order to combine them into an ending internal state.

Note that this function is also called in hash aggregate mode within a segment. Therefore, if you call this aggregate function without a preliminary function, hash aggregate is never chosen. Since hash aggregate is efficient, consider defining preliminary function whenever possible.

`ffunc`

The name of the final function called to compute the aggregate result after all input rows have been traversed. The function must take a single argument of type `state_data_type`. The return data type of the aggregate is defined as the return type of this function. If `ffunc` is not specified, then the ending state value is used as the aggregate result, and the return type is `state_data_type`.

`initial_condition`

The initial setting for the state value. This must be a string constant in the form accepted for the data type `state_data_type`. If not specified, the state value starts out `NULL`.

`sort_operator`

The associated sort operator for a MIN- or MAX-like aggregate function. This is just an operator name (possibly schema-qualified). The operator is assumed to have the same input data types as the aggregate function (which must be a single-argument aggregate function).

**Notes**

The ordinary functions used to define a new aggregate function must be defined first. Note that in this release of Greenplum Database, it is required that the `sfunc`, `ffunc`, and `prefunc` functions used to create the aggregate are defined as `IMMUTABLE`.

If a user-defined aggregate is used in a window expression, a `prefunc` function must be defined for the aggregate.

If the value of the Greenplum Database server configuration parameter `gp_enable_multiphase_agg` is `off`, only single-level aggregation is performed.

Any compiled code (shared library files) for custom functions must be placed in the same location on every host in your Greenplum Database array (master and all segments). This location must also be in the `LD_LIBRARY_PATH` so that the server can locate the files.

**Example**

The following simple example creates an aggregate function that computes the sum of two columns.

Before creating the aggregate function, create two functions that are used as the `SFUNC` and `PREFUNC` functions of the aggregate function.

This function is specified as the `SFUNC` function in the aggregate function.

```sql
CREATE FUNCTION mysfunc_accum(numeric, numeric, numeric)
RETURNS numeric
AS 'select $1 + $2 + $3'
LANGUAGE SQL
IMMUTABLE
RETURNS NULL ON NULL INPUT;
```
This function is specified as the `PREFUNC` function in the aggregate function.

```sql
CREATE FUNCTION mypre_accum(numeric, numeric )
    RETURNS numeric
    AS 'select $1 + $2'
    LANGUAGE SQL
    IMMUTABLE
    RETURNS NULL ON NULL INPUT;
```

This `CREATE AGGREGATE` command creates the aggregate function that adds two columns.

```sql
CREATE AGGREGATE agg_prefunc(numeric, numeric) (
    SFUNC = mysfunc_accum,
    STYPE = numeric,
    PREFUNC = mypre_accum,
    INITCOND = 0 );
```

The following commands create a table, adds some rows, and runs the aggregate function.

```sql
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;
```

This `EXPLAIN` command shows two phase aggregation.

```sql
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)
```

**Compatibility**

`CREATE AGGREGATE` is a Greenplum Database language extension. The SQL standard does not provide for user-defined aggregate functions.

**See Also**

`ALTER AGGREGATE`, `DROP AGGREGATE`, `CREATE FUNCTION`

**CREATE CAST**

Defines a new cast.

**Synopsis**

```sql
CREATE CAST (sourcetype AS targettype)
    WITH FUNCTION funcname (argtypes)
    [AS ASSIGNMENT | AS IMPLICIT]
```

```sql
CREATE CAST (sourcetype AS targettype) WITHOUT FUNCTION
```
Description

CREATE CAST defines a new cast. A cast specifies how to perform a conversion between two data types. For example,

```
SELECT CAST(42 AS text);
```

converts the integer constant 42 to type text by invoking a previously specified function, in this case text(int4). If no suitable cast has been defined, the conversion fails.

Two types may be binary compatible, which means that they can be converted into one another without invoking any function. This requires that corresponding values use the same internal representation. For instance, the types text and varchar are binary compatible.

By default, a cast can be invoked only by an explicit cast request, that is an explicit CAST(x AS typename) or x:: typename construct.

If the cast is marked AS ASSIGNMENT then it can be invoked implicitly when assigning a value to a column of the target data type. For example, supposing that foo.f1 is a column of type text, then:

```
INSERT INTO foo (f1) VALUES (42);
```

will be allowed if the cast from type integer to type text is marked AS ASSIGNMENT, otherwise not. The term assignment cast is typically used to describe this kind of cast.

If the cast is marked AS IMPLICIT then it can be invoked implicitly in any context, whether assignment or internally in an expression. The term implicit cast is typically used to describe this kind of cast. For example, since || takes text operands,

```
SELECT 'The time is ' || now();
```

will be allowed only if the cast from type timestamp to text is marked AS IMPLICIT. Otherwise, it will be necessary to write the cast explicitly, for example

```
SELECT 'The time is ' || CAST(now() AS text);
```

It is wise to be conservative about marking casts as implicit. An overabundance of implicit casting paths can cause Greenplum Database to choose surprising interpretations of commands, or to be unable to resolve commands at all because there are multiple possible interpretations. A good rule of thumb is to make a cast implicitly invokable only for information-preserving transformations between types in the same general type category. For example, the cast from int2 to int4 can reasonably be implicit, but the cast from float8 to int4 should probably be assignment-only. Cross-type-category casts, such as text to int4, are best made explicit-only.

To be able to create a cast, you must own the source or the target data type. To create a binary-compatible cast, you must be superuser.

Parameters

source\text{\textit{type}}

The name of the source data type of the cast.

target\text{\textit{type}}

The name of the target data type of the cast.

\textit{funcname}\left(arg\text{\textit{types}}\right)

The function name and its arguments used in the cast.
The function used to perform the cast. The function name may be schema-qualified. If it is not, the function will be looked up in the schema search path. The function's result data type must match the target type of the cast.

Cast implementation functions may have one to three arguments. The first argument type must be identical to the cast's source type. The second argument, if present, must be type integer; it receives the type modifier associated with the destination type, or -1 if there is none. The third argument, if present, must be type boolean; it receives true if the cast is an explicit cast, false otherwise. The SQL specification demands different behaviors for explicit and implicit casts in some cases. This argument is supplied for functions that must implement such casts. It is not recommended that you design your own data types this way.

Ordinarily a cast must have different source and target data types. However, it is allowed to declare a cast with identical source and target types if it has a cast implementation function with more than one argument. This is used to represent type-specific length coercion functions in the system catalogs. The named function is used to coerce a value of the type to the type modifier value given by its second argument. (Since the grammar presently permits only certain built-in data types to have type modifiers, this feature is of no use for user-defined target types.)

When a cast has different source and target types and a function that takes more than one argument, it represents converting from one type to another and applying a length coercion in a single step. When no such entry is available, coercion to a type that uses a type modifier involves two steps, one to convert between data types and a second to apply the modifier.

**WITHOUT FUNCTION**

Indicates that the source type and the target type are binary compatible, so no function is required to perform the cast.

**AS ASSIGNMENT**

Indicates that the cast may be invoked implicitly in assignment contexts.

**AS IMPLICIT**

Indicates that the cast may be invoked implicitly in any context.

**Notes**

Note that in this release of Greenplum Database, user-defined functions used in a user-defined cast must be defined as **IMMUTABLE**. Any compiled code (shared library files) for custom functions must be placed in the same location on every host in your Greenplum Database array (master and all segments). This location must also be in the `LD_LIBRARY_PATH` so that the server can locate the files.

Remember that if you want to be able to convert types both ways you need to declare casts both ways explicitly.

It is recommended that you follow the convention of naming cast implementation functions after the target data type, as the built-in cast implementation functions are named. Many users are used to being able to cast data types using a function-style notation, that is `typename(x)`.

**Examples**

To create a cast from type `text` to type `int4` using the function `int4(text)` (This cast is already predefined in the system.):

```
CREATE CAST (text AS int4) WITH FUNCTION int4(text);
```
Compatibility

The CREATE CAST command conforms to the SQL standard, except that SQL does not make provisions for binary-compatible types or extra arguments to implementation functions. AS IMPLICIT is a Greenplum Database extension, too.

See Also

CREATE FUNCTION, CREATE TYPE, DROP CAST

CREATE CONVERSION

Defines a new encoding conversion.

Synopsis

CREATE [DEFAULT] CONVERSION name FOR source_encoding TO dest_encoding FROM funcname

Description

CREATE CONVERSION defines a new conversion between character set encodings. Conversion names may be used in the convert function to specify a particular encoding conversion. Also, conversions that are marked DEFAULT can be used for automatic encoding conversion between client and server. For this purpose, two conversions, from encoding A to B and from encoding B to A, must be defined.

To create a conversion, you must have EXECUTE privilege on the function and CREATE privilege on the destination schema.

Parameters

DEFAULT

Indicates that this conversion is the default for this particular source to destination encoding. There should be only one default encoding in a schema for the encoding pair.

name

The name of the conversion. The conversion name may be schema-qualified. If it is not, the conversion is defined in the current schema. The conversion name must be unique within a schema.

source_encoding

The source encoding name.

dest_encoding

The destination encoding name.

funcname

The function used to perform the conversion. The function name may be schema-qualified. If it is not, the function will be looked up in the path. The function must have the following signature:

```c
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;
```
Notes
Note that in this release of Greenplum Database, user-defined functions used in a user-defined conversion must be defined as IMMUTABLE. Any compiled code (shared library files) for custom functions must be placed in the same location on every host in your Greenplum Database array (master and all segments). This location must also be in the LD_LIBRARY_PATH so that the server can locate the files.

Examples
To create a conversion from encoding UTF8 to LATIN1 using myfunc:

```
CREATE CONVERSION myconv FOR 'UTF8' TO 'LATIN1' FROM myfunc;
```

Compatibility
There is no CREATE CONVERSION statement in the SQL standard.

See Also
ALTER CONVERSION, CREATE FUNCTION, DROP CONVERSION

CREATE DATABASE
Creates a new database.

Synopsis
```
CREATE DATABASE name  [ [WITH] [OWNER [=] dbowner]  
   [TEMPLATE [=] template]  
   [ENCODING [=] encoding]  
   [TABLESPACE [=] tablespace]  
   [CONNECTION LIMIT [=] connlimit ] ]
```

Description
CREATE DATABASE creates a new database. To create a database, you must be a superuser or have the special CREATEDB privilege.

The creator becomes the owner of the new database by default. Superusers can create databases owned by other users by using the OWNER clause. They can even create databases owned by users with no special privileges. Non-superusers with CREATEDB privilege can only create databases owned by themselves.

By default, the new database will be created by cloning the standard system database template1. A different template can be specified by writing TEMPLATE name. In particular, by writing TEMPLATE template0, you can create a clean database containing only the standard objects predefined by Greenplum Database. This is useful if you wish to avoid copying any installation-local objects that may have been added to template1.

Parameters

- **name**
  The name of a database to create.

- **dbowner**
  The name of the database user who will own the new database, or DEFAULT to use the default owner (the user executing the command).

- **template**
The name of the template from which to create the new database, or `DEFAULT` to use the default template (`template1`).

**encoding**

Character set encoding to use in the new database. Specify a string constant (such as `'SQL_ASCII'`), an integer encoding number, or `DEFAULT` to use the default encoding. For more information, see Character Set Support.

**tablespace**

The name of the tablespace that will be associated with the new database, or `DEFAULT` to use the template database's tablespace. This tablespace will be the default tablespace used for objects created in this database.

**conlimit**

The maximum number of concurrent connections possible. The default of -1 means there is no limitation.

**Notes**

`CREATE DATABASE` cannot be executed inside a transaction block.

When you copy a database by specifying its name as the template, no other sessions can be connected to the template database while it is being copied. New connections to the template database are locked out until `CREATE DATABASE` completes.

The `CONNECTION LIMIT` is not enforced against superusers.

**Examples**

To create a new database:

```
CREATE DATABASE gpdb;
```

To create a database `sales` owned by user `salesapp` with a default tablespace of `salesspace`:

```
CREATE DATABASE sales OWNER salesapp TABLESPACE salesspace;
```

To create a database `music` which supports the ISO-8859-1 character set:

```
CREATE DATABASE music ENCODING 'LATIN1';
```

**Compatibility**

There is no `CREATE DATABASE` statement in the SQL standard. Databases are equivalent to catalogs, whose creation is implementation-defined.

**See Also**

`ALTER DATABASE`, `DROP DATABASE`

---

**CREATE DOMAIN**

Defines a new domain.

**Synopsis**

```
CREATE DOMAIN name [AS] data_type [DEFAULT expression] [CONSTRAINT constraint_name] [NOT NULL] [NULL]
```
Description

CREATE DOMAIN creates a new domain. A domain is essentially a data type with optional constraints (restrictions on the allowed set of values). The user who defines a domain becomes its owner. The domain name must be unique among the data types and domains existing in its schema.

Domains are useful for abstracting common constraints on fields into a single location for maintenance. For example, several tables might contain email address columns, all requiring the same CHECK constraint to verify the address syntax. It is easier to define a domain rather than setting up a column constraint for each table that has an email column.

Parameters

name

The name (optionally schema-qualified) of a domain to be created.

data_type

The underlying data type of the domain. This may include array specifiers.

DEFAULT expression

Specifies a default value for columns of the domain data type. The value is any variable-free expression (but subqueries are not allowed). The data type of the default expression must match the data type of the domain. If no default value is specified, then the default value is the null value. The default expression will be used in any insert operation that does not specify a value for the column. If a default value is defined for a particular column, it overrides any default associated with the domain. In turn, the domain default overrides any default value associated with the underlying data type.

CONSTRAINT constraint_name

An optional name for a constraint. If not specified, the system generates a name.

NOT NULL

Values of this domain are not allowed to be null.

NULL

Values of this domain are allowed to be null. This is the default. This clause is only intended for compatibility with nonstandard SQL databases. Its use is discouraged in new applications.

CHECK (expression)

CHECK clauses specify integrity constraints or tests which values of the domain must satisfy. Each constraint must be an expression producing a Boolean result. It should use the key word VALUE to refer to the value being tested. Currently, CHECK expressions cannot contain subqueries nor refer to variables other than VALUE.

Examples

Create the us_zip_code data type. A regular expression test is used to verify that the value looks like a valid US zip code.

```
CREATE DOMAIN us_zip_code AS TEXT CHECK
( VALUE ~ '^\d{5}$' OR VALUE ~ '^\d{5}-\d{4}$' );
```

Compatibility

CREATE DOMAIN conforms to the SQL standard.
See Also

ALTER DOMAIN, DROP DOMAIN

CREATE EXTENSION

Registers an extension in a Greenplum database.

Synopsis

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
    [ WITH ] [ SCHEMA schema_name ]
    [ VERSION version ]
    [ FROM old_version ]
    [ CASCADE ]
```

Description

CREATE EXTENSION loads a new extension into the current database. There must not be an extension of the same name already loaded.

Loading an extension essentially amounts to running the extension script file. The script typically creates new SQL objects such as functions, data types, operators and index support methods. The CREATE EXTENSION command also records the identities of all the created objects, so that they can be dropped again if DROP EXTENSION is issued.

Loading an extension requires the same privileges that would be required to create the component extension objects. For most extensions this means superuser or database owner privileges are required. The user who runs CREATE EXTENSION becomes the owner of the extension for purposes of later privilege checks, as well as the owner of any objects created by the extension script.

Parameters

**IF NOT EXISTS**

Do not throw an error if an extension with the same name already exists. A notice is issued in this case. There is no guarantee that the existing extension is similar to the extension that would have been installed.

**extension_name**

The name of the extension to be installed. The name must be unique within the database. An extension is created from the details in the extension control file `SHAREDIR/extension/extension_name.control`.

`SHAREDIR` is the installation shared-data directory, for example `/usr/local/greenplum-db/share/postgresql`. The command `pg_config --sharedir` displays the directory.

**SCHEMA schema_name**

The name of the schema in which to install the extension objects. This assumes that the extension allows its contents to be relocated. The named schema must already exist. If not specified, and the extension control file does not specify a schema, the current default object creation schema is used.

If the extension specifies a schema parameter in its control file, then that schema cannot be overridden with a SCHEMA clause. Normally, an error is raised if a SCHEMA clause is given and it conflicts with the extension schema parameter. However, if the CASCADE clause is also given, then schema_name is ignored when it conflicts. The given schema_name is used for the installation of any needed extensions that do not a specify schema in their control files.
The extension itself is not within any schema: extensions have unqualified names that must be unique within the database. But objects belonging to the extension can be within a schema.

**VERSION version**

The version of the extension to install. This can be written as either an identifier or a string literal. The default version is value that is specified in the extension control file.

**FROM old_version**

Specify FROM old_version only if you are attempting to install an extension that replaces an old-style module that is a collection of objects that is not packaged into an extension. If specified, CREATE EXTENSION runs an alternative installation script that absorbs the existing objects into the extension, instead of creating new objects. Ensure that SCHEMA clause specifies the schema containing these pre-existing objects.

The value to use for old_version is determined by the extension author, and might vary if there is more than one version of the old-style module that can be upgraded into an extension. For the standard additional modules supplied with pre-9.1 PostgreSQL, specify unpackaged for the old_version when updating a module to extension style.

**CASCADE**

Automatically install dependant extensions are not already installed. Dependant extensions are checked recursively and those dependencies are also installed automatically. If the SCHEMA clause is specified, the schema applies to the extension and all dependant extensions that are installed. Other options that are specified are not applied to the automatically-installed dependant extensions. In particular, default versions are always selected when installing dependant extensions.

**Notes**

The extensions currently available for loading can be identified from the pg_available_extensions or pg_available_extension_versions system views.

Before you use CREATE EXTENSION to load an extension into a database, the supporting extension files must be installed including an extension control file and at least one least one SQL script file. The support files must be installed in the same location on all Greenplum Database hosts. For information about creating new extensions, see PostgreSQL information about Packaging Related Objects into an Extension.

**Compatibility**

CREATE EXTENSION is a Greenplum Database extension.

**See Also**

ALTER EXTENSION, DROP EXTENSION

---

**CREATE EXTERNAL TABLE**

Defines a new external table.

**Synopsis**

```
CREATE [READABLE] EXTERNAL TABLE table_name
        ( column name data type [, ...] | LIKE other_table )
        LOCATION ('file://seghost[:port]/path/file'[, ...])
        | ('gpfdist://filehost[:port]/file_pattern[#transform=trans_name]' [, ...])
```
CREATE [READABLE] EXTERNAL WEB TABLE table_name
  ( 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] ]]
  | 'AVRO'
  | 'PARQUET'
  | 'CUSTOM' (Formatter=<formatter_specifications>)
  [ ENCODING 'encoding' ]
  [ [ LOG ERRORS] SEGMENT REJECT LIMIT count
  [ROWS | PERCENT] ]

CREATE WRITABLE EXTERNAL TABLE table_name
  ( column_name data_type [, ...] | LIKE other_table )
  LOCATION('gpfdist://outputhost[:port]/filename[#transform=trans_name]' [, ...])
Description

See "Working with External Tables" in the Greenplum Database Administrator Guide for detailed information about external tables.

CREATE EXTERNAL TABLE or CREATE EXTERNAL WEB TABLE creates a new readable external table definition in Greenplum Database. Readable external tables are typically used for fast, parallel data loading. Once an external table is defined, you can query its data directly (and in parallel) using SQL.
commands. For example, you can select, join, or sort external table data. You can also create views for external tables. DML operations (UPDATE, INSERT, DELETE, or TRUNCATE) are not allowed on readable external tables, and you cannot create indexes on readable external tables.

`CREATE WRITABLE EXTERNAL TABLE` or `CREATE WRITABLE EXTERNAL WEB TABLE` creates a new writable external table definition in Greenplum Database. Writable external tables are typically used for unloading data from the database into a set of files or named pipes. Writable external web tables can also be used to output data to an executable program. Writable external tables can also be used as output targets for Greenplum parallel MapReduce calculations. Once a writable external table is defined, data can be selected from database tables and inserted into the writable external table. Writable external tables only allow INSERT operations – SELECT, UPDATE, DELETE or TRUNCATE are not allowed.

The main difference between regular external tables and external web tables is their data sources. Regular readable external tables access static flat files, whereas external web tables access dynamic data sources – either on a web server or by executing OS commands or scripts.

**Parameters**

**READABLE | WRITABLE**

- Specifies the type of external table, readable being the default. Readable external tables are used for loading data into Greenplum Database. Writable external tables are used for unloading data.

**WEB**

- Creates a readable or writable external web table definition in Greenplum Database. There are two forms of readable external web tables – those that access files via the `http://` protocol or those that access data by executing OS commands. Writable external web tables output data to an executable program that can accept an input stream of data. External web tables are not rescannable during query execution.
- The s3 protocol does not support external web tables. You can, however, create an external web table that executes a third-party tool to read data from or write data to S3 directly.

**table_name**

- The name of the new external table.

**column_name**

- The name of a column to create in the external table definition. Unlike regular tables, external tables do not have column constraints or default values, so do not specify those.

**LIKE other_table**

- The LIKE clause specifies a table from which the new external table automatically copies all column names, data types and Greenplum distribution policy. If the original table specifies any column constraints or default column values, those will not be copied over to the new external table definition.

**data_type**

- The data type of the column.

**LOCATION (protocol://host[:port]/path/file [, ...])**

- If you use the `gphdfs` protocol to read or write a file to a Hadoop file system (HDFS), refer to the `Specify gphdfs Protocol in an External Table Definition` for additional information about the `gphdfs` protocol LOCATION clause syntax.

- If you use the `pxf` protocol to access an external data source, refer to the `Creating an External Table Using PXF` documentation for detailed information about the `pxf` protocol LOCATION clause syntax.

- If you use the `s3` protocol to read or write to S3, refer to the `About the S3 Protocol URL` for additional information about the `s3` protocol LOCATION clause syntax.
For readable external tables, specifies the URI of the external data source(s) to be used to populate the external table or web table. Regular readable external tables allow the gpfdist or file protocols. External web tables allow the http protocol. If port is omitted, port 8080 is assumed for http and gpfdist protocols, and port 9000 for the gphdfs protocol. If using the gpfdist protocol, the path is relative to the directory from which gpfdist is serving files (the directory specified when you started the gpfdist program). Also, gpfdist can use wildcards or other C-style pattern matching (for example, a whitespace character is \\[:space:\]\\) to denote multiple files in a directory. For example:

'gpfdist://filehost:8081/*'
'gpfdist://masterhost:/my_load_file'
'file://seghost1/dbfast1/external/myfile.txt'
'http://intranet.example.com/finance/expenses.csv'

For writable external tables, specifies the URI location of the gpfdist process or S3 protocol that will collect data output from the Greenplum segments and write it to one or more named files. For gpfdist the path is relative to the directory from which gpfdist is serving files (the directory specified when you started the gpfdist program). If multiple gpfdist locations are listed, the segments sending data will be evenly divided across the available output locations. For example:

'gpfdist://outputhost:8081/data1.out',
'gpfdist://outputhost:8081/data2.out'

With two gpfdist locations listed as in the above example, half of the segments would send their output data to the data1.out file and the other half to the data2.out file.

With the option #transform=trans_name, you can specify a transform to apply when loading or extracting data. The trans_name is the name of the transform in the YAML configuration file you specify with the you run the gpfdist utility. For information about specifying a transform, see gpfdist in the Greenplum Utility Guide.

**ON MASTER**

Restricts all table-related operations to the Greenplum master segment. Permitted only on readable and writable external tables created with the s3 or custom protocols. The gpfdist, gpfdists, gphdfs, pxf, and file protocols do not support ON MASTER.

**Note:** Be aware of potential resource impacts when reading from or writing to external tables you create with the ON MASTER clause. You may encounter performance issues when you restrict table operations solely to the Greenplum master segment.

**EXECUTE 'command' [ON ...]**

Allowed for readable external web tables or writable external tables only. For readable external web tables, specifies the OS command to be executed by the segment instances. The command can be a single OS command or a script. The ON clause is used to specify which segment instances will execute the given command.

- **ON ALL** is the default. The command will be executed by every active (primary) segment instance on all segment hosts in the Greenplum Database system. If the command executes a script, that script must reside in the same location on all of the segment hosts and be executable by the Greenplum superuser (gpadmin).
- **ON MASTER** runs the command on the master host only.

**Note:** Logging is not supported for external web tables when the ON MASTER clause is specified.

- **ON number** means the command will be executed by the specified number of segments. The particular segments are chosen randomly at runtime by the Greenplum
Database system. If the command executes a script, that script must reside in the same
location on all of the segment hosts and be executable by the Greenplum superuser
(gpadmin).

- HOST means the command will be executed by one segment on each segment host
  (once per segment host), regardless of the number of active segment instances per
  host.
- HOST `segment_hostname` means the command will be executed by all active (primary)
  segment instances on the specified segment host.
- SEGMENT `segment_id` means the command will be executed only once by the
  specified segment. You can determine a segment instance's ID by looking at the
  `content` number in the system catalog table `gp_segment_configuration`. The `content` ID
  of the Greenplum Database master is always -1.

For writable external tables, the `command` specified in the `execute` clause must be
prepared to have data piped into it. Since all segments that have data to send will write
their output to the specified command or program, the only available option for the `on` clause is `all`.

```
FORMAT 'TEXT | CSV | AVRO | PARQUET' (options)
```

When the `format` clause identifies delimited text (`TEXT`) or comma separated values
(`CSV`) format, formatting options are similar to those available with the PostgreSQL
`COPY` command. If the data in the file does not use the default column delimiter, escape
character, null string and so on, you must specify the additional formatting options so that
the data in the external file is read correctly by Greenplum Database. For information about
using a custom format, see "Loading and Unloading Data" in the Greenplum Database
Administrator Guide.

The AVRO and PARQUET formats are supported only when you specify the `gphdfs`
protocol. Refer to Accessing HDFS Data with `gphdfs` for detailed information about the
gphdfs protocol `format` clause syntax.

If you use the `pxf` protocol to access an external data source, refer to the PXF Creating an
External Table Using PXF documentation for detailed information about the `pxf` protocol
`format` clause syntax.

```
FORMAT 'CUSTOM' (formatter=formatter_specification)
```

Specifies a custom data format. The `formatter_specification` specifies the function to use
to format the data, followed by comma-separated parameters to the formatter function.
The length of the formatter specification, the string including `Formatter=`, can be up to
approximately 50K bytes.

If you use the `pxf` protocol to access an external data source, refer to the PXF Creating an
External Table Using PXF documentation for detailed information about the `pxf` protocol
`format` clause syntax.

For general information about using a custom format, see "Loading and Unloading Data" in
the Greenplum Database Administrator Guide.

**Delimiter**

Specifies a single ASCII character that separates columns within each row (line) of data.
The default is a tab character in `TEXT` mode, a comma in `CSV` mode. In `TEXT` mode for
readable external tables, the delimiter can be set to `off` for special use cases in which
unstructured data is loaded into a single-column table.

For the `s3` protocol, the delimiter cannot be a newline character (`\n`) or a carriage return
character (`\r`).

**Null**

Specifies the string that represents a NULL value. The default is `\N` (backslash-N) in `TEXT`
mode, and an empty value with no quotations in `CSV` mode. You might prefer an empty
string even in TEXT mode for cases where you do not want to distinguish NULL values from empty strings. When using external and web tables, any data item that matches this string will be considered a NULL value.

As an example for the text format, this FORMAT clause can be used to specify that the string of two single quotes ("'") is a NULL value.

```sql
FORMAT 'text' (delimiter ',' null '\''\''\'' )
```

**ESCAPE**

Specifies the single character that is used for C escape sequences (such as \n, \t, \100, and so on) and for escaping data characters that might otherwise be taken as row or column delimiters. Make sure to choose an escape character that is not used anywhere in your actual column data. The default escape character is a \ (backslash) for text-formatted files and a " (double quote) for csv-formatted files, however it is possible to specify another character to represent an escape. It is also possible to disable escaping in text-formatted files by specifying the value 'OFF' as the escape value. This is very useful for data such as text-formatted web log data that has many embedded backslashes that are not intended to be escapes.

**NEWLINE**

Specifies the newline used in your data files – LF (Line feed, 0x0A), CR (Carriage return, 0x0D), or CRLF (Carriage return plus line feed, 0x0D 0x0A). If not specified, a Greenplum Database segment will detect the newline type by looking at the first row of data it receives and using the first newline type encountered.

**HEADER**

For readable external tables, specifies that the first line in the data file(s) is a header row (contains the names of the table columns) and should not be included as data for the table. If using multiple data source files, all files must have a header row.

For the s3 protocol, the column names in the header row cannot contain a newline character (\n) or a carriage return (\r).

The pxf protocol does not support the HEADER formatting option.

**QUOTE**

Specifies the quotation character for CSV mode. The default is double-quote (".

**FORCE NOT NULL**

In CSV mode, processes each specified column as though it were quoted and hence not a NULL value. For the default null string in CSV mode (nothing between two delimiters), this causes missing values to be evaluated as zero-length strings.

**FORCE QUOTE**

In CSV mode for writable external tables, forces quoting to be used for all non-NULL values in each specified column. NULL output is never quoted.

**FILL MISSING FIELDS**

In both TEXT and CSV mode for readable external tables, specifying FILL MISSING FIELDS will set missing trailing field values to NULL (instead of reporting an error) when a row of data has missing data fields at the end of a line or row. Blank rows, fields with a NOT NULL constraint, and trailing delimiters on a line will still report an error.

**ENCODING 'encoding'**

Character set encoding to use for the external table. Specify a string constant (such as 'SQL_ASCII'), an integer encoding number, or DEFAULT to use the default client encoding. See Character Set Support.

**LOG ERRORS**
This is an optional clause that can precede a `SEGMENT REJECT LIMIT` clause to log information about rows with formatting errors. The error log information is stored internally and is accessed with the Greenplum Database built-in SQL function `gp_read_error_log()`.

See Notes for information about the error log information and built-in functions for viewing and managing error log information.

**SEGMENT REJECT LIMIT count [ROWS | PERCENT]**

Runs a `COPY FROM` operation in single row error isolation mode. If the input rows have format errors they will be discarded provided that the reject limit count is not reached on any Greenplum segment instance during the load operation. The reject limit count can be specified as number of rows (the default) or percentage of total rows (1-100). If `PERCENT` is used, each segment starts calculating the bad row percentage only after the number of rows specified by the parameter `gp_reject_percent_threshold` has been processed. The default for `gp_reject_percent_threshold` is 300 rows. Constraint errors such as violation of a `NOT NULL`, `CHECK`, or `UNIQUE` constraint will still be handled in "all-or-nothing" input mode. If the limit is not reached, all good rows will be loaded and any error rows discarded.

**Note:** When reading an external table, Greenplum Database limits the initial number of rows that can contain formatting errors if the `SEGMENT REJECT LIMIT` is not triggered first or is not specified. If the first 1000 rows are rejected, the `COPY` operation is stopped and rolled back.

The limit for the number of initial rejected rows can be changed with the Greenplum Database server configuration parameter `gp_initial_bad_row_limit`. See **Server Configuration Parameters** for information about the parameter.

**DISTRIBUTED BY (column, [ ... ] )**

**DISTRIBUTED RANDOMLY**

Used to declare the Greenplum Database distribution policy for a writable external table. By default, writable external tables are distributed randomly. If the source table you are exporting data from has a hash distribution policy, defining the same distribution key column(s) for the writable external table will improve unload performance by eliminating the need to move rows over the interconnect. When you issue an unload command such as `INSERT INTO wex_table SELECT * FROM source_table`, the rows that are unloaded can be sent directly from the segments to the output location if the two tables have the same hash distribution policy.

**Examples**

Start the `gpfdist` file server program in the background on port 8081 serving files from directory `/var/data/staging`:

```
gpfdist -p 8081 -d /var/data/staging -l /home/gpadmin/log &
```

Create a readable external table named `ext_customer` using the `gpfdist` protocol and any text formatted files (`*.txt`) found in the `gpfdist` directory. The files are formatted with a pipe (`|`) as the column delimiter and an empty space as `NULL`. Also access the external table in single row error isolation mode:

```
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;
```
Create the same readable external table definition as above, but with CSV formatted files:

```sql
CREATE EXTERNAL TABLE ext_customer
(id int, name text, sponsor text)
LOCATION ('gpfdist://filehost:8081/*.csv')
FORMAT 'CSV' (DELIMITER ',');
```

Create a readable external table named `ext_expenses` using the file protocol and several CSV formatted files that have a header row:

```sql
CREATE EXTERNAL TABLE ext_expenses (name text, date date, amount float4, category text, description text)
FORMAT 'CSV' (HEADER);
```

Create a readable external web table that executes a script once per segment host:

```sql
CREATE EXTERNAL WEB TABLE log_output (linenum int, message text) EXECUTE '/var/load_scripts/get_log_data.sh' ON HOST FORMAT 'TEXT' (DELIMITER '|');
```

Create a writable external table named `sales_out` that uses `gpfdist` to write output data to a file named `sales.out`. The files are formatted with a pipe (`|`) as the column delimiter and an empty space as `NULL`.

```sql
CREATE WRITABLE EXTERNAL TABLE sales_out (LIKE sales)
LOCATION ('gpfdist://etl1:8081/sales.out')
FORMAT 'TEXT' (DELIMITER '|' NULL ' ')
DISTRIBUTED BY (txn_id);
```

Create a writable external web table that pipes output data received by the segments to an executable script named `to_adreport_etl.sh`:

```sql
CREATE WRITABLE EXTERNAL WEB TABLE campaign_out (LIKE campaign)
EXECUTE '/var/unload_scripts/to_adreport_etl.sh'
FORMAT 'TEXT' (DELIMITER '|');
```

Use the writable external table defined above to unload selected data:

```sql
INSERT INTO campaign_out SELECT * FROM campaign WHERE customer_id=123;
```

**Notes**

When you specify the `LOG ERRORS` clause, Greenplum Database captures errors that occur while reading the external table data. You can view and manage the captured error log data.

- Use the built-in SQL function `gp_read_error_log('table_name')`. It requires `SELECT` privilege on `table_name`. This example displays the error log information for data loaded into table `ext_expenses` with a `COPY` command:

  ```sql
  SELECT * from gp_read_error_log('ext_expenses');
  ```
For information about the error log format, see Viewing Bad Rows in the Error Log in the Greenplum Database Administrator Guide.

The function returns FALSE if table_name does not exist.

- If error log data exists for the specified table, the new error log data is appended to existing error log data. The error log information is not replicated to mirror segments.
- Use the built-in SQL function gp_truncate_error_log('table_name') to delete the error log data for table_name. It requires the table owner privilege. This example deletes the error log information captured when moving data into the table ext_expenses:

  SELECT gp_truncate_error_log('ext_expenses');

The function returns FALSE if table_name does not exist.

Specify the * wildcard character to delete error log information for existing tables in the current database. Specify the string *.* to delete all database error log information, including error log information that was not deleted due to previous database issues. If * is specified, database owner privilege is required. If *.* is specified, operating system super-user privilege is required.

When multiple Greenplum Database external tables are defined with the gpfdist, gpfdists, or file protocol and access the same named pipe a Linux system, Greenplum Database restricts access to the named pipe to a single reader. An error is returned if a second reader attempts to access the named pipe.

Compatibility

CREATE EXTERNAL TABLE is a Greenplum Database extension. The SQL standard makes no provisions for external tables.

See Also

CREATE TABLE AS, CREATE TABLE, COPY, SELECT INTO, INSERT

CREATE FUNCTION

Defines a new function.

Synopsis

```
CREATE [OR REPLACE] FUNCTION name
  ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } defexpr ] [, ...] ] )
  [ RETURNS { [ SETOF ] rettype ]
    | TABLE ( [{ argname argtype | LIKE other table } | [...] ])
  ]
  [ { LANGUAGE langname
      | IMMUTABLE | STABLE | VOLATILE
      | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
      | [EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINE
      | COST execution_cost
      | SET configuration_parameter { TO value | = value | FROM CURRENT }
      | AS 'definition'
      | AS 'obj_file', 'link_symbol' } ... ]
  [ WITH ( { DESCRIBE = describe_function } [, ...] ) ]
```

Description

CREATE FUNCTION defines a new function. CREATE OR REPLACE FUNCTION will either create a new function, or replace an existing definition.
The name of the new function must not match any existing function with the same argument types in the same schema. However, functions of different argument types may share a name (overloading).

To update the definition of an existing function, use `CREATE OR REPLACE FUNCTION`. It is not possible to change the name or argument types of a function this way (this would actually create a new, distinct function). Also, `CREATE OR REPLACE FUNCTION` will not let you change the return type of an existing function. To do that, you must drop and recreate the function. If you drop and then recreate a function, you will have to drop existing objects (rules, views, triggers, and so on) that refer to the old function. Use `CREATE OR REPLACE FUNCTION` to change a function definition without breaking objects that refer to the function.

For more information about creating functions, see the User Defined Functions section of the PostgreSQL documentation.

**Limited Use of VOLATILE and STABLE Functions**

To prevent data from becoming out-of-sync across the segments in Greenplum Database, any function classified as `STABLE` or `VOLATILE` cannot be executed at the segment level if it contains SQL or modifies the database in any way. For example, functions such as `random()` or `timeofday()` are not allowed to execute on distributed data in Greenplum Database because they could potentially cause inconsistent data between the segment instances.

To ensure data consistency, `VOLATILE` and `STABLE` functions can safely be used in statements that are evaluated on and execute from the master. For example, the following statements are always executed on the master (statements without a `FROM` clause):

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

In cases where a statement has a `FROM` clause containing a distributed table and the function used in the `FROM` clause simply returns a set of rows, execution may be allowed on the segments:

```sql
SELECT * FROM foo();
```

One exception to this rule are functions that return a table reference (`rangeFuncs`) or functions that use the `refCursor` data type. Note that you cannot return a `refcursor` from any kind of function in Greenplum Database.

**Parameters**

- **name**
  The name (optionally schema-qualified) of the function to create.

- **argmode**
  The mode of an argument: either `IN`, `OUT`, `INOUT`, or `VARIADIC`. Only `OUT` arguments can follow an argument declared as `VARIADIC`. If omitted, the default is `IN`.

- **argname**
  The name of an argument. Some languages (currently only PL/pgSQL) let you use the name in the function body. For other languages the name of an input argument is just extra documentation. But the name of an output argument is significant, since it defines the column name in the result row type. (If you omit the name for an output argument, the system will choose a default column name.)

- **argtype**
  The data type(s) of the function's arguments (optionally schema-qualified), if any. The argument types may be base, composite, or domain types, or may reference the type of a table column.
Depending on the implementation language it may also be allowed to specify pseudotypes such as `cstring`. Pseudotypes indicate that the actual argument type is either incompletely specified, or outside the set of ordinary SQL data types.

The type of a column is referenced by writing `tablename.columnname%TYPE`. Using this feature can sometimes help make a function independent of changes to the definition of a table.

**defexpr**

An expression to be used as the default value if the parameter is not specified. The expression must be coercible to the argument type of the parameter. Only `IN` and `INOUT` parameters can have a default value. Each input parameter in the argument list that follows a parameter with a default value must have a default value as well.

**rettype**

The return data type (optionally schema-qualified). The return type can be a base, composite, or domain type, or may reference the type of a table column. Depending on the implementation language it may also be allowed to specify pseudotypes such as `cstring`. If the function is not supposed to return a value, specify `void` as the return type.

When there are `OUT` or `INOUT` parameters, the `RETURNS` clause may be omitted. If present, it must agree with the result type implied by the output parameters: `RECORD` if there are multiple output parameters, or the same type as the single output parameter.

The `SETOF` modifier indicates that the function will return a set of items, rather than a single item.

The type of a column is referenced by writing `tablename.columnname%TYPE`.

**langname**

The name of the language that the function is implemented in. May be `SQL`, `C`, `internal`, or the name of a user-defined procedural language. See `CREATE LANGUAGE` for the procedural languages supported in Greenplum Database. For backward compatibility, the name may be enclosed by single quotes.

**IMMUTABLE**

**STABLE**

**VOLATILE**

These attributes inform the query optimizer about the behavior of the function. At most one choice may be specified. If none of these appear, `VOLATILE` is the default assumption. Since Greenplum Database currently has limited use of `VOLATILE` functions, if a function is truly `IMMUTABLE`, you must declare it as so to be able to use it without restrictions.

`IMMUTABLE` indicates that the function cannot modify the database and always returns the same result when given the same argument values. It does not do database lookups or otherwise use information not directly present in its argument list. If this option is given, any call of the function with all-constant arguments can be immediately replaced with the function value.

`STABLE` indicates that the function cannot modify the database, and that within a single table scan it will consistently return the same result for the same argument values, but that its result could change across SQL statements. This is the appropriate selection for functions whose results depend on database lookups, parameter values (such as the current time zone), and so on. Also note that the `current_timestamp` family of functions qualify as stable, since their values do not change within a transaction.

`VOLATILE` indicates that the function value can change even within a single table scan, so no optimizations can be made. Relatively few database functions are volatile in this sense; some examples are `random()`, `timeofday()`. But note that any function that has side-effects must be classified volatile, even if its result is quite predictable, to prevent calls from being optimized away; an example is `setval()`.
CALLED ON NULL INPUT
RETURNS NULL ON NULL INPUT

STRICT

CALLED ON NULL INPUT (the default) indicates that the function will be called normally when some of its arguments are null. It is then the function author’s responsibility to check for null values if necessary and respond appropriately. RETURNS NULL ON NULL INPUT or STRICT indicates that the function always returns null whenever any of its arguments are null. If this parameter is specified, the function is not executed when there are null arguments; instead a null result is assumed automatically.

[EXTERNAL] SECURITY INVOKER
[EXTERNAL] SECURITY DEFINER

SECURITY INVOKER (the default) indicates that the function is to be executed with the privileges of the user that calls it. SECURITY DEFINER specifies that the function is to be executed with the privileges of the user that created it. The key word EXTERNAL is allowed for SQL conformance, but it is optional since, unlike in SQL, this feature applies to all functions not just external ones.

COST execution_cost

A positive number identifying the estimated execution cost for the function, in cpu_operator_cost units. If the function returns a set, execution_cost identifies the cost per returned row. If the cost is not specified, C-language and internal functions default to 1 unit, while functions in other languages default to 100 units. The planner tries to evaluate the function less often when you specify larger execution_cost values.

configuration_parameter value

The SET clause applies a value to a session configuration parameter when the function is entered. The configuration parameter is restored to its prior value when the function exits. SET FROM CURRENT applies the session's current value of the parameter when the function is entered.

definition

A string constant defining the function; the meaning depends on the language. It may be an internal function name, the path to an object file, an SQL command, or text in a procedural language.

obj_file, link_symbol

This form of the AS clause is used for dynamically loadable C language functions when the function name in the C language source code is not the same as the name of the SQL function. The string obj_file is the name of the file containing the dynamically loadable object, and link_symbol is the name of the function in the C language source code. If the link symbol is omitted, it is assumed to be the same as the name of the SQL function being defined. It is recommended to locate shared libraries either relative to $libdir (which is located at $GPHOME/lib) or through the dynamic library path (set by the dynamic_library_path server configuration parameter). This simplifies version upgrades if the new installation is at a different location.

describe_function

The name of a callback function to execute when a query that calls this function is parsed. The callback function returns a tuple descriptor that indicates the result type.

Notes

Any compiled code (shared library files) for custom functions must be placed in the same location on every host in your Greenplum Database array (master and all segments). This location must also be in the LD_LIBRARY_PATH so that the server can locate the files. It is recommended to locate shared libraries either relative to $libdir (which is located at $GPHOME/lib) or through the dynamic library path (set
by the `dynamic_library_path` server configuration parameter) on all master segment instances in the Greenplum array.

The full SQL type syntax is allowed for input arguments and return value. However, some details of the type specification (such as the precision field for type `numeric`) are the responsibility of the underlying function implementation and are not recognized or enforced by the `CREATE FUNCTION` command.

Greenplum Database allows function overloading. The same name can be used for several different functions so long as they have distinct argument types. However, the C names of all functions must be different, so you must give overloaded C functions different C names (for example, use the argument types as part of the C names).

Two functions are considered the same if they have the same names and input argument types, ignoring any `OUT` parameters. Thus for example these declarations conflict:

```sql
CREATE FUNCTION foo(int) ...
CREATE FUNCTION foo(int, out text) ...
```

Functions that have different argument type lists are not considered to conflict at creation time, but if argument defaults are provided, they might conflict in use. For example, consider:

```sql
CREATE FUNCTION foo(int) ...
CREATE FUNCTION foo(int, int default 42) ...
```

The call `foo(10)`, will fail due to the ambiguity about which function should be called.

When repeated `CREATE FUNCTION` calls refer to the same object file, the file is only loaded once. To unload and reload the file, use the `LOAD` command.

You must have the `USAGE` privilege on a language to be able to define a function using that language.

It is often helpful to use dollar quoting to write the function definition string, rather than the normal single quote syntax. Without dollar quoting, any single quotes or backslashes in the function definition must be escaped by doubling them. A dollar-quoted string constant consists of a dollar sign (`$`), an optional tag of zero or more characters, another dollar sign, an arbitrary sequence of characters that makes up the string content, a dollar sign, the same tag that began this dollar quote, and a dollar sign. Inside the dollar-quoted string, single quotes, backslashes, or any character can be used without escaping. The string content is always written literally. For example, here are two different ways to specify the string "Dianne's horse" using dollar quoting:

```sql
$$Dianne's horse$$
$SomeTag$Dianne's horse$SomeTag$
```

If a `SET` clause is attached to a function, the effects of a `SET LOCAL` command executed inside the function for the same variable are restricted to the function; the configuration parameter's prior value is still restored when the function exits. However, an ordinary `SET` command (without `LOCAL`) overrides the `CREATE FUNCTION SET` clause, much as it would for a previous `SET LOCAL` command. The effects of such a command will persist after the function exits, unless the current transaction is rolled back.

If a function with a `VARIADIC` argument is declared as `STRICT`, the strictness check tests that the variadic array as a whole is non-null. PL/pgSQL will still call the function if the array has null elements.

### Using Functions With Queries on Distributed Data

In some cases, Greenplum Database does not support using functions in a query where the data in a table specified in the `FROM` clause is distributed over Greenplum Database segments. As an example, this SQL query contains the function `func()`:

```sql
SELECT func(a) FROM table1;
```

The function is not supported for use in the query if all of the following conditions are met:
• The data of table `table1` is distributed over Greenplum Database segments.
• The function `func()` reads or modifies data from distributed tables.
• The function `func()` returns more than one row or takes an argument (a) that comes from `table1`.

If any of the conditions are not met, the function is supported. Specifically, the function is supported if any of the following conditions apply:

• The function `func()` does not access data from distributed tables, or accesses data that is only on the Greenplum Database master.
• The table `table1` is a master only table.
• The function `func()` returns only one row and only takes input arguments that are constant values. The function is supported if it can be changed to require no input arguments.

Examples

A very simple addition function:

```sql
CREATE FUNCTION add(integer, integer) RETURNS integer
    AS 'select $1 + $2;'
    LANGUAGE SQL
    IMMUTABLE
    RETURNS NULL ON NULL INPUT;
```

Increment an integer, making use of an argument name, in PL/pgSQL:

```sql
CREATE OR REPLACE FUNCTION increment(i integer) RETURNS integer
    AS $$
    BEGIN
        RETURN i + 1;
    END;
    $$ LANGUAGE plpgsql;
```

Increase the default segment host memory per query for a PL/pgSQL function:

```sql
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';
```

Use polymorphic types to return an `ENUM` array:

```sql
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);
```

Return a record containing multiple output parameters:

```sql
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);
```
You can do the same thing more verbosely with an explicitly named composite type:

```
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);
```

## Compatibility

`CREATE FUNCTION` is defined in SQL:1999 and later. The Greenplum Database version is similar but not fully compatible. The attributes are not portable, neither are the different available languages.

For compatibility with some other database systems, `argmode` can be written either before or after `argname`. But only the first way is standard-compliant.

The SQL standard does not specify parameter defaults.

### See Also

`ALTER FUNCTION`, `DROP FUNCTION`, `LOAD`

## CREATE GROUP

Defines a new database role.

### Synopsis

```
CREATE GROUP name [[WITH] option [ ... ]]
```

where `option` can be:

```
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
| 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]
```

### Description

`CREATE GROUP` is an alias for `CREATE ROLE`. 
Compatibility
There is no CREATE GROUP statement in the SQL standard.

See Also
CREATE ROLE

**CREATE INDEX**

Defines a new index.

**Synopsis**

```sql
CREATE [UNIQUE] INDEX name ON table
[USING btree|bitmap|gist]
( {column | (expression)} [opclass] [, ...] )
[ WITH ( FILLFACTOR = value ) ]
[TABLESPACE tablespace]
[WHERE predicate]
```

**Description**

**CREATE INDEX** constructs an index on the specified table. Indexes are primarily used to enhance database performance (though inappropriate use can result in slower performance).

The key field(s) for the index are specified as column names, or alternatively as expressions written in parentheses. Multiple fields can be specified if the index method supports multicolumn indexes.

An index field can be an expression computed from the values of one or more columns of the table row. This feature can be used to obtain fast access to data based on some transformation of the basic data. For example, an index computed on `upper(col)` would allow the clause `WHERE upper(col) = 'JIM'` to use an index.

Greenplum Database provides the index methods B-tree, bitmap, and GiST. Users can also define their own index methods, but that is fairly complicated.

When the **WHERE** clause is present, a partial index is created. A partial index is an index that contains entries for only a portion of a table, usually a portion that is more useful for indexing than the rest of the table. For example, if you have a table that contains both billed and unbilled orders where the unbilled orders take up a small fraction of the total table and yet is most often selected, you can improve performance by creating an index on just that portion.

The expression used in the **WHERE** clause may refer only to columns of the underlying table, but it can use all columns, not just the ones being indexed. Subqueries and aggregate expressions are also forbidden in **WHERE**. The same restrictions apply to index fields that are expressions.

All functions and operators used in an index definition must be immutable. Their results must depend only on their arguments and never on any outside influence (such as the contents of another table or a parameter value). This restriction ensures that the behavior of the index is well-defined. To use a user-defined function in an index expression or **WHERE** clause, remember to mark the function **IMMUTABLE** when you create it.

**Parameters**

**UNIQUE**

Checks for duplicate values in the table when the index is created and each time data is added. Duplicate entries will generate an error. Unique indexes only apply to B-tree indexes. In Greenplum Database, unique indexes are allowed only if the columns of the index key are the same as (or a superset of) the Greenplum distribution key. On partitioned
tables, a unique index is only supported within an individual partition - not across all partitions.

**name**
The name of the index to be created. The index is always created in the same schema as its parent table.

**table**
The name (optionally schema-qualified) of the table to be indexed.

**btree | bitmap | gist**
The name of the index method to be used. Choices are btree, bitmap, and gist. The default method is btree.

**column**
The name of a column of the table on which to create the index. Only the B-tree, bitmap, and GiST index methods support multicolumn indexes.

**expression**
An expression based on one or more columns of the table. The expression usually must be written with surrounding parentheses, as shown in the syntax. However, the parentheses may be omitted if the expression has the form of a function call.

**opclass**
The name of an operator class. The operator class identifies the operators to be used by the index for that column. For example, a B-tree index on four-byte integers would use the int4_ops class (this operator class includes comparison functions for four-byte integers). In practice the default operator class for the column's data type is usually sufficient. The main point of having operator classes is that for some data types, there could be more than one meaningful ordering. For example, a complex-number data type could be sorted by either absolute value or by real part. We could do this by defining two operator classes for the data type and then selecting the proper class when making an index.

**FILLFACTOR**
The fillfactor for an index is a percentage that determines how full the index method will try to pack index pages. For B-trees, leaf pages are filled to this percentage during initial index build, and also when extending the index at the right (largest key values). If pages subsequently become completely full, they will be split, leading to gradual degradation in the index's efficiency.

B-trees use a default fillfactor of 90, but any value from 10 to 100 can be selected. If the table is static then fillfactor 100 is best to minimize the index's physical size, but for heavily updated tables a smaller fillfactor is better to minimize the need for page splits. The other index methods use fillfactor in different but roughly analogous ways; the default fillfactor varies between methods.

**tablespace**
The tablespace in which to create the index. If not specified, the default tablespace is used.

**predicate**
The constraint expression for a partial index.

**Notes**
When an index is created on a partitioned table, the index is propagated to all the child tables created by Greenplum Database. Creating an index on a table that is created by Greenplum Database for use by a partitioned table is not supported.
**UNIQUE** indexes are allowed only if the index columns are the same as (or a superset of) the Greenplum distribution key columns.

**UNIQUE** indexes are not allowed on append-optimized tables.

A **UNIQUE** index can be created on a partitioned table. However, uniqueness is enforced only within a partition; uniqueness is not enforced between partitions. For example, for a partitioned table with partitions that are based on year and a subpartitions that are based on quarter, uniqueness is enforced only on each individual quarter partition. Uniqueness is not enforced between quarter partitions.

Indexes are not used for **IS NULL** clauses by default. The best way to use indexes in such cases is to create a partial index using an **IS NULL** predicate.

bitmap indexes perform best for columns that have between 100 and 100,000 distinct values. For a column with more than 100,000 distinct values, the performance and space efficiency of a bitmap index decline. The size of a bitmap index is proportional to the number of rows in the table times the number of distinct values in the indexed column.

Columns with fewer than 100 distinct values usually do not benefit much from any type of index. For example, a gender column with only two distinct values for male and female would not be a good candidate for an index.

Prior releases of Greenplum Database also had an R-tree index method. This method has been removed because it had no significant advantages over the GiST method. If **USING rtree** is specified, **CREATE INDEX** will interpret it as **USING gist**.

For more information on the GiST index type, refer to the *PostgreSQL documentation*.

The use of hash and GIN indexes has been disabled in Greenplum Database.

**Examples**

To create a B-tree index on the column **title** in the table **films**:

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

To create a bitmap index on the column **gender** in the table **employee**:

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

To create an index on the expression **lower(title)**, allowing efficient case-insensitive searches:

```
CREATE INDEX lower_title_idx ON films ((lower(title)));
```

To create an index with non-default fill factor:

```
CREATE UNIQUE INDEX title_idx ON films (title) WITH (fillfactor = 70);
```

To create an index on the column **code** in the table **films** and have the index reside in the tablespace **indexspace**:

```
CREATE INDEX code_idx ON films(code) TABLESPACE indexspace;
```

**Compatibility**

**CREATE INDEX** is a Greenplum Database language extension. There are no provisions for indexes in the SQL standard.
Greenplum Database does not support the concurrent creation of indexes (CONCURRENTLY keyword not supported).

**See Also**

ALTER INDEX, DROP INDEX, CREATE TABLE, CREATE OPERATOR CLASS

### CREATE LANGUAGE

Defines a new procedural language.

#### Synopsis

```
CREATE [PROCEDURAL] LANGUAGE name
CREATE [TRUSTED] [PROCEDURAL] LANGUAGE name
    HANDLER call_handler [ INLINE inline_handler ]
    [VALIDATOR valfunction]
```

#### Description

CREATE LANGUAGE registers a new procedural language with a Greenplum database. Subsequently, functions and trigger procedures can be defined in this new language.

Superusers can register a new language with a Greenplum database. A database owner can also register within that database any language listed in the pg_pltemplate catalog in which the tmpldbacreate field is true. The default configuration allows only trusted languages to be registered by database owners. The creator of a language becomes its owner and can later drop it, rename it, or assign ownership to a new user.

CREATE LANGUAGE effectively associates the language name with a call handler that is responsible for executing functions written in that language. For a function written in a procedural language (a language other than C or SQL), the database server has no built-in knowledge about how to interpret the function's source code. The task is passed to a special handler that knows the details of the language. The handler could either do all the work of parsing, syntax analysis, execution, and so on or it could serve as a bridge between Greenplum Database and an existing implementation of a programming language. The handler itself is a C language function compiled into a shared object and loaded on demand, just like any other C function. Therese procedural language packages are included in the standard Greenplum Database distribution: PL/pgSQL, PL/Perl, and PL/Python. Language handlers have also been added for PL/Java and PL/R, but those languages are not pre-installed with Greenplum Database. See the topic on Procedural Languages in the PostgreSQL documentation for more information on developing functions using these procedural languages.

The PL/Perl, PL/Java, and PL/R libraries require the correct versions of Perl, Java, and R to be installed, respectively.

On RHEL and SUSE platforms, download the appropriate extensions from Pivotal Network, then install the extensions using the Greenplum Package Manager (gppkg) utility to ensure that all dependencies are installed as well as the extensions. See the Greenplum Database Utility Guide for details about gppkg.

There are two forms of the CREATE LANGUAGE command. In the first form, the user specifies the name of the desired language and the Greenplum Database server uses the pg_pltemplate system catalog to determine the correct parameters. In the second form, the user specifies the language parameters as well as the language name. You can use the second form to create a language that is not defined in pg_pltemplate.

When the server finds an entry in the pg_pltemplate catalog for the given language name, it will use the catalog data even if the command includes language parameters. This behavior simplifies loading of old dump files, which are likely to contain out-of-date information about language support functions.
Parameters

TRUSTED
Ignored if the server has an entry for the specified language name in `pg_pltemplate`. Specifies that the call handler for the language is safe and does not offer an unprivileged user any functionality to bypass access restrictions. If this key word is omitted when registering the language, only users with the superuser privilege can use this language to create new functions.

PROCEDURAL
This is a noise word.

name
The name of the new procedural language. The language name is case insensitive. The name must be unique among the languages in the database. Built-in support is included for `plpgsql`, `plperl`, `plpythonu`, and `plr`. The languages `plpgsql` (PL/pgSQL) and `plpythonu` (PL/Python) are installed by default in Greenplum Database.

HANDLER call_handler
Ignored if the server has an entry for the specified language name in `pg_pltemplate`. The name of a previously registered function that will be called to execute the procedural language functions. The call handler for a procedural language must be written in a compiled language such as C with version 1 call convention and registered with Greenplum Database as a function taking no arguments and returning the `language_handler` type, a placeholder type that is simply used to identify the function as a call handler.

INLINE inline_handler
The name of a previously registered function that is called to execute an anonymous code block in this language that is created with the `DO` command. If an `inline_handler` function is not specified, the language does not support anonymous code blocks. The handler function must take one argument of type `internal`, which is the `DO` command internal representation. The function typically return `void`. The return value of the handler is ignored.

VALIDATOR valfunction
Ignored if the server has an entry for the specified language name in `pg_pltemplate`. The name of a previously registered function that will be called to execute the procedural language functions. The call handler for a procedural language must be written in a compiled language such as C with version 1 call convention and registered with Greenplum Database as a function taking no arguments and returning the `language_handler` type, a placeholder type that is simply used to identify the function as a call handler.

Notes
The PL/pgSQL language is already registered in all databases by default. The PL/Python language extension is installed but not registered.

The system catalog `pg_language` records information about the currently installed languages.

To create functions in a procedural language, a user must have the `USAGE` privilege for the language. By default, `USAGE` is granted to `PUBLIC` (everyone) for trusted languages. This may be revoked if desired.

Procedural languages are local to individual databases. You create and drop languages for individual databases.

The call handler function and the validator function (if any) must already exist if the server does not have an entry for the language in `pg_pltemplate`. But when there is an entry, the functions need not already exist; they will be automatically defined if not present in the database.
Any shared library that implements a language must be located in the same LD_LIBRARY_PATH location on all segment hosts in your Greenplum Database array.

Examples
The preferred way of creating any of the standard procedural languages:

```sql
CREATE LANGUAGE plpgsql;
CREATE LANGUAGE plr;
```

For a language not known in the pg_pltemplate catalog:

```sql
CREATE FUNCTION plsample_call_handler() RETURNS language_handler
  AS '$libdir/plsample'
  LANGUAGE C;
CREATE LANGUAGE plsample
  HANDLER plsample_call_handler;
```

Compatibility
CREATE LANGUAGE is a Greenplum Database extension.

See Also
ALTER LANGUAGE, CREATE FUNCTION, DROP LANGUAGE

CREATE OPERATOR
Defines a new operator.

Synopsis

```sql
CREATE OPERATOR name (
  PROCEDURE = funcname
  [, LEFTARG = lefttype] [, RIGHTARG = righttype]
  [, COMMUTATOR = com_op] [, NEGATOR = neg_op]
  [, RESTRICT = res_proc] [, JOIN = join_proc]
  [, HASHES] [, MERGES] )
```

Description
CREATE OPERATOR defines a new operator. The user who defines an operator becomes its owner.

The operator name is a sequence of up to NAMEDATALEN-1 (63 by default) characters from the following list: + - * / < > = ~ ! @ # % ^ & | ` ?

There are a few restrictions on your choice of name:

- `--` and `/*` cannot appear anywhere in an operator name, since they will be taken as the start of a comment.
- A multicharacter operator name cannot end in `+` or `-`, unless the name also contains at least one of these characters: `~ ! @ # % ^ & | ` ?

For example, `@-` is an allowed operator name, but `*` is not. This restriction allows Greenplum Database to parse SQL-compliant commands without requiring spaces between tokens.

The operator `!=` is mapped to `<>` on input, so these two names are always equivalent.
At least one of LEFTARG and RIGHTARG must be defined. For binary operators, both must be defined. For right unary operators, only LEFTARG should be defined, while for left unary operators only RIGHTARG should be defined.

The funcname procedure must have been previously defined using CREATE FUNCTION, must be IMMUTABLE, and must be defined to accept the correct number of arguments (either one or two) of the indicated types.

The other clauses specify optional operator optimization clauses. These clauses should be provided whenever appropriate to speed up queries that use the operator. But if you provide them, you must be sure that they are correct. Incorrect use of an optimization clause can result in server process crashes, subtly wrong output, or other unexpected results. You can always leave out an optimization clause if you are not sure about it.

Parameters

name
The (optionally schema-qualified) name of the operator to be defined. Two operators in the same schema can have the same name if they operate on different data types.

funcname
The function used to implement this operator (must be an IMMUTABLE function).

lefttype
The data type of the operator’s left operand, if any. This option would be omitted for a left-unary operator.

righttype
The data type of the operator’s right operand, if any. This option would be omitted for a right-unary operator.

com_op
The optional COMMUTATOR clause names an operator that is the commutator of the operator being defined. We say that operator A is the commutator of operator B if (x A y) equals (y B x) for all possible input values x, y. Notice that B is also the commutator of A. For example, operators < and > for a particular data type are usually each others commutators, and operator + is usually commutative with itself. But operator - is usually not commutative with anything. The left operand type of a commutable operator is the same as the right operand type of its commutator, and vice versa. So the name of the commutator operator is all that needs to be provided in the COMMUTATOR clause.

neg_op
The optional NEGATOR clause names an operator that is the negator of the operator being defined. We say that operator A is the negator of operator B if both return Boolean results and (x A y) equals NOT (x B y) for all possible inputs x, y. Notice that B is also the negator of A. For example, < and >= are a negator pair for most data types. An operator’s negator must have the same left and/or right operand types as the operator to be defined, so only the operator name need be given in the NEGATOR clause.

res_proc
The optional RESTRICT names a restriction selectivity estimation function for the operator. Note that this is a function name, not an operator name. RESTRICT clauses only make sense for binary operators that return boolean. The idea behind a restriction selectivity estimator is to guess what fraction of the rows in a table will satisfy a WHERE-clause condition of the form:

\[
\text{column OP constant}
\]

for the current operator and a particular constant value. This assists the optimizer by giving it some idea of how many rows will be eliminated by WHERE clauses that have this form.
You can usually just use one of the following system standard estimator functions for many of your own operators:

- eqsel for =
- neqsel for <>
- scalarltsel for < or <=
- scalargtssel for > or >=

**join_proc**

The optional JOIN clause names a join selectivity estimation function for the operator. Note that this is a function name, not an operator name. JOIN clauses only make sense for binary operators that return boolean. The idea behind a join selectivity estimator is to guess what fraction of the rows in a pair of tables will satisfy a WHERE-clause condition of the form

\[ \text{table1.column1 OP table2.column2} \]

for the current operator. This helps the optimizer by letting it figure out which of several possible join sequences is likely to take the least work.

You can usually just use one of the following system standard join selectivity estimator functions for many of your own operators:

- eqjoinsel for =
- neqjoinsel for <>
- scalarltjoinsel for < or <=
- scalargtjoinsel for > or >=
- areajoinsel for 2D area-based comparisons
- positionjoinsel for 2D position-based comparisons
- contjoinsel for 2D containment-based comparisons

**HASHES**

The optional HASHES clause tells the system that it is permissible to use the hash join method for a join based on this operator. HASHES only makes sense for a binary operator that returns boolean. The hash join operator can only return true for pairs of left and right values that hash to the same hash code. If two values are put in different hash buckets, the join will never compare them, implicitly assuming that the result of the join operator must be false. Because of this, it never makes sense to specify HASHES for operators that do not represent equality.

In most cases, it is only practical to support hashing for operators that take the same data type on both sides. However, you can design compatible hash functions for two or more data types, which are functions that will generate the same hash codes for "equal" values, even if the values are differently represented.

To be marked HASHES, the join operator must appear in a hash index operator class. Attempts to use the operator in hash joins will fail at run time if no such operator class exists. The system needs the operator class to find the data-type-specific hash function for the operator's input data type. You must also supply a suitable hash function before you can create the operator class. Exercise care when preparing a hash function, as there are machine-dependent ways in which it could fail to function correctly. For example, on machines that meet the IEEE floating-point standard, negative zero and positive zero are different values (different bit patterns) but are defined to compare as equal. If a float value could contain a negative zero, define it to generate the same hash value as positive zero.
A hash-joinable operator must have a commutator (itself, if the two operand data types are the same, or a related equality operator if they are different) that appears in the same operator family. Otherwise, planner errors can occur when the operator is used. For better optimization, a hash operator family that supports multiple data types should provide equality operators for every combination of the data types.

**Note:** The function underlying a hash-joinable operator must be marked immutable or stable; an operator marked as volatile will not be used. If a hash-joinable operator has an underlying function that is marked strict, the function must also be complete, returning true or false, and not null, for any two non-null inputs.

**MERGES**

The `MERGES` clause, if present, tells the system that it is permissible to use the merge-join method for a join based on this operator. `MERGES` only makes sense for a binary operator that returns `boolean`, and in practice the operator must represent equality for some data type or pair of data types.

Merge join is based on the idea of sorting the left- and right-hand tables into order and then scanning them in parallel. This means both data types must be capable of being fully ordered, and the join operator must be one that can only succeed for pairs of values that fall at equivalent places in the sort order. In practice, this means that the join operator must behave like an equality operator. However, you can merge-join two distinct data types so long as they are logically compatible. For example, the `smallint-versus-integer` equality operator is merge-joinable. Only sorting operators that bring both data types into a logically compatible sequence are needed.

To be marked `MERGES`, the join operator must appear as an equality member of a btree index operator family. This is not enforced when you create the operator, because the referencing operator family does not exist until later. However, the operator will not actually be used for merge joins unless a matching operator family can be found. The `MERGE` flag thus acts as a suggestion to the planner to look for a matching operator family.

A merge-joinable operator must have a commutator that appears in the same operator family. This would be itself, if the two operand data types are the same, or a related equality operator if the data types are different. Without an appropriate commutator, planner errors can occur when the operator is used. Also, although not strictly required, a btree operator family that supports multiple data types should be able to provide equality operators for every combination of the data types; this allows better optimization.

**Note:** `SORT1`, `SORT2`, `LTCMP`, and `GTCMP` were formerly used to specify the names of sort operators associated with a merge-joinable operator. Information about associated operators is now found by looking at B-tree operator families; specifying any of these operators will be ignored, except that it will implicitly set `MERGES` to true.

**Notes**

Any functions used to implement the operator must be defined as `IMMUTABLE`.

**Examples**

Here is an example of creating an operator for adding two complex numbers, assuming we have already created the definition of type `complex`. First define the function that does the work, then define the operator:

```sql
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 = +
);

To use this operator in a query:

```
SELECT (a + b) AS c FROM test_complex;
```

**Compatibility**

CREATE OPERATOR is a Greenplum Database language extension. The SQL standard does not provide for user-defined operators.

**See Also**

CREATE FUNCTION, CREATE TYPE, ALTER OPERATOR, DROP OPERATOR

**CREATE OPERATOR CLASS**

Defines a new operator class.

**Synopsis**

```
CREATE OPERATOR CLASS name [DEFAULT] FOR TYPE data_type
    USING index_method AS
    {
        OPERATOR strategy_number op_name [(op_type, op_type)] [RECHECK]
        | FUNCTION support_number funcname (argument_type [, ...] )
        | STORAGE storage_type
    } [, ... ]
```

**Description**

CREATE OPERATOR CLASS creates a new operator class. An operator class defines how a particular data type can be used with an index. The operator class specifies that certain operators will fill particular roles or strategies for this data type and this index method. The operator class also specifies the support procedures to be used by the index method when the operator class is selected for an index column. All the operators and functions used by an operator class must be defined before the operator class is created. Any functions used to implement the operator class must be defined as IMMUTABLE.

CREATE OPERATOR CLASS does not presently check whether the operator class definition includes all the operators and functions required by the index method, nor whether the operators and functions form a self-consistent set. It is the user's responsibility to define a valid operator class.

You must be a superuser to create an operator class.

**Parameters**

`name`

The (optionally schema-qualified) name of the operator class to be defined. Two operator classes in the same schema can have the same name only if they are for different index methods.

`DEFAULT`

Makes the operator class the default operator class for its data type. At most one operator class can be the default for a specific data type and index method.
**data_type**

The column data type that this operator class is for.

**index_method**

The name of the index method this operator class is for. Choices are `btree`, `bitmap`, and `gist`.

**strategy_number**

The operators associated with an operator class are identified by *strategy numbers*, which serve to identify the semantics of each operator within the context of its operator class. For example, B-trees impose a strict ordering on keys, lesser to greater, and so operators like `less than` and `greater than or equal to` are interesting with respect to a B-tree. These strategies can be thought of as generalized operators. Each operator class specifies which actual operator corresponds to each strategy for a particular data type and interpretation of the index semantics. The corresponding strategy numbers for each index method are as follows:

**Table 90: B-tree and Bitmap Strategies**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Strategy Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>less than</td>
<td>1</td>
</tr>
<tr>
<td>less than or equal</td>
<td>2</td>
</tr>
<tr>
<td>equal</td>
<td>3</td>
</tr>
<tr>
<td>greater than or equal</td>
<td>4</td>
</tr>
<tr>
<td>greater than</td>
<td>5</td>
</tr>
</tbody>
</table>

**Table 91: GiST Two-Dimensional Strategies (R-Tree)**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Strategy Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>strictly left of</td>
<td>1</td>
</tr>
<tr>
<td>does not extend to right of</td>
<td>2</td>
</tr>
<tr>
<td>overlaps</td>
<td>3</td>
</tr>
<tr>
<td>does not extend to left of</td>
<td>4</td>
</tr>
<tr>
<td>strictly right of</td>
<td>5</td>
</tr>
<tr>
<td>same</td>
<td>6</td>
</tr>
<tr>
<td>contains</td>
<td>7</td>
</tr>
<tr>
<td>contained by</td>
<td>8</td>
</tr>
<tr>
<td>does not extend above</td>
<td>9</td>
</tr>
<tr>
<td>strictly below</td>
<td>10</td>
</tr>
<tr>
<td>strictly above</td>
<td>11</td>
</tr>
<tr>
<td>does not extend below</td>
<td>12</td>
</tr>
</tbody>
</table>

**operator_name**

The name (optionally schema-qualified) of an operator associated with the operator class.

**op_type**
The operand data type(s) of an operator, or \texttt{NONE} to signify a left-unary or right-unary operator. The operand data types may be omitted in the normal case where they are the same as the operator class data type.

**RECHECK**

If present, the index is "lossy" for this operator, and so the rows retrieved using the index must be rechecked to verify that they actually satisfy the qualification clause involving this operator.

**support_number**

Index methods require additional support routines in order to work. These operations are administrative routines used internally by the index methods. As with strategies, the operator class identifies which specific functions should play each of these roles for a given data type and semantic interpretation. The index method defines the set of functions it needs, and the operator class identifies the correct functions to use by assigning them to the \texttt{support function numbers} as follows:

**Table 92: B-tree and Bitmap Support Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Support Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare two keys and return an integer less than zero, zero, or greater than zero, indicating whether the first key is less than, equal to, or greater than the second.</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 93: GiST Support Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Support Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>consistent - determine whether key satisfies the query qualifier.</td>
<td>1</td>
</tr>
<tr>
<td>union - compute union of a set of keys.</td>
<td>2</td>
</tr>
<tr>
<td>compress - compute a compressed representation of a key or value to be indexed.</td>
<td>3</td>
</tr>
<tr>
<td>decompress - compute a decompressed representation of a compressed key.</td>
<td>4</td>
</tr>
<tr>
<td>penalty - compute penalty for inserting new key into subtree with given subtree's key.</td>
<td>5</td>
</tr>
<tr>
<td>picksplit - determine which entries of a page are to be moved to the new page and compute the union keys for resulting pages.</td>
<td>6</td>
</tr>
<tr>
<td>equal - compare two keys and return true if they are equal.</td>
<td>7</td>
</tr>
</tbody>
</table>

**funcname**

The name (optionally schema-qualified) of a function that is an index method support procedure for the operator class.

**argument_types**

The parameter data type(s) of the function.

**storage_type**
The data type actually stored in the index. Normally this is the same as the column data type, but the GiST index method allows it to be different. The `STORAGE` clause must be omitted unless the index method allows a different type to be used.

**Notes**

Because the index machinery does not check access permissions on functions before using them, including a function or operator in an operator class is the same as granting public execute permission on it. This is usually not an issue for the sorts of functions that are useful in an operator class.

The operators should not be defined by SQL functions. A SQL function is likely to be inlined into the calling query, which will prevent the optimizer from recognizing that the query matches an index.

Any functions used to implement the operator class must be defined as `IMMUTABLE`.

**Examples**

The following example command defines a GiST index operator class for the data type `_int4` (array of `int4`):

```sql
CREATE OPERATOR CLASS gist__int_ops
DEFAULT FOR TYPE _int4 USING gist AS
    OPERATOR 3 &&,
    OPERATOR 6 = RECHECK,
    OPERATOR 7 @>,
    OPERATOR 8 <@,
    OPERATOR 20 @@ (_int4, query_int),
    FUNCTION 1 g_int_consistent (internal, _int4, int4),
    FUNCTION 2 g_int_union (bytea, 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);
```

**Compatibility**

CREATE OPERATOR CLASS is a Greenplum Database extension. There is no CREATE OPERATOR CLASS statement in the SQL standard.

**See Also**

CREATE OPERATOR FAMILY, DROP OPERATOR FAMILY, CREATE FUNCTION

**CREATE OPERATOR FAMILY**

Defines a new operator family.

**Synopsis**

```
CREATE OPERATOR FAMILY name USING index_method
```

**Description**

CREATE OPERATOR FAMILY creates a new operator family. An operator family defines a collection of related operator classes, and perhaps some additional operators and support functions that are compatible with these operator classes but not essential for the functioning of any individual index. (Operators and functions that are essential to indexes should be grouped within the relevant operator class, rather than being "loose" in the operator family. Typically, single-data-type operators are bound to operator classes,
while cross-data-type operators can be loose in an operator family containing operator classes for both data types.)

The new operator family is initially empty. It should be populated by issuing subsequent CREATE OPERATOR CLASS commands to add contained operator classes, and optionally ALTER OPERATOR FAMILY commands to add "loose" operators and their corresponding support functions.

If a schema name is given then the operator family is created in the specified schema. Otherwise it is created in the current schema. Two operator families in the same schema can have the same name only if they are for different index methods.

The user who defines an operator family becomes its owner. Presently, the creating user must be a superuser. (This restriction is made because an erroneous operator family definition could confuse or even crash the server.)

**Parameters**

*name*

The (optionally schema-qualified) name of the operator family to be defined. The name can be schema-qualified.

*index_method*

The name of the index method this operator family is for.

**Compatibility**

CREATE OPERATOR FAMILY is a Greenplum Database extension. There is no CREATE OPERATOR FAMILY statement in the SQL standard.

**See Also**

ALTER OPERATOR FAMILY, DROP OPERATOR FAMILY, CREATE FUNCTION, ALTER OPERATOR CLASS, CREATE OPERATOR CLASS, DROP OPERATOR CLASS

---

**CREATE PROTOCOL**

Registers a custom data access protocol that can be specified when defining a Greenplum Database external table.

**Synopsis**

```
CREATE [TRUSTED] PROTOCOL name (  
    [readfunc='read_call_handler'] [, writefunc='write_call_handler']  
    [, validatorfunc='validate_handler' ]
)
```

**Description**

CREATE PROTOCOL associates a data access protocol name with call handlers that are responsible for reading from and writing data to an external data source.

The CREATE PROTOCOL must specify either a read call handler or a write call handler. The call handlers specified in the CREATE PROTOCOL command must be defined in the database.

The protocol name can be specified in an CREATE EXTERNAL TABLE command.

For information about creating and enabling a custom data access protocol, see "Example Custom Data Access Protocol" in the Greenplum Database Administrator Guide.

**Parameters**

TRUSTED
A noise word.

**name**

The name of the data access protocol. The protocol name is case sensitive. The name must be unique among the protocols in the database.

**readfunc='read_call_handler'**

The name of a previously registered function that Greenplum Database calls to read data from an external data source. The command must specify either a read call handler or a write call handler.

**writefunc='write_call_handler'**

The name of a previously registered function that Greenplum Database calls to write data to an external data source. The command must specify either a read call handler or a write call handler.

**validatorfunc='validate_handler'**

An optional validator function that validates the URL specified in the `CREATE EXTERNAL TABLE` command.

---

**Notes**

Greenplum Database installs the custom protocols `file`, `gpfdist`, `gpfdists`, and `gphdfs` by default. Optionally, the `s3` protocol can be installed.

Any shared library that implements a data access protocol must be located in the same location on all Greenplum Database segment hosts. For example, the shared library can be in a location specified by the operating system environment variable `LD_LIBRARY_PATH` on all hosts. You can also specify the location when you define the handler function. For example, when you define the `s3` protocol in the `CREATE PROTOCOL` command, you specify `$libdir/gps3ext.so` as the location of the shared object, where `$libdir` is located at `$GPHOME/lib`.

**Compatibility**

`CREATE PROTOCOL` is a Greenplum Database extension.

**See Also**

`ALTER PROTOCOL`, `CREATE EXTERNAL TABLE`, `DROP PROTOCOL`

---

**CREATE RESOURCE GROUP**

**Warning:** Resource groups are an experimental feature and are not intended for use in a production environment. Experimental features are subject to change without notice in future releases.

Defines a new resource group.

**Synopsis**

```sql
CREATE RESOURCE GROUP name WITH (group_attribute=value [, ... ])
```

where `group_attribute` is:

```sql
CPU_RATE_LIMIT=integer
MEMORY_LIMIT=integer
[ CONCURRENCY=integer ]
[ MEMORY_SHARED_QUOTA=integer ]
[ MEMORY_SPILL_RATIO=integer ]
```
**Description**

Creates a new resource group for Greenplum Database resource management. A resource group is assigned to one or more roles and identifies concurrent transaction, memory, and CPU limits for the role when resource groups are enabled.

You must have SUPERUSER privileges to create a resource group. The maximum number of resource groups allowed in your Greenplum Database cluster is 100.

Greenplum Database pre-defines two default resource groups: admin_group and default_group. These group names, as well as the group name none, are reserved.

To set appropriate limits for resource groups, the Greenplum Database administrator must be familiar with the queries typically executed on the system, as well as the users/roles executing those queries.

After creating a resource group, assign the group to one or more roles using the ALTER ROLE or CREATE ROLE commands.

**Parameters**

`name`

The name of the resource group.

**CONCURRENCY integer**

The maximum number of concurrent transactions, including active and idle transactions, that are permitted for this resource group. The CONCURRENCY value must be an integer in the range [0 .. max_connections]. The default CONCURRENCY value is 20.

*Note:* You cannot set the CONCURRENCY value for the admin_group to zero (0).

**CPU_RATE_LIMIT integer**

Required. The percentage of CPU resources to allocate to this resource group. The minimum CPU percentage you can specify for a resource group is 1. The maximum is 100. The sum of the CPU_RATE_LIMIT values specified for all resource groups defined in the Greenplum Database cluster must be less than or equal to 100.

**MEMORY_LIMIT integer**

Required. The total percentage of Greenplum Database memory resources to allocate to this resource group. The minimum memory percentage you can specify for a resource group is 1. The maximum is 100. The sum of the MEMORY_LIMIT values specified for all resource groups defined in the Greenplum Database cluster must be less than or equal to 100.

**MEMORY_SHARED_QUOTA integer**

The quota of shared memory in the resource group. Resource groups with a MEMORY_SHARED_QUOTA threshold set aside a percentage of memory allotted to the resource group to share across transactions. This shared memory is allocated on a first-come, first-served basis as available. A transaction may use none, some, or all of this memory. The minimum memory shared quota percentage you can specify for a resource group is 0. The maximum is 100. The default MEMORY_SHARED_QUOTA value is 20.

**MEMORY_SPILL_RATIO integer**

The memory usage threshold for memory-intensive operators in a transaction. When the transaction reaches this threshold, it spills to disk. The minimum memory spill ratio percentage you can specify for a resource group is 0. The maximum is 100. The default MEMORY_SPILL_RATIO value is 20.

**Notes**

You cannot submit a CREATE RESOURCE GROUP command in an explicit transaction or sub-transaction.
Use the `gp_toolkit.gp_resgroup_config` system view to display the limit settings of all resource groups:

```
SELECT * FROM gp_toolkit.gp_resgroup_config;
```

**Examples**

Create a resource group with CPU and memory limit percentages of 35:

```
CREATE RESOURCE GROUP rgroup1 WITH (CPU_RATE_LIMIT=35, MEMORY_LIMIT=35);
```

Create a resource group with a concurrent transaction limit of 30, a memory limit of 15, and a CPU limit of 25:

```
CREATE RESOURCE GROUP rgroup2 WITH (CONCURRENCY=20,
MEMORY_LIMIT=15, CPU_RATE_LIMIT=25);
```

**Compatibility**

`CREATE RESOURCE GROUP` is a Greenplum Database extension. There is no provision for resource groups or resource management in the SQL standard.

**See Also**

`ALTER ROLE`, `CREATE ROLE`, `ALTER RESOURCE GROUP`, `DROP RESOURCE GROUP`

### CREATE RESOURCE QUEUE

Defines a new resource queue.

**Synopsis**

```
CREATE RESOURCE QUEUE name WITH (queue_attribute=value [, ... ])
```

where `queue_attribute` is:

```
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' ]

**Description**

Creates a new resource queue for Greenplum Database resource management. A resource queue must have either an `ACTIVE_STATEMENTS` or a `MAX_COST` value (or it can have both). Only a superuser can create a resource queue.

Resource queues with an `ACTIVE_STATEMENTS` threshold set a maximum limit on the number of queries that can be executed by roles assigned to that queue. It controls the number of active queries that are allowed to run at the same time. The value for `ACTIVE_STATEMENTS` should be an integer greater than 0.
Resource queues with a **MAX_COST** threshold set a maximum limit on the total cost of queries that can be executed by roles assigned to that queue. Cost is measured in the *estimated total cost* for the query as determined by the Greenplum Database query planner (as shown in the `EXPLAIN` output for a query). Therefore, an administrator must be familiar with the queries typically executed on the system in order to set an appropriate cost threshold for a queue. Cost is measured in units of disk page fetches; 1.0 equals one sequential disk page read. The value for **MAX_COST** is specified as a floating point number (for example 100.0) or can also be specified as an exponent (for example 1e+2). If a resource queue is limited based on a cost threshold, then the administrator can allow **COST_OVERCOMMIT=TRUE** (the default). This means that a query that exceeds the allowed cost threshold will be allowed to run but only when the system is idle. If **COST_OVERCOMMIT=FALSE** is specified, queries that exceed the cost limit will always be rejected and never allowed to run. Specifying a value for **MIN_COST** allows the administrator to define a cost for small queries that will be exempt from resource queuing.

If a value is not defined for **ACTIVE_STATEMENTS** or **MAX_COST**, it is set to -1 by default (meaning no limit). After defining a resource queue, you must assign roles to the queue using the `ALTER ROLE` or `CREATE ROLE` command.

You can optionally assign a **PRIORITY** to a resource queue to control the relative share of available CPU resources used by queries associated with the queue in relation to other resource queues. If a value is not defined for **PRIORITY**, queries associated with the queue have a default priority of **MEDIUM**.

Resource queues with an optional **MEMORY_LIMIT** threshold set a maximum limit on the amount of memory that all queries submitted through a resource queue can consume on a segment host. This determines the total amount of memory that all worker processes of a query can consume on a segment host during query execution. Greenplum recommends that **MEMORY_LIMIT** be used in conjunction with **ACTIVE_STATEMENTS** rather than with **MAX_COST**. The default amount of memory allotted per query on statement-based queues is: **MEMORY_LIMIT / ACTIVE_STATEMENTS**. The default amount of memory allotted per query on cost-based queues is: **MEMORY_LIMIT * (query_cost / MAX_COST)**.

The default memory allotment can be overridden on a per-query basis using the `statement_mem` server configuration parameter, provided that **MEMORY_LIMIT** or `max_statement_mem` is not exceeded. For example, to allocate more memory to a particular query:

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

The **MEMORY_LIMIT** value for all of your resource queues should not exceed the amount of physical memory of a segment host. If workloads are staggered over multiple queues, memory allocations can be oversubscribed. However, queries can be cancelled during execution if the segment host memory limit specified in `gp_vmem_protect_limit` is exceeded.

For information about `statement_mem`, `max_statement`, and `gp_vmem_protect_limit`, see Server Configuration Parameters.

### Parameters

- **name**
  
  The name of the resource queue.

- **ACTIVE_STATEMENTS** integer
  
  Resource queues with an **ACTIVE_STATEMENTS** threshold limit the number of queries that can be executed by roles assigned to that queue. It controls the number of active queries that are allowed to run at the same time. The value for **ACTIVE_STATEMENTS** should be an integer greater than 0.

- **MEMORY_LIMIT** `memory_units`
  
  Sets the total memory quota for all statements submitted from users in this resource queue. Memory units can be specified in kB, MB or GB. The minimum memory quota for a resource queue is 10MB. There is no maximum, however the upper boundary at query
execution time is limited by the physical memory of a segment host. The default is no limit (-1).

**MAX_COST float**

Resource queues with a **MAX_COST** threshold set a maximum limit on the total cost of queries that can be executed by roles assigned to that queue. Cost is measured in the *estimated total cost* for the query as determined by the Greenplum Database query optimizer (as shown in the **EXPLAIN** output for a query). Therefore, an administrator must be familiar with the queries typically executed on the system in order to set an appropriate cost threshold for a queue. Cost is measured in units of disk page fetches; 1.0 equals one sequential disk page read. The value for **MAX_COST** is specified as a floating point number (for example 100.0) or can also be specified as an exponent (for example 1e+2).

**COST_OVERCOMMIT boolean**

If a resource queue is limited based on **MAX_COST**, then the administrator can allow **COST_OVERCOMMIT** (the default). This means that a query that exceeds the allowed cost threshold will be allowed to run but only when the system is idle. If **COST_OVERCOMMIT=FALSE** is specified, queries that exceed the cost limit will always be rejected and never allowed to run.

**MIN_COST float**

The minimum query cost limit of what is considered a small query. Queries with a cost under this limit will not be queued and run immediately. Cost is measured in the *estimated total cost* for the query as determined by the Greenplum Database query planner (as shown in the **EXPLAIN** output for a query). Therefore, an administrator must be familiar with the queries typically executed on the system in order to set an appropriate cost for what is considered a small query. Cost is measured in units of disk page fetches; 1.0 equals one sequential disk page read. The value for **MIN_COST** is specified as a floating point number (for example 100.0) or can also be specified as an exponent (for example 1e+2).

**PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX}**

Sets the priority of queries associated with a resource queue. Queries or statements in queues with higher priority levels will receive a larger share of available CPU resources in case of contention. Queries in low-priority queues may be delayed while higher priority queries are executed. If no priority is specified, queries associated with the queue have a priority of **MEDIUM**.

**Notes**

Use the **gp_toolkit.gp_resqueue_status** system view to see the limit settings and current status of a resource queue:

```
SELECT * from gp_toolkit.gp_resqueue_status WHERE rsqname='queue_name';
```

There is also another system view named **pg_stat_resqueues** which shows statistical metrics for a resource queue over time. To use this view, however, you must enable the **stats_queue_level** server configuration parameter. See "Managing Workload and Resources" in the *Greenplum Database Administrator Guide* for more information about using resource queues.

**CREATE RESOURCE QUEUE** cannot be run within a transaction.

Also, an SQL statement that is run during the execution of an **EXPLAIN ANALYZE** command is excluded from resource queues.
Examples

Create a resource queue with an active query limit of 20:

```sql
CREATE RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=20);
```

Create a resource queue with an active query limit of 20 and a total memory limit of 2000MB (each query will be allocated 100MB of segment host memory at execution time):

```sql
CREATE RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=20, MEMORY_LIMIT='2000MB');
```

Create a resource queue with a query cost limit of 3000.0:

```sql
CREATE RESOURCE QUEUE myqueue WITH (MAX_COST=3000.0);
```

Create a resource queue with a query cost limit of 3.10 (or 30000000000.0) and do not allow overcommit.

```sql
CREATE RESOURCE QUEUE myqueue WITH (MAX_COST=3e+10, COST_OVERCOMMIT=FALSE, MIN_COST=500.0);
```

Create a resource queue with both an active query limit and a query cost limit:

```sql
CREATE RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=30, MAX_COST=5000.00);
```

Create a resource queue with an active query limit of 5 and a maximum priority setting:

```sql
CREATE RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=5, PRIORITY=MAX);
```

Compatibility

`CREATE RESOURCE QUEUE` is a Greenplum Database extension. There is no provision for resource queues or resource management in the SQL standard.

See Also

`ALTER ROLE`, `CREATE ROLE`, `ALTER RESOURCE QUEUE`, `DROP RESOURCE QUEUE`

CREATE ROLE

Defines a new database role (user or group).

Synopsis

```
CREATE ROLE name [[WITH] option [ ... ]]
```

where `option` can be:

- `SUPERUSER | NOSUPERUSER`
- `CREATEDB | NOCREATEDB`
- `CREATEROLE | NOCREATEROLE`
- `CREATEUSER | NOCREATEUSER`
- `CREATEEXTTABLE | NOCREATEEXTTABLE`

```sql
[ ( attribute='value'[, ...] ) ]
```

where attributes and value are:
CREATE ROLE adds a new role to a Greenplum Database system. A role is an entity that can own database objects and have database privileges. A role can be considered a user, a group, or both depending on how it is used. You must have CREATEROLE privilege or be a database superuser to use this command.

Note that roles are defined at the system-level and are valid for all databases in your Greenplum Database system.

Parameters

- **name**
  The name of the new role.

- **SUPERUSER**
  - NOSUPERUSER
    - If SUPERUSER is specified, the role being defined will be a superuser, who can override all access restrictions within the database. Superuser status is dangerous and should be used only when really needed. You must yourself be a superuser to create a new superuser.
    - NOSUPERUSER is the default.

- **CREATEDB**
  - NOCREATEDB
    - If CREATEDB is specified, the role being defined will be allowed to create new databases.
    - NOCREATEDB (the default) will deny a role the ability to create databases.

- **CREATEROLE**
  - NOCREATEROLE
    - If CREATEROLE is specified, the role being defined will be allowed to create new roles, alter other roles, and drop other roles. NOCREATEROLE (the default) will deny a role the ability to create roles or modify roles other than their own.

- **CREATEUSER**
  - NOCREATEUSER
    - These clauses are obsolete, but still accepted, spellings of SUPERUSER and NOSUPERUSER. Note that they are not equivalent to the CREATEROLE and NOCREATEROLE clauses.

- **CREATEEXTTABLE**
  - NOCREATEEXTTABLE
    - If CREATEEXTTABLE is specified, the role being defined is allowed to create external tables. The default type is readable and the default protocol is gpfdist if not specified. NOCREATEEXTTABLE (the default) denies the role the ability to create external
tables. Note that external tables that use the file or execute protocols can only be created by superusers.

**INHERIT**

**NOINHERIT**

If specified, INHERIT (the default) allows the role to use whatever database privileges have been granted to all roles it is directly or indirectly a member of. With NOINHERIT, membership in another role only grants the ability to SET ROLE to that other role.

**LOGIN**

**NOLOGIN**

If specified, LOGIN allows a role to log in to a database. A role having the LOGIN attribute can be thought of as a user. Roles with NOLOGIN (the default) are useful for managing database privileges, and can be thought of as groups.

**CONNECTION LIMIT**

`conlimit`

The number maximum of concurrent connections this role can make. The default of -1 means there is no limitation.

**PASSWORD**

`password`

Sets the user password for roles with the LOGIN attribute. If you do not plan to use password authentication you can omit this option. If no password is specified, the password will be set to null and password authentication will always fail for that user. A null password can optionally be written explicitly as PASSWORD NULL.

**ENCRYPTED**

**UNENCRYPTED**

These key words control whether the password is stored encrypted in the system catalogs. (If neither is specified, the default behavior is determined by the configuration parameter `password_encryption`.) If the presented password string is already in MD5-encrypted format, then it is stored encrypted as-is, regardless of whether ENCRYPTED or UNENCRYPTED is specified (since the system cannot decrypt the specified encrypted password string). This allows reloading of encrypted passwords during dump/restore.

Note that older clients may lack support for the MD5 authentication mechanism that is needed to work with passwords that are stored encrypted.

**VALID UNTIL**

`timestamp`

The VALID UNTIL clause sets a date and time after which the role’s password is no longer valid. If this clause is omitted the password will never expire.

**IN ROLE**

`rolename`

Adds the new role as a member of the named roles. Note that there is no option to add the new role as an administrator; use a separate GRANT command to do that.

**ROLE**

`rolename`

Adds the named roles as members of this role, making this new role a group.

**ADMIN**

`rolename`

The ADMIN clause is like ROLE, but the named roles are added to the new role WITH ADMIN OPTION, giving them the right to grant membership in this role to others.

**RESOURCE GROUP**

`group_name`

The name of the resource group to assign to the new role. The role will be subject to the concurrent transaction, memory, and CPU limits configured for the resource group. You can assign a single resource group to one or more roles.

If you do not specify a resource group for a new role, the role is automatically assigned the default resource group for the role’s capability, admin_group for SUPERUSER roles, default_group for non-admin roles.
You can assign the admin_group resource group to any role having the SUPERUSER attribute.

You can assign the default_group resource group to any role.

Warning: Resource groups are an experimental feature and are not intended for use in a production environment. Experimental features are subject to change without notice in future releases.

**RESOURCE QUEUE queue_name**

The name of the resource queue to which the new user-level role is to be assigned. Only roles with LOGIN privilege can be assigned to a resource queue. The special keyword NONE means that the role is assigned to the default resource queue. A role can only belong to one resource queue.

Roles with the SUPERUSER attribute are exempt from resource queue limits. For a superuser role, queries always run immediately regardless of limits imposed by an assigned resource queue.

**DENY deny_point**

**DENY BETWEEN deny_point AND deny_point**

The DENY and DENY BETWEEN keywords set time-based constraints that are enforced at login. DENY sets a day or a day and time to deny access. DENY BETWEEN sets an interval during which access is denied. Both use the parameter deny_point that has the following format:

```
DAY day [ TIME 'time' ]
```

The two parts of the deny_point parameter use the following formats:

For **day**:

```
{'Sunday' | 'Monday' | 'Tuesday' |'Wednesday' | 'Thursday' | 'Friday' | 'Saturday' | 0-6 }
```

For **time**:

```
{ 00-23 : 00-59 | 01-12 : 00-59 { AM | PM }}
```

The **DENY BETWEEN** clause uses two deny_point parameters:

```
DENY BETWEEN deny_point AND deny_point
```

For more information and examples about time-based constraints, see "Managing Roles and Privileges" in the Greenplum Database Administrator Guide.

**Notes**

The preferred way to add and remove role members (manage groups) is to use **GRANT** and **REVOKE**.

The **VALID UNTIL** clause defines an expiration time for a password only, not for the role. The expiration time is not enforced when logging in using a non-password-based authentication method.

The **INHERIT** attribute governs inheritance of grantable privileges (access privileges for database objects and role memberships). It does not apply to the special role attributes set by **CREATE ROLE** and **ALTER ROLE**. For example, being a member of a role with CREATEDB privilege does not immediately grant the ability to create databases, even if **INHERIT** is set. These privileges/attributes are never inherited: SUPERUSER, CREATEDB, CREATEROLE, CREATEEXTTABLE, LOGIN, RESOURCE GROUP, and RESOURCE QUEUE. The attributes must be set on each user-level role.
The **INHERIT** attribute is the default for reasons of backwards compatibility. In prior releases of Greenplum Database, users always had access to all privileges of groups they were members of. However, **NOINHERIT** provides a closer match to the semantics specified in the SQL standard.

Be careful with the **CREATEROLE** privilege. There is no concept of inheritance for the privileges of a **CREATEROLE**-role. That means that even if a role does not have a certain privilege but is allowed to create other roles, it can easily create another role with different privileges than its own (except for creating roles with superuser privileges). For example, if a role has the **CREATEROLE** privilege but not the **CREATEDB** privilege, it can create a new role with the **CREATEDB** privilege. Therefore, regard roles that have the **CREATEROLE** privilege as almost-supuser-roles.

The **CONNECTION LIMIT** option is never enforced for superusers.

Caution must be exercised when specifying an unencrypted password with this command. The password will be transmitted to the server in clear-text, and it might also be logged in the client's command history or the server log. The client program **createuser**, however, transmits the password encrypted. Also, psql contains a command **\password** that can be used to safely change the password later.

**Examples**

Create a role that can log in, but don’t give it a password:

```
CREATE ROLE jonathan LOGIN;
```

Create a role that belongs to a resource queue:

```
CREATE ROLE jonathan LOGIN RESOURCE QUEUE poweruser;
```

Create a role with a password that is valid until the end of 2016 (**CREATE USER** is the same as **CREATE ROLE** except that it implies **LOGIN**):

```
CREATE USER joelle WITH PASSWORD 'jw8s0F4' VALID UNTIL '2017-01-01';
```

Create a role that can create databases and manage other roles:

```
CREATE ROLE admin WITH CREATEDB CREATEROLE;
```

Create a role that does not allow login access on Sundays:

```
CREATE ROLE user3 DENY DAY 'Sunday';
```

Create a role, assigning a resource group:

```
CREATE ROLE bill RESOURCE GROUP rg_light;
```

**Compatibility**

The SQL standard defines the concepts of users and roles, but it regards them as distinct concepts and leaves all commands defining users to be specified by the database implementation. In Greenplum Database users and roles are unified into a single type of object. Roles therefore have many more optional attributes than they do in the standard.

**CREATE ROLE** is in the SQL standard, but the standard only requires the syntax:

```
CREATE ROLE name [WITH ADMIN rolename]
```

Allowing multiple initial administrators, and all the other options of **CREATE ROLE**, are Greenplum Database extensions.
The behavior specified by the SQL standard is most closely approximated by giving users the \texttt{NOINHERIT} attribute, while roles are given the \texttt{INHERIT} attribute.

**See Also**

\texttt{SET ROLE, ALTER ROLE, DROP ROLE, GRANT, REVOKE, CREATE RESOURCE QUEUE CREATE RESOURCE GROUP}

## CREATE RULE

Defines a new rewrite rule.

### Synopsis

```
CREATE [OR REPLACE] RULE name AS ON event
   TO table [WHERE condition]
   DO [ALSO | INSTEAD] { NOTHING | command | (command; command
                      ...} }
```

### Description

\texttt{CREATE RULE} defines a new rule applying to a specified table or view. \texttt{CREATE OR REPLACE RULE} will either create a new rule, or replace an existing rule of the same name for the same table.

The Greenplum Database rule system allows one to define an alternate action to be performed on insertions, updates, or deletions in database tables. A rule causes additional or alternate commands to be executed when a given command on a given table is executed. Rules can be used on views as well. It is important to realize that a rule is really a command transformation mechanism, or command macro. The transformation happens before the execution of the commands starts. It does not operate independently for each physical row as does a trigger.

\texttt{ON SELECT} rules must be unconditional \texttt{INSTEAD} rules and must have actions that consist of a single \texttt{SELECT} command. Thus, an \texttt{ON SELECT} rule effectively turns the table into a view, whose visible contents are the rows returned by the rule's \texttt{SELECT} command rather than whatever had been stored in the table (if anything). It is considered better style to write a \texttt{CREATE VIEW} command than to create a real table and define an \texttt{ON SELECT} rule for it.

You can create the illusion of an updatable view by defining \texttt{ON INSERT}, \texttt{ON UPDATE}, and \texttt{ON DELETE} rules to replace update actions on the view with appropriate updates on other tables.

There is a catch if you try to use conditional rules for view updates: there must be an unconditional \texttt{INSTEAD} rule for each action you wish to allow on the view. If the rule is conditional, or is not \texttt{INSTEAD}, then the system will still reject attempts to perform the update action, because it thinks it might end up trying to perform the action on the dummy table of the view in some cases. If you want to handle all the useful cases in conditional rules, add an unconditional \texttt{DO INSTEAD NOTHING} rule to ensure that the system understands it will never be called on to update the dummy table. Then make the conditional rules non-\texttt{INSTEAD}; in the cases where they are applied, they add to the default \texttt{INSTEAD NOTHING} action. (This method does not currently work to support \texttt{RETURNING} queries, however.)

### Parameters

- **name**
  The name of a rule to create. This must be distinct from the name of any other rule for the same table. Multiple rules on the same table and same event type are applied in alphabetical name order.

- **event**
  The event is one of \texttt{SELECT}, \texttt{INSERT}, \texttt{UPDATE}, or \texttt{DELETE}.

- **table**
The name (optionally schema-qualified) of the table or view the rule applies to.

**condition**

Any SQL conditional expression (returning boolean). The condition expression may not refer to any tables except NEW and OLD, and may not contain aggregate functions. NEW and OLD refer to values in the referenced table. NEW is valid in ON INSERT and ON UPDATE rules to refer to the new row being inserted or updated. OLD is valid in ON UPDATE and ON DELETE rules to refer to the existing row being updated or deleted.

**INSTEAD**

INSTEAD indicates that the commands should be executed instead of the original command.

**ALSO**

ALSO indicates that the commands should be executed in addition to the original command. If neither ALSO nor INSTEAD is specified, ALSO is the default.

**command**

The command or commands that make up the rule action. Valid commands are SELECT, INSERT, UPDATE, or DELETE. The special table names NEW and OLD may be used to refer to values in the referenced table. NEW is valid in ON INSERT and ON UPDATE rules to refer to the new row being inserted or updated. OLD is valid in ON UPDATE and ON DELETE rules to refer to the existing row being updated or deleted.

**Notes**

You must be the owner of a table to create or change rules for it.

It is very important to take care to avoid circular rules. Recursive rules are not validated at rule create time, but will report an error at execution time.

**Examples**

Create a rule that inserts rows into the child table b2001 when a user tries to insert into the partitioned parent table rank:

```
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);
```

**Compatibility**

CREATE RULE is a Greenplum Database language extension, as is the entire query rewrite system.

**See Also**

DROP RULE, CREATE TABLE, CREATE VIEW

---

**CREATE SCHEMA**

Defines a new schema.

**Synopsis**

```
CREATE SCHEMA schema_name [AUTHORIZATION username] [schema_element [ ... ]]
CREATE SCHEMA AUTHORIZATION rolename [schema_element [ ... ]]
```
Description

CREATE SCHEMA enters a new schema into the current database. The schema name must be distinct from the name of any existing schema in the current database.

A schema is essentially a namespace: it contains named objects (tables, data types, functions, and operators) whose names may duplicate those of other objects existing in other schemas. Named objects are accessed either by qualifying their names with the schema name as a prefix, or by setting a search path that includes the desired schema(s). A CREATE command specifying an unqualified object name creates the object in the current schema (the one at the front of the search path, which can be determined with the function current_schema).

Optionally, CREATE SCHEMA can include subcommands to create objects within the new schema. The subcommands are treated essentially the same as separate commands issued after creating the schema, except that if the AUTHORIZATION clause is used, all the created objects will be owned by that role.

Parameters

schema_name
The name of a schema to be created. If this is omitted, the user name is used as the schema name. The name cannot begin with pg_, as such names are reserved for system catalog schemas.

rolename
The name of the role who will own the schema. If omitted, defaults to the role executing the command. Only superusers may create schemas owned by roles other than themselves.

schema_element
An SQL statement defining an object to be created within the schema. Currently, only CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE SEQUENCE, CREATE TRIGGER and GRANT are accepted as clauses within CREATE SCHEMA. Other kinds of objects may be created in separate commands after the schema is created.

Note: Greenplum Database does not support triggers.

Notes
To create a schema, the invoking user must have the CREATE privilege for the current database or be a superuser.

Examples
Create a schema:

CREATE SCHEMA myschema;

Create a schema for role joe (the schema will also be named joe):

CREATE SCHEMA AUTHORIZATION joe;

Compatibility
The SQL standard allows a DEFAULT CHARACTER SET clause in CREATE SCHEMA, as well as more subcommand types than are presently accepted by Greenplum Database.

The SQL standard specifies that the subcommands in CREATE SCHEMA may appear in any order. The present Greenplum Database implementation does not handle all cases of forward references in subcommands; it may sometimes be necessary to reorder the subcommands in order to avoid forward references.
According to the SQL standard, the owner of a schema always owns all objects within it. Greenplum Database allows schemas to contain objects owned by users other than the schema owner. This can happen only if the schema owner grants the `CREATE` privilege on the schema to someone else.

**See Also**

`ALTER SCHEMA, DROP SCHEMA`

---

**CREATE SEQUENCE**

Defines a new sequence generator.

**Synopsis**

```
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 }]
```

**Description**

`CREATE SEQUENCE` creates a new sequence number generator. This involves creating and initializing a new special single-row table. The generator will be owned by the user issuing the command.

If a schema name is given, then the sequence is created in the specified schema. Otherwise it is created in the current schema. Temporary sequences exist in a special schema, so a schema name may not be given when creating a temporary sequence. The sequence name must be distinct from the name of any other sequence, table, index, or view in the same schema.

After a sequence is created, you use the `nextval()` function to operate on the sequence. For example, to insert a row into a table that gets the next value of a sequence:

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

You can also use the function `setval()` to operate on a sequence, but only for queries that do not operate on distributed data. For example, the following query is allowed because it resets the sequence counter value for the sequence generator process on the master:

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

But the following query will be rejected in Greenplum Database because it operates on distributed data:

```
INSERT INTO product VALUES (setval('myserial', 201), 'gizmo');
```

In a regular (non-distributed) database, functions that operate on the sequence go to the local sequence table to get values as they are needed. In Greenplum Database, however, keep in mind that each segment is its own distinct database process. Therefore the segments need a single point of truth to go for sequence values so that all segments get incremented correctly and the sequence moves forward in the right order. A sequence server process runs on the master and is the point-of-truth for a sequence in a Greenplum distributed database. Segments get sequence values at runtime from the master.

Because of this distributed sequence design, there are some limitations on the functions that operate on a sequence in Greenplum Database:

- `lastval()` and `currval()` functions are not supported.
- `setval()` can only be used to set the value of the sequence generator on the master, it cannot be used in subqueries to update records on distributed table data.
- `nextval()` sometimes grabs a block of values from the master for a segment to use, depending on the query. So values may sometimes be skipped in the sequence if all of the block turns out not to be needed at the segment level. Note that a regular PostgreSQL database does this too, so this is not something unique to Greenplum Database.

Although you cannot update a sequence directly, you can use a query like:

```sql
SELECT * FROM sequence_name;
```

to examine the parameters and current state of a sequence. In particular, the `last_value` field of the sequence shows the last value allocated by any session.

**Parameters**

**TEMPORARY | TEMP**

If specified, the sequence object is created only for this session, and is automatically dropped on session exit. Existing permanent sequences with the same name are not visible (in this session) while the temporary sequence exists, unless they are referenced with schema-qualified names.

**name**

The name (optionally schema-qualified) of the sequence to be created.

**increment**

Specifies which value is added to the current sequence value to create a new value. A positive value will make an ascending sequence, a negative one a descending sequence. The default value is 1.

**minvalue**

**NO MINVALUE**

Determines the minimum value a sequence can generate. If this clause is not supplied or **NO MINVALUE** is specified, then defaults will be used. The defaults are 1 and -263-1 for ascending and descending sequences, respectively.

**maxvalue**

**NO MAXVALUE**

Determines the maximum value for the sequence. If this clause is not supplied or **NO MAXVALUE** is specified, then default values will be used. The defaults are 263-1 and -1 for ascending and descending sequences, respectively.

**start**

Allows the sequence to begin anywhere. The default starting value is `minvalue` for ascending sequences and `maxvalue` for descending ones.

**cache**

Specifies how many sequence numbers are to be preallocated and stored in memory for faster access. The minimum (and default) value is 1 (no cache).

**CYCLE | NO CYCLE**

Allows the sequence to wrap around when the `maxvalue` (for ascending) or `minvalue` (for descending) has been reached. If the limit is reached, the next number generated will be the `minvalue` (for ascending) or `maxvalue` (for descending). If **NO CYCLE** is specified, any calls to `nextval()` after the sequence has reached its maximum value will return an error. If not specified, **NO CYCLE** is the default.

**OWNED BY table.column**

**OWNED BY NONE**
Causes the sequence to be associated with a specific table column, such that if that column (or its whole table) is dropped, the sequence will be automatically dropped as well. The specified table must have the same owner and be in the same schema as the sequence. OWNED BY NONE, the default, specifies that there is no such association.

Notes
Sequences are based on bigint arithmetic, so the range cannot exceed the range of an eight-byte integer (-9223372036854775808 to 9223372036854775807).

Although multiple sessions are guaranteed to allocate distinct sequence values, the values may be generated out of sequence when all the sessions are considered. For example, session A might reserve values 1..10 and return nextval=1, then session B might reserve values 11..20 and return nextval=11 before session A has generated nextval=2. Thus, you should only assume that the nextval() values are all distinct, not that they are generated purely sequentially. Also, last_value will reflect the latest value reserved by any session, whether or not it has yet been returned by nextval().

Examples
Create a sequence named myseq:

```
CREATE SEQUENCE myseq START 101;
```

Insert a row into a table that gets the next value of the sequence named idseq:

```
INSERT INTO distributors VALUES (nextval('idseq'), 'acme');
```

Reset the sequence counter value on the master:

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

Illegal use of setval() in Greenplum Database (setting sequence values on distributed data):

```
INSERT INTO product VALUES (setval('myseq', 201), 'gizmo');
```

Compatibility
CREATE SEQUENCE conforms to the SQL standard, with the following exceptions:

- The AS data_type expression specified in the SQL standard is not supported.
- Obtaining the next value is done using the nextval() function instead of the NEXT VALUE FOR expression specified in the SQL standard.
- The OWNED BY clause is a Greenplum Database extension.

See Also
ALTER SEQUENCE, DROP SEQUENCE

CREATE TABLE

Defines a new table.

Note: Referential integrity syntax (foreign key constraints) is accepted but not enforced.

Synopsis

```
CREATE [[GLOBAL | LOCAL] {TEMPORARY | TEMP}] TABLE table_name (  
  [ { column_name data_type [ DEFAULT default_expr ] }
```
column_constraint is:

[CONSTRAINT constraint_name]
NOT NULL | NULL
| UNIQUE [USING INDEX TABLESPACE tablespace]
[WITH ( FILLFACTOR = value )]
| PRIMARY KEY [USING INDEX TABLESPACE tablespace]
[WITH ( FILLFACTOR = value )]
| CHECK ( expression )
| REFERENCES table_name [ ( column_name [, ... ] ) ]
| [ key_match_type ]
| [ key_action ]

storage_directive for a column is:

COMPRESSTYPE={ZLIB | QUICKLZ | RLE_TYPE | NONE}
[COMPRESSLEVEL={0-9} ]
[BLOCKSIZE={8192-2097152} ]

storage_parameter for the table is:

APPENDONLY={TRUE|FALSE}
BLOCKSIZE={8192-2097152}
ORIENTATION={COLUMN|ROW}
CHECKSUM={TRUE|FALSE}
COMPRESSTYPE={ZLIB|QUICKLZ|RLE_TYPE|NONE}
COMPRESSLEVEL={0-9}
FILLFACTOR={10-100}
OIDS [=TRUE|FALSE]

and table_constraint is:

[CONSTRAINT constraint_name]
UNIQUE ( column_name [, ... ] )
[USING INDEX TABLESPACE tablespace]
[WITH ( FILLFACTOR=value )]
| PRIMARY KEY ( column_name [, ... ] )
| [USING INDEX TABLESPACE tablespace]
| [WITH ( FILLFACTOR=value )]
| CHECK ( expression )
| FOREIGN KEY ( column_name [, ... ] )
| REFERENCES table_name ( column_name [, ... ] )
| [ key_match_type ]
| [ key_action ]
| [ key_checking_mode ]

where key_match_type is:

MATCH FULL
| SIMPLE

where key_action is:

ON DELETE
| ON UPDATE
| NO ACTION
| RESTRICT
| CASCADE
| SET NULL
| SET DEFAULT

where key_checking_mode is:

DEFERRABLE
| NOT DEFERRABLE
| INITIALLY DEFERRED
| INITIALLY IMMEDIATE

where partition_type is:

LIST
| RANGE

where partition_specification is:

partition_element [, ...]

and partition_element is:

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 [, ... ] ) ]
| [ TABLESPACE tablespace ]

where subpartition_spec or template_spec is:

subpartition_element [, ...]
and subpartition_element is:

```
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 [, ...])]
| [TABLESPACE tablespace]
```

where storage_parameter for a partition is:

```
APPENDONLY={TRUE|FALSE}
BLOCKSIZE={8192-2097152}
ORIENTATION={COLUMN|ROW}
CHECKSUM={TRUE|FALSE}
COMPRESSTYPE={ZLIB|QUICKLZ|RLE_TYPE|NONE}
COMPRESSLEVEL={1-9}
FILLFACTOR={10-100}
OIDS [=TRUE|FALSE]
```

**Description**

`CREATE TABLE` creates an initially empty table in the current database. The user who issues the command owns the table.

If you specify a schema name, Greenplum creates the table in the specified schema. Otherwise Greenplum creates the table in the current schema. Temporary tables exist in a special schema, so you cannot specify a schema name when creating a temporary table. Table names must be distinct from the name of any other table, external table, sequence, index, or view in the same schema.

The optional constraint clauses specify conditions that new or updated rows must satisfy for an insert or update operation to succeed. A constraint is an SQL object that helps define the set of valid values in the table in various ways. Constraints apply to tables, not to partitions. You cannot add a constraint to a partition or subpartition.

Referential integrity constraints (foreign keys) are accepted but not enforced. The information is kept in the system catalogs but is otherwise ignored.

There are two ways to define constraints: table constraints and column constraints. A column constraint is defined as part of a column definition. A table constraint definition is not tied to a particular column, and it can encompass more than one column. Every column constraint can also be written as a table constraint; a column constraint is only a notational convenience for use when the constraint only affects one column.

When creating a table, there is an additional clause to declare the Greenplum Database distribution policy. If a `DISTRIBUTED BY` or `DISTRIBUTED RANDOMLY` clause is not supplied, then Greenplum assigns a hash distribution policy to the table using either the `PRIMARY KEY` (if the table has one) or the first column of the table as the distribution key. Columns of geometric or user-defined data types are not eligible as Greenplum distribution key columns. If a table does not have a column of an eligible data type, the rows are distributed based on a round-robin or random distribution. To ensure an even distribution of data in your Greenplum Database system, you want to choose a distribution key that is unique for each record, or if that is not possible, then choose `DISTRIBUTED RANDOMLY`.

The `PARTITION BY` clause allows you to divide the table into multiple sub-tables (or parts) that, taken together, make up the parent table and share its schema. Though the sub-tables exist as independent tables, the Greenplum Database restricts their use in important ways. Internally, partitioning is implemented as a special form of inheritance. Each child table partition is created with a distinct
CHECK constraint which limits the data the table can contain, based on some defining criteria. The CHECK constraints are also used by the query optimizer to determine which table partitions to scan in order to satisfy a given query predicate. These partition constraints are managed automatically by the Greenplum Database.

Parameters

**GLOBAL | LOCAL**

These keywords are present for SQL standard compatibility, but have no effect in Greenplum Database.

**TEMPORARY | TEMP**

If specified, the table is created as a temporary table. Temporary tables are automatically dropped at the end of a session, or optionally at the end of the current transaction (see **ON COMMIT**). Existing permanent tables with the same name are not visible to the current session while the temporary table exists, unless they are referenced with schema-qualified names. Any indexes created on a temporary table are automatically temporary as well.

**table_name**

The name (optionally schema-qualified) of the table to be created.

**column_name**

The name of a column to be created in the new table.

**data_type**

The data type of the column. This may include array specifiers.

For table columns that contain textual data, Specify the data type **VARCHAR** or **TEXT**. Specifying the data type **CHAR** is not recommended. In Greenplum Database, the data types **VARCHAR** or **TEXT** handles padding added to the data (space characters added after the last non-space character) as significant characters, the data type **CHAR** does not. See **Notes**.

**DEFAULT default_expr**

The **DEFAULT** clause assigns a default data value for the column whose column definition it appears within. The value is any variable-free expression (subqueries and cross-references to other columns in the current table are not allowed). The data type of the default expression must match the data type of the column. The default expression will be used in any insert operation that does not specify a value for the column. If there is no default for a column, then the default is null.

**ENCODING ( storage_directive [, ...] )**

For a column, the optional **ENCODING** clause specifies the type of compression and block size for the column data. See **storage_options** for **COMPRESSTYPE**, **COMPRESSLEVEL**, and **BLOCKSIZE** values.

The clause is valid only for append-optimized, column-oriented tables.

Column compression settings are inherited from the table level to the partition level to the subpartition level. The lowest-level settings have priority.

**INHERITS**

The optional **INHERITS** clause specifies a list of tables from which the new table automatically inherits all columns. Use of **INHERITS** creates a persistent relationship between the new child table and its parent table(s). Schema modifications to the parent(s) normally propagate to children as well, and by default the data of the child table is included in scans of the parent(s).

In Greenplum Database, the **INHERITS** clause is not used when creating partitioned tables. Although the concept of inheritance is used in partition hierarchies, the inheritance structure of a partitioned table is created using the **PARTITION BY** clause.
If the same column name exists in more than one parent table, an error is reported unless the data types of the columns match in each of the parent tables. If there is no conflict, then the duplicate columns are merged to form a single column in the new table. If the column name list of the new table contains a column name that is also inherited, the data type must likewise match the inherited column(s), and the column definitions are merged into one. However, inherited and new column declarations of the same name need not specify identical constraints: all constraints provided from any declaration are merged together and all are applied to the new table. If the new table explicitly specifies a default value for the column, this default overrides any defaults from inherited declarations of the column. Otherwise, any parents that specify default values for the column must all specify the same default, or an error will be reported.

**LIKE other_table [{INCLUDING | EXCLUDING} {DEFAULTS | CONSTRAINTS}]**

The LIKE clause specifies a table from which the new table automatically copies all column names, data types, not-null constraints, and distribution policy. Storage properties like append-optimized or partition structure are not copied. Unlike **INHERITS**, the new table and original table are completely decoupled after creation is complete.

Default expressions for the copied column definitions will only be copied if **INCLUDING DEFAULTS** is specified. The default behavior is to exclude default expressions, resulting in the copied columns in the new table having null defaults.

Not-null constraints are always copied to the new table. **CHECK** constraints will only be copied if **INCLUDING CONSTRAINTS** is specified; other types of constraints will never be copied. Also, no distinction is made between column constraints and table constraints — when constraints are requested, all check constraints are copied.

Note also that unlike **INHERITS**, copied columns and constraints are not merged with similarly named columns and constraints. If the same name is specified explicitly or in another LIKE clause an error is signalled.

**CONSTRAINT constraint_name**

An optional name for a column or table constraint. If the constraint is violated, the constraint name is present in error messages, so constraint names like *column must be positive* can be used to communicate helpful constraint information to client applications. (Double-quotes are needed to specify constraint names that contain spaces.) If a constraint name is not specified, the system generates a name.

**Note:** The specified **constraint_name** is used for the constraint, but a system-generated unique name is used for the index name. In some prior releases, the provided name was used for both the constraint name and the index name.

**NULL | NOT NULL**

Specifies if the column is or is not allowed to contain null values. **NULL** is the default.

**UNIQUE ( column constraint )**

**UNIQUE ( column_name [, ... ] ) ( table constraint )**

The **UNIQUE** constraint specifies that a group of one or more columns of a table may contain only unique values. The behavior of the unique table constraint is the same as that for column constraints, with the additional capability to span multiple columns. For the purpose of a unique constraint, null values are not considered equal. The column(s) that are unique must contain all the columns of the Greenplum distribution key. In addition, the <key> must contain all the columns in the partition key if the table is partitioned. Note that a <key> constraint in a partitioned table is not the same as a simple **UNIQUE INDEX**.

For information about unique constraint management and limitations, see **Notes**.

**PRIMARY KEY ( column constraint )**

**PRIMARY KEY ( column_name [, ... ] ) ( table constraint )**
The primary key constraint specifies that a column or columns of a table may contain only unique (non-duplicate), non-null values. Technically, `PRIMARY KEY` is merely a combination of `UNIQUE` and `NOT NULL`, but identifying a set of columns as primary key also provides metadata about the design of the schema, as a primary key implies that other tables may rely on this set of columns as a unique identifier for rows. For a table to have a primary key, it must be hash distributed (not randomly distributed), and the primary key The column(s) that are unique must contain all the columns of the Greenplum distribution key. In addition, the `<key>` must contain all the columns in the partition key if the table is partitioned. Note that a `<key>` constraint in a partitioned table is not the same as a simple `UNIQUE INDEX`.

For information about primary key management and limitations, see Notes.

**CHECK (expression)**

The `CHECK` clause specifies an expression producing a Boolean result which new or updated rows must satisfy for an insert or update operation to succeed. Expressions evaluating to `TRUE` or `UNKNOWN` succeed. Should any row of an insert or update operation produce a `FALSE` result an error exception is raised and the insert or update does not alter the database. A check constraint specified as a column constraint should reference that column’s value only, while an expression appearing in a table constraint may reference multiple columns. `CHECK` expressions cannot contain subqueries nor refer to variables other than columns of the current row.

**REFERENCES table_name [ ( column_name [, .. ] ) ]**

[ key_match_type ] [ key_action ]

**FOREIGN KEY ( column_name [, .. ] ) REFERENCES table_name [ ( column_name [, .. ] ) ]**

[ key_match_type ] [ key_action [ key_checking_mode ]

The `REFERENCES` and `FOREIGN KEY` clauses specify referential integrity constraints (foreign key constraints). Greenplum accepts referential integrity constraints as specified in PostgreSQL syntax but does not enforce them. See the PostgreSQL documentation for information about referential integrity constraints.

**WITH (storage_option=value)**

The `WITH` clause can be used to set storage options for the table or its indexes. Note that you can also set storage parameters on a particular partition or subpartition by declaring the `WITH` clause in the partition specification. The lowest-level settings have priority.

The defaults for some of the table storage options can be specified with the server configuration parameter `gp_default_storage_options`. For information about setting default storage options, see Notes.

The following storage options are available:

**APPENDONLY** — Set to `TRUE` to create the table as an append-optimized table. If `FALSE` or not declared, the table will be created as a regular heap-storage table.

**BLOCKSIZE** — Set to the size, in bytes for each block in a table. The `BLOCKSIZE` must be between 8192 and 2097152 bytes, and be a multiple of 8192. The default is 32768.

**ORIENTATION** — Set to `column` for column-oriented storage, or `row` (the default) for row-oriented storage. This option is only valid if `APPENDONLY=TRUE`. Heap-storage tables can only be row-oriented.

**CHECKSUM** — This option is valid only for append-optimized tables (`APPENDONLY=TRUE`). The value `TRUE` is the default and enables CRC checksum validation for append-optimized tables. The checksum is calculated during block creation and is stored on disk. Checksum validation is performed during block reads. If the checksum calculated during the read does not match the stored checksum, the transaction is aborted. If you set the value to `FALSE` to disable checksum validation, checking the table data for on-disk corruption will not be performed.
COMPRESSSTYPE — Set to ZLIB (the default), RLE-TYPE, or QUICKLZ¹ to specify the type of compression used. The value NONE disables compression. QuickLZ uses less CPU power and compresses data faster at a lower compression ratio than zlib. Conversely, zlib provides more compact compression ratios at lower speeds. This option is only valid if APPENDONLY=TRUE.

Note: ¹QuickLZ compression is available only in the commercial release of Pivotal Greenplum Database.

The value RLE TYPE is supported only if ORIENTATION=column is specified, Greenplum Database uses the run-length encoding (RLE) compression algorithm. RLE compresses data better than the zlib or QuickLZ compression algorithm when the same data value occurs in many consecutive rows.

For columns of type BIGINT, INTEGER, DATE, TIME, or TIMESTAMPT, delta compression is also applied if the COMPRESSSTYPE option is set to RLE-TYPE compression. The delta compression algorithm is based on the delta between column values in consecutive rows and is designed to improve compression when data is loaded in sorted order or the compression is applied to column data that is in sorted order.

For information about using table compression, see "Choosing the Table Storage Model" in the Greenplum Database Administrator Guide.

COMPRESSLEVEL — For zlib compression of append-optimized tables, set to an integer value between 1 (fastest compression) to 9 (highest compression ratio). QuickLZ compression level can only be set to 1. If not declared, the default is 1. For RLE_TYPE, the compression level can be set an integer value between 1 (fastest compression) to 4 (highest compression ratio).

This option is valid only if APPENDONLY=TRUE.

FILLFACTOR — See CREATE INDEX for more information about this index storage parameter.

OIDS — Set to OIDS=FALSE (the default) so that rows do not have object identifiers assigned to them. Greenplum strongly recommends that you do not enable OIDS when creating a table. On large tables, such as those in a typical Greenplum Database system, using OIDs for table rows can cause wrap-around of the 32-bit OID counter. Once the counter wraps around, OIDs can no longer be assumed to be unique, which not only makes them useless to user applications, but can also cause problems in the Greenplum Database system catalog tables. In addition, excluding OIDs from a table reduces the space required to store the table on disk by 4 bytes per row, slightly improving performance. OIDS are not allowed on partitioned tables or append-optimized column-oriented tables.

ON COMMIT

The behavior of temporary tables at the end of a transaction block can be controlled using ON COMMIT. The three options are:

PRESEERVE ROWS - No special action is taken at the ends of transactions for temporary tables. This is the default behavior.

DELETE ROWS - All rows in the temporary table will be deleted at the end of each transaction block. Essentially, an automatic TRUNCATE is done at each commit.

DROP - The temporary table will be dropped at the end of the current transaction block.

TABLESPACE tablespace

The name of the tablespace in which the new table is to be created. If not specified, the database’s default tablespace is used.

USING INDEX TABLESPACE tablespace
This clause allows selection of the tablespace in which the index associated with a UNIQUE or PRIMARY KEY constraint will be created. If not specified, the database's default tablespace is used.

**DISTRIBUTED BY (column, [ ... ])
DISTRIBUTED RANDOMLY**

Used to declare the Greenplum Database distribution policy for the table. DISTRIBUTED BY uses hash distribution with one or more columns declared as the distribution key. For the most even data distribution, the distribution key should be the primary key of the table or a unique column (or set of columns). If that is not possible, then you may choose DISTRIBUTED RANDOMLY, which will send the data round-robin to the segment instances.

The Greenplum Database server configuration parameter `gp_create_table_random_default_distribution` controls the default table distribution policy if the DISTRIBUTED BY clause is not specified when you create a table. Greenplum Database follows these rules to create a table if a distribution policy is not specified.

If the value of the parameter is off (the default), Greenplum Database chooses the table distribution key based on the command. If the LIKE or INHERITS clause is specified in table creation command, the created table uses the same distribution key as the source or parent table.

If the value of the parameter is set to on, Greenplum Database follows these rules:

- If PRIMARY KEY or UNIQUE columns are not specified, the distribution of the table is random (DISTRIBUTED RANDOMLY). Table distribution is random even if the table creation command contains the LIKE or INHERITS clause.
- If PRIMARY KEY or UNIQUE columns are specified, a DISTRIBUTED BY clause must also be specified. If a DISTRIBUTED BY clause is not specified as part of the table creation command, the command fails.

For information about the parameter, see “Server Configuration Parameters.”

**PARTITION BY**

Declares one or more columns by which to partition the table.

When creating a partitioned table, Greenplum Database creates the root partitioned table (the root partition) with the specified table name. Greenplum Database also creates a hierarchy of tables, child tables, that are the subpartitions based on the partitioning options that you specify. The Greenplum Database `pg_partition*` system views contain information about the subpartition tables.

For each partition level (each hierarchy level of tables), a partitioned table can have a maximum of 32,767 partitions.

**Note:** Greenplum Database stores partitioned table data in the leaf child tables, the lowest-level tables in the hierarchy of child tables for use by the partitioned table.

**partition_type**

Declares partition type: LIST (list of values) or RANGE (a numeric or date range).

**partition_specification**

Declares the individual partitions to create. Each partition can be defined individually or, for range partitions, you can use the EVERY clause (with a START and optional END clause) to define an increment pattern to use to create the individual partitions.

**DEFAULT PARTITION name** — Declares a default partition. When data does not match to an existing partition, it is inserted into the default partition. Partition designs that do not have a default partition will reject incoming rows that do not match to an existing partition.
**PARTITION** name — Declares a name to use for the partition. Partitions are created using the following naming convention: `parentname_level#_prt_givenname`.

**VALUES** — For list partitions, defines the value(s) that the partition will contain.

**START** — For range partitions, defines the starting range value for the partition. By default, start values are INCLUSIVE. For example, if you declared a start date of '2016-01-01', then the partition would contain all dates greater than or equal to '2016-01-01'. Typically the data type of the **START** expression is the same type as the partition key column. If that is not the case, then you must explicitly cast to the intended data type.

**END** — For range partitions, defines the ending range value for the partition. By default, end values are EXCLUSIVE. For example, if you declared an end date of '2016-02-01', then the partition would contain all dates less than but not equal to '2016-02-01'. Typically the data type of the **END** expression is the same type as the partition key column. If that is not the case, then you must explicitly cast to the intended data type.

**EVERY** — For range partitions, defines how to increment the values from **START** to **END** to create individual partitions. Typically the data type of the **EVERY** expression is the same type as the partition key column. If that is not the case, then you must explicitly cast to the intended data type.

**WITH** — Sets the table storage options for a partition. For example, you may want older partitions to be append-optimized tables and newer partitions to be regular heap tables.

**TABLESPACE** — The name of the tablespace in which the partition is to be created.

**SUBPARTITION** BY

Declares one or more columns by which to subpartition the first-level partitions of the table. The format of the subpartition specification is similar to that of a partition specification described above.

**SUBPARTITION** TEMPLATE

Instead of declaring each subpartition definition individually for each partition, you can optionally declare a subpartition template to be used to create the subpartitions (lower level child tables). This subpartition specification would then apply to all parent partitions.

**Notes**

- In Greenplum Database (a Postgres-based system) the data types `VARCHAR` or `TEXT` handles padding added to the textual data (space characters added after the last non-space character) as significant characters, the data type `CHAR` does not.

  In Greenplum Database, values of type `CHAR(n)` are padded with trailing spaces to the specified width `n`. The values are stored and displayed with the spaces. However, the padding spaces are treated as semantically insignificant. When the values are distributed, the trailing spaces are disregarded. The trailing spaces are also treated as semantically insignificant when comparing two values of data type `CHAR`, and the trailing spaces are removed when converting a character value to one of the other string types.

- Using OIDs in new applications is not recommended: where possible, using a `SERIAL` or other sequence generator as the table's primary key is preferred. However, if your application does make use of OIDs to identify specific rows of a table, it is recommended to create a unique constraint on the OID column of that table, to ensure that OIDs in the table will indeed uniquely identify rows even after counter wrap-around. Avoid assuming that OIDs are unique across tables; if you need a database-wide unique identifier, use the combination of table OID and row OID for the purpose.

- Greenplum Database has some special conditions for primary key and unique constraints with regards to columns that are the *distribution key* in a Greenplum table. For a unique constraint to be enforced in Greenplum Database, the table must be hash-distributed (not `DISTRIBUTED RANDOMLY`), and the constraint columns must be the same as (or a superset of) the table's distribution key columns. Also, the distribution key must be a left-subset of the constraint columns with the columns in the correct
order. For example, if the primary key is (a,b,c), the distribution key can be only one of the following: (a), (a,b), or (a,b,c).

A primary key constraint is simply a combination of a unique constraint and a not-null constraint. Greenplum Database automatically creates a `UNIQUE` index for each `UNIQUE` or `PRIMARY` key constraint to enforce uniqueness. Thus, it is not necessary to create an index explicitly for primary key columns. `UNIQUE` and `PRIMARY` key constraints are not allowed on append-optimized tables because the `UNIQUE` indexes that are created by the constraints are not allowed on append-optimized tables.

Foreign key constraints are not supported in Greenplum Database.

For inherited tables, unique constraints, primary key constraints, indexes and table privileges are not inherited in the current implementation.

- For append-optimized tables, `UPDATE` and `DELETE` are not allowed in a serializable transaction and will cause the transaction to abort. `CLUSTER`, `DECLARE...FORUPDATE`, and triggers are not supported with append-optimized tables.
- To insert data into a partitioned table, you specify the root partitioned table, the table created with the `CREATE TABLE` command. You also can specify a leaf child table of the partitioned table in an `INSERT` command. An error is returned if the data is not valid for the specified leaf child table. Specifying a child table that is not a leaf child table in the `INSERT` command is not supported. Execution of other DML commands such as `UPDATE` and `DELETE` on any child table of a partitioned table is not supported. These commands must be executed on the root partitioned table, the table created with the `CREATE TABLE` command.
- The default values for these table storage options can be specified with the server configuration parameter `gp_default_storage_option`.
  - `APPENDONLY`
  - `BLOCKSIZE`
  - `CHECKSUM`
  - `COMPRESSTYPE`
  - `COMPRESSLEVEL`
  - `ORIENTATION`

The defaults can be set for the system, a database, or a user. For information about setting storage options, see the server configuration parameter `gp_default_storage_options`.

**Important:** The current Greenplum Database legacy optimizer allows list partitions with multi-column (composite) partition keys. GPORCA does not support composite keys, so using composite partition keys is not recommended.

**Examples**

Create a table named `rank` in the schema named `baby` and distribute the data using the columns `rank`, `gender`, and `year`:

```sql
CREATE TABLE baby.rank (id int, rank int, year smallint, gender char(1), count int ) DISTRIBUTED BY (rank, gender, year);
```

Create table `films` and table `distributors` (the primary key will be used as the Greenplum distribution key by default):

```sql
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
```

```sql
CREATE TABLE distributors (code        char(5) CONSTRAINT distkey PRIMARY KEY, name       varchar(50) NOT NULL, phone      varchar(20) NOT NULL, address    varchar(100), city       varchar(50), country    varchar(50)) DISTRIBUTED BY (code);
```
CREATE TABLE distributors (did integer PRIMARY KEY DEFAULT nextval('serial'), name varchar(40) NOT NULL CHECK (name <> '')
);

Create a gzip-compressed, append-optimized table:

CREATE TABLE sales (txn_id int, qty int, date date)
WITH (appendonly=true, compresslevel=5)
DISTRIBUTED BY (txn_id);

Create a three level partitioned table using subpartition templates and default partitions at each level:

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);

Compatibility

CREATE TABLE command conforms to the SQL standard, with the following exceptions:

- **Temporary Tables** — In the SQL standard, temporary tables are defined just once and automatically exist (starting with empty contents) in every session that needs them. Greenplum Database instead requires each session to issue its own CREATE TEMPORARY TABLE command for each temporary table to be used. This allows different sessions to use the same temporary table name for different purposes, whereas the standard's approach constrains all instances of a given temporary table name to have the same table structure.

  The standard's distinction between global and local temporary tables is not in Greenplum Database. Greenplum Database will accept the GLOBAL and LOCAL keywords in a temporary table declaration, but they have no effect.

  If the ON COMMIT clause is omitted, the SQL standard specifies that the default behavior as ON COMMIT DELETE ROWS. However, the default behavior in Greenplum Database is ON COMMIT PRESERVE ROWS. The ON COMMIT DROP option does not exist in the SQL standard.

- **Column Check Constraints** — The SQL standard says that CHECK column constraints may only refer to the column they apply to; only CHECK table constraints may refer to multiple columns. Greenplum Database does not enforce this restriction; it treats column and table check constraints alike.

- **NULL Constraint** — The NULL constraint is a Greenplum Database extension to the SQL standard that is included for compatibility with some other database systems (and for symmetry with the NOT NULL constraint). Since it is the default for any column, its presence is not required.
• **Inheritance** — Multiple inheritance via the INHERITS clause is a Greenplum Database language extension. SQL:1999 and later define single inheritance using a different syntax and different semantics. SQL:1999-style inheritance is not yet supported by Greenplum Database.

• **Partitioning** — Table partitioning via the PARTITION BY clause is a Greenplum Database language extension.

• **Zero-column tables** — Greenplum Database allows a table of no columns to be created (for example, CREATE TABLE foo();). This is an extension from the SQL standard, which does not allow zero-column tables. Zero-column tables are not in themselves very useful, but disallowing them creates odd special cases for ALTER TABLE DROP COLUMN, so Greenplum decided to ignore this spec restriction.

• **WITH clause** — The WITH clause is a Greenplum Database extension; neither storage parameters nor OIDs are in the standard.

• **Tablespaces** — The Greenplum Database concept of tablespaces is not part of the SQL standard. The clauses TABLESPACE and USING INDEX TABLESPACE are extensions.

• **Data Distribution** — The Greenplum Database concept of a parallel or distributed database is not part of the SQL standard. The DISTRIBUTED clauses are extensions.

**See Also**

ALTER TABLE, DROP TABLE, CREATE EXTERNAL TABLE, CREATE TABLE AS

**CREATE TABLE AS**

Defines a new table from the results of a query.

**Synopsis**

```
CREATE [ [GLOBAL | LOCAL] {TEMPORARY | TEMP} ] TABLE table_name
    [(column_name [, ...] )]
    [ WITH (storage_parameter=value [, ... ] ) ]
    [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP}]
    [TABLESPACE tablespace]
    AS query
    [DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY]
```

where `storage_parameter` is:

- APPENDONLY={TRUE|FALSE}
- BLOCKSIZE={8192-2097152}
- ORIENTATION={COLUMN|ROW}
- COMPRESSTYPE={ZLIB|QUICKLZ}
- COMPRESSLEVEL={1-9 | 1}
- FILLFACTOR={10-100}
- OIDS=[TRUE|FALSE]

**Description**

CREATE TABLE AS creates a table and fills it with data computed by a SELECT command. The table columns have the names and data types associated with the output columns of the SELECT, however you can override the column names by giving an explicit list of new column names.

CREATE TABLE AS creates a new table and evaluates the query just once to fill the new table initially. The new table will not track subsequent changes to the source tables of the query.

**Parameters**

GLOBAL | LOCAL

These keywords are present for SQL standard compatibility, but have no effect in Greenplum Database.
TEMPORARY | TEMP
If specified, the new table is created as a temporary table. Temporary tables are automatically dropped at the end of a session, or optionally at the end of the current transaction (see ON COMMIT). Existing permanent tables with the same name are not visible to the current session while the temporary table exists, unless they are referenced with schema-qualified names. Any indexes created on a temporary table are automatically temporary as well.

**table_name**
The name (optionally schema-qualified) of the new table to be created.

**column_name**
The name of a column in the new table. If column names are not provided, they are taken from the output column names of the query. If the table is created from an EXECUTE command, a column name list cannot be specified.

**WITH ( storage_parameter=value )**
The WITH clause can be used to set storage options for the table or its indexes. Note that you can also set different storage parameters on a particular partition or subpartition by declaring the WITH clause in the partition specification. The following storage options are available:

- **APPENDONLY** — Set to TRUE to create the table as an append-optimized table. If FALSE or not declared, the table will be created as a regular heap-storage table.
- **BLOCKSIZE** — Set to the size, in bytes for each block in a table. The BLOCKSIZE must be between 8192 and 2097152 bytes, and be a multiple of 8192. The default is 32768.
- **ORIENTATION** — Set to column for column-oriented storage, or row (the default) for row-oriented storage. This option is only valid if APPENDONLY=TRUE. Heap-storage tables can only be row-oriented.
- **COMPRESSTYPE** — Set to ZLIB (the default) or QUICKLZ to specify the type of compression used. QuickLZ uses less CPU power and compresses data faster at a lower compression ratio than zlib. Conversely, zlib provides more compact compression ratios at lower speeds. This option is only valid if APPENDONLY=TRUE.

**Note:** QuickLZ compression is available only in the commercial release of Pivotal Greenplum Database.

- **COMPRESSLEVEL** — For zlib compression of append-optimized tables, set to a value between 1 (fastest compression) to 9 (highest compression ratio). QuickLZ compression level can only be set to 1. If not declared, the default is 1. This option is only valid if APPENDONLY=TRUE.

- **FILLFACTOR** — See CREATE INDEX for more information about this index storage parameter.

- **OIDS** — Set to OIDS=FALSE (the default) so that rows do not have object identifiers assigned to them. Greenplum strongly recommends that you do not enable OIDS when creating a table. On large tables, such as those in a typical Greenplum Database system, using OIDs for table rows can cause wrap-around of the 32-bit OID counter. Once the counter wraps around, OIDs can no longer be assumed to be unique, which not only makes them useless to user applications, but can also cause problems in the Greenplum Database system catalog tables. In addition, excluding OIDs from a table reduces the space required to store the table on disk by 4 bytes per row, slightly improving performance. OIDS are not allowed on column-oriented tables.

**ON COMMIT**
The behavior of temporary tables at the end of a transaction block can be controlled using ON COMMIT. The three options are:
PRESERVE ROWS — No special action is taken at the ends of transactions for temporary tables. This is the default behavior.

DELETE ROWS — All rows in the temporary table will be deleted at the end of each transaction block. Essentially, an automatic TRUNCATE is done at each commit.

DROP — The temporary table will be dropped at the end of the current transaction block.

**TABLESPACE tablespace**

The tablespace is the name of the tablespace in which the new table is to be created. If not specified, the database's default tablespace is used.

**AS query**

A SELECT or VALUES command, or an EXECUTE command that runs a prepared SELECT or VALUES query.

**DISTRIBUTED BY (column, [ ... ] ) DISTRIBUTED RANDOMLY**

Used to declare the Greenplum Database distribution policy for the table. DISTRIBUTED BY uses hash distribution with one or more columns declared as the distribution key. For the most even data distribution, the distribution key should be the primary key of the table or a unique column (or set of columns). If that is not possible, then you may choose DISTRIBUTED RANDOMLY, which will send the data round-robin to the segment instances.

The Greenplum Database server configuration parameter gp_create_table_random_default_distribution controls the default table distribution policy if the DISTRIBUTED BY clause is not specified when you create a table. Greenplum Database follows these rules to create a table if a distribution policy is not specified.

- If the legacy query optimizer creates the table, and the value of the parameter is off, the table distribution policy is determined based on the command.
- If the legacy query optimizer creates the table, and the value of the parameter is on, the table distribution policy is random.
- If GPORCA creates the table, the table distribution policy is random. The parameter value has no affect.

For information about the parameter, see “Server Configuration Parameters.” For information about the legacy query optimizer and GPORCA, see “Querying Data” in the Greenplum Database Administrator Guide.

**Notes**

This command is functionally similar to SELECT INTO, but it is preferred since it is less likely to be confused with other uses of the SELECT INTO syntax. Furthermore, CREATE TABLE AS offers a superset of the functionality offered by SELECT INTO.

CREATE TABLE AS can be used for fast data loading from external table data sources. See CREATE EXTERNAL TABLE.

**Examples**

Create a new table films_recent consisting of only recent entries from the table films:

```sql
CREATE TABLE films_recent AS SELECT * FROM films WHERE date_prod >= '2007-01-01';
```

Create a new temporary table films_recent, consisting of only recent entries from the table films, using a prepared statement. The new table has OIDs and will be dropped at commit:

```sql
PREPARE recentfilms(date) AS SELECT * FROM films WHERE
```
**Compatibility**

CREATE TABLE AS conforms to the SQL standard, with the following exceptions:

- The standard requires parentheses around the subquery clause; in Greenplum Database, these parentheses are optional.
- The standard defines a WITH [NO] DATA clause; this is not currently implemented by Greenplum Database. The behavior provided by Greenplum Database is equivalent to the standard's WITH DATA case. WITH NO DATA can be simulated by appending LIMIT 0 to the query.
- Greenplum Database handles temporary tables differently from the standard; see CREATE TABLE for details.
- The WITH clause is a Greenplum Database extension; neither storage parameters nor OIDs are in the standard.
- The Greenplum Database concept of tablespaces is not part of the standard. The TABLESPACE clause is an extension.

**See Also**

CREATE EXTERNAL TABLE, CREATE EXTERNAL TABLE, EXECUTE, SELECT, SELECT INTO, VALUES

**CREATE TABLESPACE**

Defines a new tablespace.

**Synopsis**

```
cREATE TABLESPACE tablespace_name [OWNER username]
    FILESPACE filespace_name
```

**Description**

CREATE TABLESPACE registers a new tablespace for your Greenplum Database system. The tablespace name must be distinct from the name of any existing tablespace in the system.

A tablespace allows superusers to define an alternative location on the file system where the data files containing database objects (such as tables and indexes) may reside.

A user with appropriate privileges can pass a tablespace name to CREATE DATABASE, CREATE TABLE, or CREATE INDEX to have the data files for these objects stored within the specified tablespace.

In Greenplum Database, there must be a file system location defined for the master, each primary segment, and each mirror segment in order for the tablespace to have a location to store its objects across an entire Greenplum system. This collection of file system locations is defined in a filespace object. A filespace must be defined before you can create a tablespace. See gpfilespace in the Greenplum Database Utility Guide for more information.

**Parameters**

- **tablespace_name**
  The name of a tablespace to be created. The name cannot begin with pg_ or gp_, as such names are reserved for system tablespaces.

- **OWNER username**
The name of the user who will own the tablespace. If omitted, defaults to the user executing the command. Only superusers may create tablespaces, but they can assign ownership of tablespaces to non-superusers.

FILESPACE

The name of a Greenplum Database filespace that was defined using the gpfilespace management utility.

Notes

You must first create a filespace to be used by the tablespace. See gpfilespace in the Greenplum Database Utility Guide for more information.

Tablespaces are only supported on systems that support symbolic links.

CREATE TABLESPACE cannot be executed inside a transaction block.

Examples

Create a new tablespace by specifying the corresponding filespace to use:

```
CREATE TABLESPACE mytblspace FILESPACE myfilespace;
```

Compatibility

CREATE TABLESPACE is a Greenplum Database extension.

See Also

CREATE DATABASE, CREATE TABLE, CREATE INDEX, DROP TABLESPACE, ALTER TABLESPACE, gpfilespace in the Greenplum Database Utility Guide

CREATE TYPE

Defines a new data type.

Synopsis

```
CREATE TYPE name AS ( attribute_name data_type [, ... ] )
CREATE TYPE name AS ENUM ( 'label' [, ... ] )
CREATE TYPE name
```
## Description

`CREATE TYPE` registers a new data type for use in the current database. The user who defines a type becomes its owner.

If a schema name is given then the type is created in the specified schema. Otherwise it is created in the current schema. The type name must be distinct from the name of any existing type or domain in the same schema. The type name must also be distinct from the name of any existing table in the same schema.

### Composite Types

The first form of `CREATE TYPE` creates a composite type. The composite type is specified by a list of attribute names and data types. This is essentially the same as the row type of a table, but using `CREATE TYPE` avoids the need to create an actual table when all that is wanted is to define a type. A stand-alone composite type is useful as the argument or return type of a function.

### Enumerated Types

The second form of `CREATE TYPE` creates an enumerated (ENUM) type, as described in *Enumerated Types* in the PostgreSQL documentation. Enum types take a list of one or more quoted labels, each of which must be less than `NAMEDATALEN` bytes long (64 in a standard build).

### Base Types

The third form of `CREATE TYPE` creates a new base type (scalar type). The parameters may appear in any order, not only that shown in the syntax, and most are optional. You must register two or more functions (using `CREATE FUNCTION`) before defining the type. The support functions `input_function` and `output_function` are required, while the functions `receive_function`, `send_function`, `type_modifier_input_function`, `type_modifier_output_function`, and `analyze_function` are optional. Generally these functions have to be coded in C or another low-level language. In Greenplum Database, any function used to implement a data type must be defined as `IMMUTABLE`.

The `input_function` converts the type's external textual representation to the internal representation used by the operators and functions defined for the type. `output_function` performs the reverse transformation. The input function may be declared as taking one argument of type `cstring`, or as taking three arguments of types `cstring`, `oid`, `integer`. The first argument is the input text as a C string, the second argument is the type's own OID (except for array types, which instead receive their element type's OID), and the third is the `typmod` of the destination column, if known (-1 will be passed if not). The input function must return a value of the data type itself. Usually, an input function should be declared `STRICT`; if it is not, it will be called with a `NULL` first parameter when reading a `NULL` input value. The function must still return `NULL` in this case, unless it raises an error. (This case is mainly meant to support domain input functions, which may need to reject `NULL` inputs.) The output function must be declared as taking one argument of the new data type. The output function must return type `cstring`. Output functions are not invoked for `NULL` values.

The optional `receive_function` converts the type's external binary representation to the internal representation used by the operators and functions defined for the type. `send_function` performs the reverse transformation. The receive function must be declared as taking one argument of type `internal`, or as taking three arguments of types `internal`, `oid`, `integer`. The first argument is a pointer to a `StringInfo` buffer holding the received byte string; the optional arguments are the same as for the text input function. The receive function must return a value of the data type itself. Usually, a receive function should be declared `STRICT`; if it is not, it will be called with a `NULL` first parameter when reading a `NULL` input value. The function must still return `NULL` in this case, unless it raises an error. (This case is mainly meant to support domain receive functions, which may need to reject `NULL` inputs.) Similarly, the optional `send_function` converts from the internal representation to the external binary representation. If this function is not supplied, the type cannot participate in binary output. The send function must be declared as taking one argument of the new data type. The send function must return type `bytea`. Send functions are not invoked for `NULL` values.
The optional type_modifier_input_function and type_modifier_output_function are required if the type supports modifiers. Modifiers are optional constraints attached to a type declaration, such as char(5) or numeric(30,2). While Greenplum Database allows user-defined types to take one or more simple constants or identifiers as modifiers, this information must fit into a single non-negative integer value for storage in the system catalogs. Greenplum Database passes the declared modifier(s) to the type_modifier_input_function in the form of a cstring array. The modifier input function must check the values for validity, throwing an error if they are incorrect. If the values are correct, the modifier input function returns a single non-negative integer value that Greenplum Database stores as the column typmod. Type modifiers are rejected if the type was not defined with a type_modifier_input_function. The type_modifier_output_function converts the internal integer typmod value back to the correct form for user display. The modifier output function must return a cstring value that is the exact string to append to the type name. For example, numeric's function might return (30,2). The type_modifier_output_function is optional. When not specified, the default display format is the stored typmod integer value enclosed in parentheses.

You should at this point be wondering how the input and output functions can be declared to have results or arguments of the new type, when they have to be created before the new type can be created. The answer is that the type should first be defined as a shell type, which is a placeholder type that has no properties except a name and an owner. This is done by issuing the command CREATE TYPE name, with no additional parameters. Then the I/O functions can be defined referencing the shell type. Finally, CREATE TYPE with a full definition replaces the shell entry with a complete, valid type definition, after which the new type can be used normally.

While the details of the new type's internal representation are only known to the I/O functions and other functions you create to work with the type, there are several properties of the internal representation that must be declared to Greenplum Database. Foremost of these is internallength. Base data types can be fixed-length, in which case internallength is a positive integer, or variable length, indicated by setting internallength to VARIABLE. (Internally, this is represented by setting typlen to -1.) The internal representation of all variable-length types must start with a 4-byte integer giving the total length of this value of the type.

The optional flag PASSEDBYVALUE indicates that values of this data type are passed by value, rather than by reference. You may not pass by value types whose internal representation is larger than the size of the Datum type (4 bytes on most machines, 8 bytes on a few).

The alignment parameter specifies the storage alignment required for the data type. The allowed values equate to alignment on 1, 2, 4, or 8 byte boundaries. Note that variable-length types must have an alignment of at least 4, since they necessarily contain an int4 as their first component.

The storage parameter allows selection of storage strategies for variable-length data types. (Only plain is allowed for fixed-length types.) plain specifies that data of the type will always be stored in-line and not compressed. extended specifies that the system will first try to compress a long data value, and will move the value out of the main table row if it's still too long. external allows the value to be moved out of the main table, but the system will not try to compress it. main allows compression, but discourages moving the value out of the main table. (Data items with this storage strategy may still be moved out of the main table if there is no other way to make a row fit, but they will be kept in the main table preferentially over extended and external items.)

A default value may be specified, in case a user wants columns of the data type to default to something other than the null value. Specify the default with the DEFAULT keyword. (Such a default may be overridden by an explicit DEFAULT clause attached to a particular column.)

To indicate that a type is an array, specify the type of the array elements using the ELEMENT keyword. For example, to define an array of 4-byte integers (int4), specify ELEMENT = int4. More details about array types appear below.

To indicate the delimiter to be used between values in the external representation of arrays of this type, delimiter can be set to a specific character. The default delimiter is the comma (,). Note that the delimiter is associated with the array element type, not the array type itself.

Array Types
Whenever a user-defined base data type is created, Greenplum Database automatically creates an associated array type, whose name consists of the base type's name prepended with an underscore. The parser understands this naming convention, and translates requests for columns of type `foo[]` into requests for type `_foo`. The implicitly-created array type is variable length and uses the built-in input and output functions `array_in` and `array_out`.

You might reasonably ask why there is an `ELEMENT` option, if the system makes the correct array type automatically. The only case where it's useful to use `ELEMENT` is when you are making a fixed-length type that happens to be internally an array of a number of identical things, and you want to allow these things to be accessed directly by subscripting, in addition to whatever operations you plan to provide for the type as a whole. For example, type `name` allows its constituent `char` elements to be accessed this way. A 2-D point type could allow its two component numbers to be accessed like `point[0]` and `point[1]`. Note that this facility only works for fixed-length types whose internal form is exactly a sequence of identical fixed-length fields. A subscriptable variable-length type must have the generalized internal representation used by `array_in` and `array_out`. For historical reasons, subscripting of fixed-length array types starts from zero, rather than from one as for variable-length arrays.

**Parameters**

`name`  
The name (optionally schema-qualified) of a type to be created.

`attribute_name`  
The name of an attribute (column) for the composite type.

`data_type`  
The name of an existing data type to become a column of the composite type.

`label`  
A string literal representing the textual label associated with one value of an enum type.

`input_function`  
The name of a function that converts data from the type's external textual form to its internal form.

`output_function`  
The name of a function that converts data from the type's internal form to its external textual form.

`receive_function`  
The name of a function that converts data from the type's external binary form to its internal form.

`send_function`  
The name of a function that converts data from the type's internal form to its external binary form.

`type_modifier_input_function`  
The name of a function that converts an array of modifier(s) for the type to internal form.

`type_modifier_output_function`  
The name of a function that converts the internal form of the type's modifier(s) to external textual form.

`internallength`  
A numeric constant that specifies the length in bytes of the new type's internal representation. The default assumption is that it is variable-length.

`alignment`  
The storage alignment requirement of the data type. Must be one of `char`, `int2`, `int4`, or `double`. The default is `int4`. 
**storage**

The storage strategy for the data type. Must be one of plain, external, extended, or main. The default is plain.

**default**

The default value for the data type. If this is omitted, the default is null.

**element**

The type being created is an array; this specifies the type of the array elements.

**delimiter**

The delimiter character to be used between values in arrays made of this type.

### Notes

User-defined type names cannot begin with the underscore character (_) and can only be 62 characters long (or in general \text{NAMEDATALEN} - 2, rather than the \text{NAMEDATALEN} - 1 characters allowed for other names). Type names beginning with underscore are reserved for internally-created array type names.

Because there are no restrictions on use of a data type once it's been created, creating a base type is tantamount to granting public execute permission on the functions mentioned in the type definition. (The creator of the type is therefore required to own these functions.) This is usually not an issue for the sorts of functions that are useful in a type definition. But you might want to think twice before designing a type in a way that would require 'secret' information to be used while converting it to or from external form.

Before Greenplum Database version 2.4, the syntax `CREATE TYPE` \texttt{name} did not exist. The way to create a new base type was to create its input function first. In this approach, Greenplum Database will first see the name of the new data type as the return type of the input function. The shell type is implicitly created in this situation, and then it can be referenced in the definitions of the remaining I/O functions. This approach still works, but is deprecated and may be disallowed in some future release. Also, to avoid accidentally cluttering the catalogs with shell types as a result of simple typos in function definitions, a shell type will only be made this way when the input function is written in C.

### Examples

This example creates a composite type and uses it in a function definition:

```sql
CREATE TYPE compfoo AS (f1 int, f2 text);
CREATE FUNCTION getfoo() RETURNS SETOF compfoo AS $$
  SELECT fooid, fooname FROM foo
$$ LANGUAGE SQL;
```

This example creates the enumerated type \texttt{mood} and uses it in a table definition.

```sql
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)
```

This example creates the base data type \texttt{box} and then uses the type in a table definition:

```sql
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
);

If the internal structure of box were an array of four float4 elements, we might instead use:

CREATE TYPE box (
    INTERNALLENGTH = 16,
    INPUT = my_box_in_function,
    OUTPUT = my_box_out_function,
    ELEMENT = float4
);

which would allow a box value’s component numbers to be accessed by subscripting. Otherwise the type behaves the same as before.

This example creates a large object type and uses it in a table definition:

CREATE TYPE bigobj (
    INPUT = lo_filein, OUTPUT = lo_fileout,
    INTERNALLENGTH = VARIABLE
);
CREATE TABLE big_objs (
    id integer,
    obj bigobj
);

Compatibility
This CREATE TYPE command is a Greenplum Database extension. There is a CREATE TYPE statement in the SQL standard that is rather different in detail.

See Also
CREATE FUNCTION, ALTER TYPE, DROP TYPE, CREATE DOMAIN

CREATE USER
Defines a new database role with the LOGIN privilege by default.

Synopsis
CREATE USER name [[WITH] option [ ... ]]


where option can be:

```
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
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]
```

### Description

**CREATE USER** is an alias for **CREATE ROLE**.

The **CREATE USER** statement in the SQL standard. **CREATE USER** is that **LOGIN** is assumed by default with **CREATE USER**, whereas **NOLOGIN** is assumed by default with **CREATE ROLE**.

### Compatibility

There is no **CREATE USER** statement in the SQL standard.

### See Also

**CREATE ROLE**

**CREATE VIEW**

Defines a new view.

### Synopsis

```
CREATE [OR REPLACE] [TEMP | TEMPORARY] VIEW name
[ ( column_name [, ... ] ) ]
AS query
```

### Description

**CREATE VIEW** defines a view of a query. The view is not physically materialized. Instead, the query is run every time the view is referenced in a query.

**CREATE OR REPLACE VIEW** is similar, but if a view of the same name already exists, it is replaced. You can only replace a view with a new query that generates the identical set of columns (same column names and data types).

If a schema name is given then the view is created in the specified schema. Otherwise it is created in the current schema. Temporary views exist in a special schema, so a schema name may not be given when
creating a temporary view. The name of the view must be distinct from the name of any other view, table, sequence, or index in the same schema.

**Parameters**

**TEMPORARY | TEMP**

If specified, the view is created as a temporary view. Temporary views are automatically dropped at the end of the current session. Existing permanent relations with the same name are not visible to the current session while the temporary view exists, unless they are referenced with schema-qualified names. If any of the tables referenced by the view are temporary, the view is created as a temporary view (whether TEMPORARY is specified or not).

**name**

The name (optionally schema-qualified) of a view to be created.

**column_name**

An optional list of names to be used for columns of the view. If not given, the column names are deduced from the query.

**query**

A SELECT or VALUES command which will provide the columns and rows of the view.

**Notes**

Views in Greenplum Database are read only. The system will not allow an insert, update, or delete on a view. You can get the effect of an updatable view by creating rewrite rules on the view into appropriate actions on other tables. For more information see CREATE RULE.

Be careful that the names and data types of the view's columns will be assigned the way you want. For example:

```sql
CREATE VIEW vista AS SELECT 'Hello World';
```

is bad form in two ways: the column name defaults to `?column?`, and the column data type defaults to `unknown`. If you want a string literal in a view's result, use something like:

```sql
CREATE VIEW vista AS SELECT text 'Hello World' AS hello;
```

Access to tables referenced in the view is determined by permissions of the view owner not the current user (even if the current user is a superuser). This can be confusing in the case of superusers, since superusers typically have access to all objects. In the case of a view, even superusers must be explicitly granted access to tables referenced in the view if they are not the owner of the view.

However, functions called in the view are treated the same as if they had been called directly from the query using the view. Therefore the user of a view must have permissions to call any functions used by the view.

If you create a view with an ORDER BY clause, the ORDER BY clause is ignored when you do a SELECT from the view.

**Examples**

Create a view consisting of all comedy films:

```sql
CREATE VIEW comedies AS SELECT * FROM films WHERE kind = 'comedy';
```
Create a view that gets the top ten ranked baby names:

```
CREATE VIEW topten AS SELECT name, rank, gender, year FROM names, rank WHERE rank < '11' AND names.id=rank.id;
```

**Compatibility**

The SQL standard specifies some additional capabilities for the `CREATE VIEW` statement that are not in Greenplum Database. The optional clauses for the full SQL command in the standard are:

- **CHECK OPTION** — This option has to do with updatable views. All `INSERT` and `UPDATE` commands on the view will be checked to ensure data satisfy the view-defining condition (that is, the new data would be visible through the view). If they do not, the update will be rejected.
- **LOCAL** — Check for integrity on this view.
- **CASCADED** — Check for integrity on this view and on any dependent view. `CASCADED` is assumed if neither `CASCADED` nor `LOCAL` is specified.

`CREATE OR REPLACE VIEW` is a Greenplum Database language extension. So is the concept of a temporary view.

**See Also**

`SELECT, DROP VIEW`

**DEALLOCATE**

Deallocates a prepared statement.

**Synopsis**

```
DEALLOCATE [PREPARE] name
```

**Description**

`DEALLOCATE` is used to deallocate a previously prepared SQL statement. If you do not explicitly deallocate a prepared statement, it is deallocated when the session ends.

For more information on prepared statements, see `PREPARE`.

**Parameters**

- **PREPARE**
  
  Optional key word which is ignored.

- **name**
  
  The name of the prepared statement to deallocate.

**Examples**

Deallocated the previously prepared statement named `insert_names`:

```
DEALLOCATE insert_names;
```

**Compatibility**

The SQL standard includes a `DEALLOCATE` statement, but it is only for use in embedded SQL.
See Also
EXECUTE, PREPARE

DECLARE

Defines a cursor.

Synopsis

```
DECLARE name [BINARY] [INSENSITIVE] [NO SCROLL] CURSOR
[WITH | WITHOUT] HOLD
FOR query [FOR READ ONLY]
```

Description

DECLARE allows a user to create cursors, which can be used to retrieve a small number of rows at a time out of a larger query. Cursors can return data either in text or in binary format using FETCH.

Normal cursors return data in text format, the same as a SELECT would produce. Since data is stored natively in binary format, the system must do a conversion to produce the text format. Once the information comes back in text form, the client application may need to convert it to a binary format to manipulate it. In addition, data in the text format is often larger in size than in the binary format. Binary cursors return the data in a binary representation that may be more easily manipulated. Nevertheless, if you intend to display the data as text anyway, retrieving it in text form will save you some effort on the client side.

As an example, if a query returns a value of one from an integer column, you would get a string of 1 with a default cursor whereas with a binary cursor you would get a 4-byte field containing the internal representation of the value (in big-endian byte order).

Binary cursors should be used carefully. Many applications, including psql, are not prepared to handle binary cursors and expect data to come back in the text format.

Note:

When the client application uses the 'extended query' protocol to issue a FETCH command, the Bind protocol message specifies whether data is to be retrieved in text or binary format. This choice overrides the way that the cursor is defined. The concept of a binary cursor as such is thus obsolete when using extended query protocol — any cursor can be treated as either text or binary.

A cursor can be specified in the WHERE CURRENT OF clause of the UPDATE or DELETE statement to update or delete table data. The UPDATE or DELETE statement can only be executed on the server, for example in an interactive psql session or a script. Language extensions such as PL/pgSQL do not have support for updatable cursors.

Parameters

name

The name of the cursor to be created.

BINARY

Causes the cursor to return data in binary rather than in text format.

INSENSITIVE

Indicates that data retrieved from the cursor should be unaffected by updates to the tables underlying the cursor while the cursor exists. In Greenplum Database, all cursors are insensitive. This key word currently has no effect and is present for compatibility with the SQL standard.

NO SCROLL
A cursor cannot be used to retrieve rows in a nonsequential fashion. This is the default behavior in Greenplum Database, since scrollable cursors (SCROLL) are not supported.

**WITH HOLD**

**WITHOUT HOLD**

**WITH HOLD** specifies that the cursor may continue to be used after the transaction that created it successfully commits. **WITHOUT HOLD** specifies that the cursor cannot be used outside of the transaction that created it. **WITHOUT HOLD** is the default.

**WITH HOLD** cannot be specified when the query includes a FOR UPDATE or FOR SHARE clause.

**query**

A SELECT or VALUES command which will provide the rows to be returned by the cursor.

If the cursor is used in the WHERE CURRENT OF clause of the UPDATE or DELETE command, the SELECT command must satisfy the following conditions:

- Cannot reference a view or external table.
- References only one table.

The table must be updatable. For example, the following are not updatable: table functions, set-returning functions, append-only tables, columnar tables.

- Cannot contain any of the following:
  - A grouping clause
  - A set operation such as UNION ALL or UNION DISTINCT
  - A sorting clause
  - A windowing clause
  - A join or a self-join

Specifying the FOR UPDATE clause in the SELECT command prevents other sessions from changing the rows between the time they are fetched and the time they are updated. Without the FOR UPDATE clause, a subsequent use of the UPDATE or DELETE command with the WHERE CURRENT OF clause has no effect if the row was changed since the cursor was created.

**Note:** Specifying the FOR UPDATE clause in the SELECT command locks the entire table, not just the selected rows.

**FOR READ ONLY**

FOR READ ONLY indicates that the cursor is used in a read-only mode.

**Notes**

Unless **WITH HOLD** is specified, the cursor created by this command can only be used within the current transaction. Thus, DECLARE without **WITH HOLD** is useless outside a transaction block: the cursor would survive only to the completion of the statement. Therefore Greenplum Database reports an error if this command is used outside a transaction block. Use BEGIN, COMMIT and ROLLBACK to define a transaction block.

If **WITH HOLD** is specified and the transaction that created the cursor successfully commits, the cursor can continue to be accessed by subsequent transactions in the same session. (But if the creating transaction is aborted, the cursor is removed.) A cursor created with **WITH HOLD** is closed when an explicit CLOSE command is issued on it, or the session ends. In the current implementation, the rows represented by a held cursor are copied into a temporary file or memory area so that they remain available for subsequent transactions.

If you create a cursor with the DECLARE command in a transaction, you cannot use the SET command in the transaction until you close the cursor with the CLOSE command.
Scrollable cursors are not currently supported in Greenplum Database. You can only use FETCH to move the cursor position forward, not backwards.

DECLARE...FOR UPDATE is not supported with append-optimized tables.

You can see all available cursors by querying the pg_cursors system view.

Examples

Declare a cursor:

```
DECLARE mycursor CURSOR FOR SELECT * FROM mytable;
```

Compatibility

SQL standard allows cursors only in embedded SQL and in modules. Greenplum Database permits cursors to be used interactively.

Greenplum Database does not implement an OPEN statement for cursors. A cursor is considered to be open when it is declared.

The SQL standard allows cursors to move both forward and backward. All Greenplum Database cursors are forward moving only (not scrollable).

Binary cursors are a Greenplum Database extension.

See Also

CLOSE, DELETE, FETCH, MOVE, SELECT, UPDATE

DELETE

Deletes rows from a table.

Synopsis

```
DELETE FROM [ONLY] table [[AS] alias]
[USING usinglist]
[WHERE condition | WHERE CURRENT OF cursor_name ]
```

Description

DELETE deletes rows that satisfy the WHERE clause from the specified table. If the WHERE clause is absent, the effect is to delete all rows in the table. The result is a valid, but empty table.

By default, DELETE will delete rows in the specified table and all its child tables. If you wish to delete only from the specific table mentioned, you must use the ONLY clause.

There are two ways to delete rows in a table using information contained in other tables in the database: using sub-selects, or specifying additional tables in the USING clause. Which technique is more appropriate depends on the specific circumstances.

If the WHERE CURRENT OF clause is specified, the row that is deleted is the one most recently fetched from the specified cursor.

You must have the DELETE privilege on the table to delete from it.

Outputs
On successful completion, a `DELETE` command returns a command tag of the form

```
DELETE count
```

The count is the number of rows deleted. If count is 0, no rows matched the condition (this is not considered an error).

**Parameters**

- **ONLY**
  - If specified, delete rows from the named table only. When not specified, any tables inheriting from the named table are also processed.

- **`table`**
  - The name (optionally schema-qualified) of an existing table.

- **`alias`**
  - A substitute name for the target table. When an alias is provided, it completely hides the actual name of the table. For example, given `DELETE FROM foo AS f`, the remainder of the `DELETE` statement must refer to this table as `f` not `foo`.

- **`usinglist`**
  - A list of table expressions, allowing columns from other tables to appear in the `WHERE` condition. This is similar to the list of tables that can be specified in the `FROM` Clause of a `SELECT` statement; for example, an alias for the table name can be specified. Do not repeat the target table in the `usinglist`, unless you wish to set up a self-join.

- **`condition`**
  - An expression returning a value of type `boolean`, which determines the rows that are to be deleted.

- **`cursor_name`**
  - The name of the cursor to use in a `WHERE CURRENT OF` condition. The row to be deleted is the one most recently fetched from this cursor. The cursor must be a simple (non-join, non-aggregate) query on the `DELETE` target table.

  `WHERE CURRENT OF` cannot be specified together with a Boolean condition.

  The `DELETE...WHERE CURRENT OF` cursor statement can only be executed on the server, for example in an interactive `psql` session or a script. Language extensions such as PL/pgSQL do not have support for updatable cursors.

See `DECLARE` for more information about creating cursors.

**Notes**

Greenplum Database lets you reference columns of other tables in the `WHERE` condition by specifying the other tables in the `USING` clause. For example, to the name `Hannah` from the `rank` table, one might do:

```
DELETE FROM rank USING names WHERE names.id = rank.id AND name = 'Hannah';
```

What is essentially happening here is a join between `rank` and `names`, with all successfully joined rows being marked for deletion. This syntax is not standard. However, this join style is usually easier to write and faster to execute than a more standard sub-select style, such as:

```
DELETE FROM rank WHERE id IN (SELECT id FROM names WHERE name = 'Hannah');
```

When using `DELETE` to remove all the rows of a table (for example: `DELETE * FROM table;`), Greenplum Database adds an implicit `TRUNCATE` command (when user permissions allow). The added
**Examples**

Delete all films but musicals:

```sql
DELETE FROM films WHERE kind <> 'Musical';
```

Clear the table films:

```sql
DELETE FROM films;
```

Delete using a join:

```sql
DELETE FROM rank USING names WHERE names.id = rank.id AND name = 'Hannah';
```

**Compatibility**

This command conforms to the SQL standard, except that the `USING` clause is a Greenplum Database extension.

**See Also**

`DECLARE`, `TRUNCATE`

---

### DISCARD

Discards the session state.

**Synopsis**

```sql
DISCARD { ALL | PLANS | TEMPORARY | TEMP }
```

**Description**

`DISCARD` releases internal resources associated with a database session. These resources are normally released at the end of the session. `DISCARD TEMP` drops all temporary tables created in the current session. `DISCARD PLANS` releases all internally cached query plans. `DISCARD ALL` resets a session to its original state, discarding temporary resources and resetting session-local configuration changes.

**Parameters**

- `TEMPORARY, TEMP`

  Drops all temporary tables created in the current session.

- `PLANS`

  Releases all cached query plans.

- `ALL`
Releases all temporary resources associated with the current session and resets the session to its initial state. Currently, this has the same effect as executing the following sequence of statements:

```sql
SET SESSION AUTHORIZATION DEFAULT;
RESET ALL;
DEALLOCATE ALL;
CLOSE ALL;
UNLISTEN *;
SELECT pg_advisory_unlock_all();
DISCARD PLANS;
DISCARD TEMP;
```

**Notes**

`DISCARD ALL` cannot be executed inside a transaction block.

**Compatibility**

`DISCARD` is a Greenplum Database extension.

**DO**

Executes an anonymous code block as a transient anonymous function.

**Synopsis**

```sql
DO [ LANGUAGE lang_name ] code
```

**Description**

`DO` executes an anonymous code block, or in other words a transient anonymous function in a procedural language.

The code block is treated as though it were the body of a function with no parameters, returning void. It is parsed and executed a single time.

The optional `LANGUAGE` clause can appear either before or after the code block.

Anonymous blocks are procedural language structures that provide the capability to create and execute procedural code on the fly without persistently storing the code as database objects in the system catalogs. The concept of anonymous blocks is similar to UNIX shell scripts, which enable several manually entered commands to be grouped and executed as one step. As the name implies, anonymous blocks do not have a name, and for this reason they cannot be referenced from other objects. Although built dynamically, anonymous blocks can be easily stored as scripts in the operating system files for repetitive execution.

Anonymous blocks are standard procedural language blocks. They carry the syntax and obey the rules that apply to the procedural language, including declaration and scope of variables, execution, exception handling, and language usage.

The compilation and execution of anonymous blocks are combined in one step, while a user-defined function needs to be re-defined before use each time its definition changes.

**Parameters**

`code`

The procedural language code to be executed. This must be specified as a string literal, just as with the `CREATE FUNCTION` command. Use of a dollar-quoted literal is recommended. Optional keywords have no effect. These procedural languages are
supported: PL/pgSQL (plpgsql), PL/Python (plpythonu), and PL/Perl (plperl and plperlu).

**lang_name**
The name of the procedural language that the code is written in. The default is plpgsql. The language must be installed on the Greenplum Database system and registered in the database.

**Notes**
The PL/pgSQL language is installed on the Greenplum Database system and is registered in a user created database. The PL/Python language is installed by default, but not registered. Other languages are not installed or registered. The system catalog pg_language contains information about the registered languages in a database.

The user must have USAGE privilege for the procedural language, or must be a superuser if the language is untrusted. This is the same privilege requirement as for creating a function in the language.

**Examples**
This PL/pgSQL example grants all privileges on all views in schema public to role webuser:

```sql
DO $$
DECLARE r record;
BEGIN
  FOR r IN SELECT table_schema, table_name FROM information_schema.tables
    WHERE table_type = 'VIEW' AND table_schema = 'public'
  LOOP
    EXECUTE 'GRANT ALL ON ' || quote_ident(r.table_schema) || '.' || quote_ident(r.table_name) || ' TO webuser';
  END LOOP;
END$$;
```

This PL/pgSQL example determines if a Greenplum Database user is a superuser. In the example, the anonymous block retrieves the input value from a temporary table.

```sql
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:** The example PL/pgSQL uses SELECT with the INTO clause. It is different from the SQL command SELECT INTO.

**Compatibility**
There is no DO statement in the SQL standard.

**See Also**
CREATE LANGUAGE Greenplum PL/pgSQL Procedural Language
**DROP AGGREGATE**

Removes an aggregate function.

**Synopsis**

```
DROP AGGREGATE [IF EXISTS] name ( type [, ...] ) [CASCADE | RESTRICT]
```

**Description**

DROP AGGREGATE will delete an existing aggregate function. To execute this command the current user must be the owner of the aggregate function.

**Parameters**

**IF EXISTS**

Do not throw an error if the aggregate does not exist. A notice is issued in this case.

**name**

The name (optionally schema-qualified) of an existing aggregate function.

**type**

An input data type on which the aggregate function operates. To reference a zero-argument aggregate function, write `*` in place of the list of input data types.

**CASCADE**

Automatically drop objects that depend on the aggregate function.

**RESTRICT**

Refuse to drop the aggregate function if any objects depend on it. This is the default.

**Examples**

To remove the aggregate function `myavg` for type `integer`:

```
DROP AGGREGATE myavg(integer);
```

**Compatibility**

There is no DROP AGGREGATE statement in the SQL standard.

**See Also**

ALTER AGGREGATE, CREATE AGGREGATE

**DROP CAST**

Removes a cast.

**Synopsis**

```
DROP CAST [IF EXISTS] (sourcetype AS targettype) [CASCADE | RESTRICT]
```
Description

DROP CAST will delete a previously defined cast. To be able to drop a cast, you must own the source or the target data type. These are the same privileges that are required to create a cast.

Parameters

IF EXISTS

Do not throw an error if the cast does not exist. A notice is issued in this case.

sourcetype

The name of the source data type of the cast.

targettype

The name of the target data type of the cast.

CASCADE

RESTRICT

These keywords have no effect since there are no dependencies on casts.

Examples

To drop the cast from type text to type int:

```
DROP CAST (text AS int);
```

Compatibility

There DROP CAST command conforms to the SQL standard.

See Also

CREATE CAST

 DROP CONVERSION

Removes a conversion.

Synopsis

```
DROP CONVERSION [IF EXISTS] name [CASCADE | RESTRICT]
```

Description

DROP CONVERSION removes a previously defined conversion. To be able to drop a conversion, you must own the conversion.

Parameters

IF EXISTS

Do not throw an error if the conversion does not exist. A notice is issued in this case.

name

The name of the conversion. The conversion name may be schema-qualified.

CASCADE

RESTRICT

These keywords have no effect since there are no dependencies on conversions.
Examples
Drop the conversion named myname:

DROP CONVERSION myname;

Compatibility
There is no DROP CONVERSION statement in the SQL standard.

See Also
ALTER CONVERSION, CREATE CONVERSION

DROP DATABASE
Removes a database.

Synopsis

DROP DATABASE [IF EXISTS] name

Description
DROP DATABASE drops a database. It removes the catalog entries for the database and deletes the
directory containing the data. It can only be executed by the database owner. Also, it cannot be executed
while you or anyone else are connected to the target database. (Connect to postgres or any other
database to issue this command.)

Warning: DROP DATABASE cannot be undone. Use it with care!

Parameters

IF EXISTS
Do not throw an error if the database does not exist. A notice is issued in this case.

name
The name of the database to remove.

Notes

DROP DATABASE cannot be executed inside a transaction block.

This command cannot be executed while connected to the target database. Thus, it might be more
convenient to use the program dropdb instead, which is a wrapper around this command.

Examples
Drop the database named testdb:

DROP DATABASE testdb;

Compatibility
There is no DROP DATABASE statement in the SQL standard.

See Also
ALTER DATABASE, CREATE DATABASE
**DROP DOMAIN**

Removes a domain.

**Synopsis**

```
DROP DOMAIN [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

**Description**

`DROP DOMAIN` removes a previously defined domain. You must be the owner of a domain to drop it.

**Parameters**

- **IF EXISTS**
  Do not throw an error if the domain does not exist. A notice is issued in this case.
- **name**
  The name (optionally schema-qualified) of an existing domain.
- **CASCADE**
  Automatically drop objects that depend on the domain (such as table columns).
- **RESTRICT**
  Refuse to drop the domain if any objects depend on it. This is the default.

**Examples**

Drop the domain named `zipcode`:

```
DROP DOMAIN zipcode;
```

**Compatibility**

This command conforms to the SQL standard, except for the **IF EXISTS** option, which is a Greenplum Database extension.

**See Also**

`ALTER DOMAIN`, `CREATE DOMAIN`

---

**DROP EXTENSION**

Removes an extension from a Greenplum database.

**Synopsis**

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

**Description**

`DROP EXTENSION` removes extensions from the database. Dropping an extension causes its component objects to be dropped as well.

**Note:** The required supporting extension files what were installed to create the extension are not deleted. The files must be manually removed from the Greenplum Database hosts.
You must own the extension to use `DROP EXTENSION`.

This command fails if any of the extension objects are in use in the database. For example, if a table is defined with columns of the extension type. Add the `CASCADE` option to forcibly remove those dependent objects.

**Important:** Before issuing a `DROP EXTENSION` with the `CASCADE` keyword, you should be aware of all object that depend on the extension to avoid unintended consequences.

**Parameters**

**IF EXISTS**

Do not throw an error if the extension does not exist. A notice is issued.

**name**

The name of an installed extension.

**CASCADE**

Automatically drop objects that depend on the extension, and in turn all objects that depend on those objects. See the PostgreSQL information about *Dependency Tracking*.

**RESTRICT**

Refuse to drop an extension if any objects depend on it, other than the extension member objects. This is the default.

**Compatibility**

`DROP EXTENSION` is a Greenplum Database extension.

**See Also**

`CREATE EXTENSION`, `ALTER EXTENSION`

**DROP EXTERNAL TABLE**

Removes an external table definition.

**Synopsis**

```
DROP EXTERNAL [WEB] TABLE [IF EXISTS] name [CASCADE | RESTRICT]
```

**Description**

`DROP EXTERNAL TABLE` drops an existing external table definition from the database system. The external data sources or files are not deleted. To execute this command you must be the owner of the external table.

**Parameters**

**WEB**

Optional keyword for dropping external web tables.

**IF EXISTS**

Do not throw an error if the external table does not exist. A notice is issued in this case.

**name**

The name (optionally schema-qualified) of an existing external table.

**CASCADE**
Automatically drop objects that depend on the external table (such as views).

**RESTRICT**

Refuse to drop the external table if any objects depend on it. This is the default.

**Examples**

Remove the external table named `staging` if it exists:

```sql
DROP EXTERNAL TABLE IF EXISTS staging;
```

**Compatibility**

There is no `DROP EXTERNAL TABLE` statement in the SQL standard.

**See Also**

`CREATE EXTERNAL TABLE`  

**DROP FILESPACE**

Removes a filespace.

**Synopsis**

```sql
DROP FILESPACE [IF EXISTS] filespacename
```

**Description**

`DROP FILESPACE` removes a filespace definition and its system-generated data directories from the system.

A filespace can only be dropped by its owner or a superuser. The filespace must be empty of all tablespace objects before it can be dropped. It is possible that tablespaces in other databases may still be using a filespace even if no tablespaces in the current database are using the filespace.

**Parameters**

**IF EXISTS**

Do not throw an error if the filespace does not exist. A notice is issued in this case.

**tablespacename**

The name of the filespace to remove.

**Examples**

Remove the tablespace `myfs`:

```sql
DROP FILESPACE myfs;
```

**Compatibility**

There is no `DROP FILESPACE` statement in the SQL standard or in PostgreSQL.

**See Also**

`ALTER FILESPACE`, `DROP TABLESPACE`, `gpfilespace` in the *Greenplum Database Utility Guide*
DROP FUNCTION

Removes a function.

Synopsis

```sql
DROP FUNCTION [IF EXISTS] name ( [ [argmode] [argname] argtype [, ...] ] ) [CASCADE | RESTRICT]
```

Description

DROP FUNCTION removes the definition of an existing function. To execute this command, the user must be the owner of the function. The argument types to the function must be specified, since several different functions may exist with the same name and different argument lists.

Parameters

IF EXISTS

Do not throw an error if the function does not exist. A notice is issued in this case.

name

The name (optionally schema-qualified) of an existing function.

argmode

The mode of an argument: either IN, OUT, INOUT, or VARIADIC. If omitted, the default is IN. Note that DROP FUNCTION does not actually pay any attention to OUT arguments, since only the input arguments are needed to determine the function's identity. So it is sufficient to list the IN, INOUT, and VARIADIC arguments.

argname

The name of an argument. Note that DROP FUNCTION does not actually pay any attention to argument names, since only the argument data types are needed to determine the function's identity.

argtype

The data type(s) of the function's arguments (optionally schema-qualified), if any.

CASCADE

Automatically drop objects that depend on the function such as operators.

RESTRICT

Refuse to drop the function if any objects depend on it. This is the default.

Examples

Drop the square root function:

```sql
DROP FUNCTION sqrt(integer);
```

Compatibility

A DROP FUNCTION statement is defined in the SQL standard, but it is not compatible with this command.

See Also

CREATE FUNCTION, ALTER FUNCTION
**DROP GROUP**

Removes a database role.

**Synopsis**

```
DROP GROUP [IF EXISTS] name [, ...]
```

**Description**

`DROP GROUP` is an alias for `DROP ROLE`. See `DROP ROLE` for more information.

**Compatibility**

There is no `DROP GROUP` statement in the SQL standard.

**See Also**

`DROP ROLE`

**DROP INDEX**

Removes an index.

**Synopsis**

```
DROP INDEX [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

**Description**

`DROP INDEX` drops an existing index from the database system. To execute this command you must be the owner of the index.

**Parameters**

**IF EXISTS**

Do not throw an error if the index does not exist. A notice is issued in this case.

**name**

The name (optionally schema-qualified) of an existing index.

**CASCADE**

Automatically drop objects that depend on the index.

**RESTRICT**

Refuse to drop the index if any objects depend on it. This is the default.

**Examples**

Remove the index `title_idx`:

```
DROP INDEX title_idx;
```
**Compatibility**

DROP INDEX is a Greenplum Database language extension. There are no provisions for indexes in the SQL standard.

**See Also**

ALTER INDEX, CREATE INDEX, REINDEX

**DROP LANGUAGE**

Removes a procedural language.

**Synopsis**

DROP [PROCEDURAL] LANGUAGE [IF EXISTS] name [CASCADE | RESTRICT]

**Description**

DROP LANGUAGE will remove the definition of the previously registered procedural language. You must be a superuser or owner of the language to drop a language.

**Parameters**

PROCEDURAL

Optional keyword - has no effect.

IF EXISTS

Do not throw an error if the language does not exist. A notice is issued in this case.

name

The name of an existing procedural language. For backward compatibility, the name may be enclosed by single quotes.

CASCADE

Automatically drop objects that depend on the language (such as functions written in that language).

RESTRICT

Refuse to drop the language if any objects depend on it. This is the default.

**Examples**

Remove the procedural language plsample:

```
DROP LANGUAGE plsample;
```

**Compatibility**

There is no DROP LANGUAGE statement in the SQL standard.

**See Also**

ALTER LANGUAGE, CREATE LANGUAGE

**DROP OPERATOR**

Removes an operator.
Synopsis

```
DROP OPERATOR [IF EXISTS] name ( {lefttype | NONE} ,
   {righttype | NONE} ) [CASCADE | RESTRICT]
```

Description

DROP OPERATOR drops an existing operator from the database system. To execute this command you must be the owner of the operator.

Parameters

- **IF EXISTS**
  Do not throw an error if the operator does not exist. A notice is issued in this case.
- **name**
  The name (optionally schema-qualified) of an existing operator.
- **lefttype**
  The data type of the operator’s left operand; write NONE if the operator has no left operand.
- **righttype**
  The data type of the operator’s right operand; write NONE if the operator has no right operand.
- **CASCADE**
  Automatically drop objects that depend on the operator.
- **RESTRICT**
  Refuse to drop the operator if any objects depend on it. This is the default.

Examples

Remove the power operator \( a^b \) for type integer:

```
DROP OPERATOR ^ (integer, integer);
```

Remove the left unary bitwise complement operator \( \sim b \) for type bit:

```
DROP OPERATOR ~ (none, bit);
```

Remove the right unary factorial operator \( x! \) for type bigint:

```
DROP OPERATOR ! (bigint, none);
```

Compatibility

There is no DROP OPERATOR statement in the SQL standard.

See Also

ALTER OPERATOR, CREATE OPERATOR

**DROP OPERATOR CLASS**

Removes an operator class.
Synopsis

DROP OPERATOR CLASS [IF EXISTS] name USING index_method [CASCADE | RESTRICT]

Description

DROP OPERATOR CLASS drops an existing operator class. To execute this command you must be the owner of the operator class.

Parameters

IF EXISTS
Do not throw an error if the operator class does not exist. A notice is issued in this case.

name
The name (optionally schema-qualified) of an existing operator class.

index_method
The name of the index access method the operator class is for.

CASCADE
Automatically drop objects that depend on the operator class.

RESTRICT
Refuse to drop the operator class if any objects depend on it. This is the default.

Examples

Remove the B-tree operator class widget_ops:

DROP OPERATOR CLASS widget_ops USING btree;

This command will not succeed if there are any existing indexes that use the operator class. Add CASCADE to drop such indexes along with the operator class.

Compatibility

There is no DROP OPERATOR CLASS statement in the SQL standard.

See Also

ALTER OPERATOR CLASS, CREATE OPERATOR CLASS

DROP OPERATOR FAMILY

Removes an operator family.

Synopsis

DROP OPERATOR FAMILY [IF EXISTS] name USING index_method [CASCADE | RESTRICT]

Description

DROP OPERATOR FAMILY drops an existing operator family. To execute this command you must be the owner of the operator family.
DROP OPERATOR FAMILY includes dropping any operator classes contained in the family, but it does not drop any of the operators or functions referenced by the family. If there are any indexes depending on operator classes within the family, you will need to specify CASCADE for the drop to complete.

**Parameters**

**IF EXISTS**
Do not throw an error if the operator family does not exist. A notice is issued in this case.

**name**
The name (optionally schema-qualified) of an existing operator family.

**index_method**
The name of the index access method the operator family is for.

**CASCADE**
Automatically drop objects that depend on the operator family.

**RESTRICT**
Refuse to drop the operator family if any objects depend on it. This is the default.

**Examples**
Remove the B-tree operator family float_ops:

```
DROP OPERATOR FAMILY float_ops USING btree;
```

This command will not succeed if there are any existing indexes that use the operator family. Add CASCADE to drop such indexes along with the operator family.

**Compatibility**
There is no DROP OPERATOR FAMILY statement in the SQL standard.

**See Also**
ALTER OPERATOR FAMILY, CREATE OPERATOR FAMILY, ALTER OPERATOR CLASS, CREATE OPERATOR CLASS, DROP OPERATOR CLASS

---

**DROP OWNED**
Removes database objects owned by a database role.

**Synopsis**

```
DROP OWNED BY name [, ...] [CASCADE | RESTRICT]
```

**Description**

DROP OWNED drops all the objects in the current database that are owned by one of the specified roles. Any privileges granted to the given roles on objects in the current database will also be revoked.

**Parameters**

**name**
The name of a role whose objects will be dropped, and whose privileges will be revoked.

**CASCADE**
Automatically drop objects that depend on the affected objects.
RESTRICT

Refuse to drop the objects owned by a role if any other database objects depend on one of
the affected objects. This is the default.

Notes

DROP OWNED is often used to prepare for the removal of one or more roles. Because DROP OWNED only
affects the objects in the current database, it is usually necessary to execute this command in each
database that contains objects owned by a role that is to be removed.

Using the CASCADE option may make the command recurse to objects owned by other users.

The REASSIGN OWNED command is an alternative that reassigns the ownership of all the database objects
owned by one or more roles.

Examples

Remove any database objects owned by the role named sally:

DROP OWNED BY sally;

Compatibility

The DROP OWNED statement is a Greenplum Database extension.

See Also

REASSIGN OWNED, DROP ROLE

DROP PROTOCOL

Removes a external table data access protocol from a database.

Synopsis

DROP PROTOCOL [IF EXISTS] name

Description

DROP PROTOCOL removes the specified protocol from a database. A protocol name can be specified in the
CREATE EXTERNAL TABLE command to read data from or write data to an external data source.

Warning: If you drop a data access protocol, external tables that have been defined with the
protocol will no longer be able to access the external data source.

Parameters

IF EXISTS

Do not throw an error if the protocol does not exist. A notice is issued in this case.

name

The name of an existing data access protocol.

Notes

If you drop a data access protocol, the call handlers that defined in the database that are associated with
the protocol are not dropped. You must drop the functions manually.
Shared libraries that were used by the protocol should also be removed from the Greenplum Database hosts.

**Compatibility**

DROP PROTOCOL is a Greenplum Database extension.

**See Also**

CREATE EXTERNAL TABLE, CREATE PROTOCOL

### DROP RESOURCE GROUP

**Warning:** Resource groups are an experimental feature and are not intended for use in a production environment. Experimental features are subject to change without notice in future releases.

Removes a resource group.

**Synopsis**

```
DROP RESOURCE GROUP group_name
```

**Description**

This command removes a resource group from Greenplum Database. Only a superuser can drop a resource group.

To drop a resource group, the group cannot be assigned to any roles, nor can it have any statements pending or running in the group.

You cannot drop the pre-defined `admin_group` and `default_group` resource groups.

**Parameters**

`group_name`

The name of the resource group to remove.

**Notes**

You cannot submit a DROP RESOURCE GROUP command in an explicit transaction or sub-transaction.

Use `ALTER ROLE` to remove the resource group assigned to a specific user/role.

Perform the following query to view all of the currently active queries for all resource groups:

```
SELECT usename, current_query, waiting, procpid, rsgid, rsgname, rsgqueueduration
FROM pg_stat_activity;
```

To view the resource group assignments, perform the following query on the `pg_roles` and `pg_resgroup` system catalog tables:

```
SELECT rolname, rsgname
FROM pg_roles, pg_resgroup
WHERE pg_roles.rolresgroup=pg_resgroup.oid;
```
Examples

Remove the resource group assigned to a role. This operation then assigns the default resource group `default_group` to the role:

```sql
ALTER ROLE bob RESOURCE GROUP NONE;
```

Remove the resource group named `adhoc`:

```sql
DROP RESOURCE GROUP adhoc;
```

Compatibility

The `DROP RESOURCE GROUP` statement is a Greenplum Database extension.

See Also

`ALTER RESOURCE GROUP`, `CREATE RESOURCE GROUP`, `ALTER ROLE`

---

**DROP RESOURCE QUEUE**

Removes a resource queue.

**Synopsis**

```
DROP RESOURCE QUEUE queue_name
```

**Description**

This command removes a resource queue from Greenplum Database. To drop a resource queue, the queue cannot have any roles assigned to it, nor can it have any statements waiting in the queue. Only a superuser can drop a resource queue.

**Parameters**

`queue_name`

The name of a resource queue to remove.

**Notes**

Use `ALTER ROLE` to remove a user from a resource queue.

To see all the currently active queries for all resource queues, perform the following query of the `pg_locks` table joined with the `pg_roles` and `pg_resqueue` tables:

```sql
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;
```

To see the roles assigned to a resource queue, perform the following query of the `pg_roles` and `pg_resqueue` system catalog tables:

```sql
SELECT rolname, rsqname FROM pg_roles, pg_resqueue WHERE
pg_roles.rolresqueue=pg_resqueue.oid;
```
Examples
Remove a role from a resource queue (and move the role to the default resource queue, pg_default):

```
ALTER ROLE bob RESOURCE QUEUE NONE;
```

Remove the resource queue named adhoc:

```
DROP RESOURCE QUEUE adhoc;
```

Compatibility
The DROP RESOURCE QUEUE statement is a Greenplum Database extension.

See Also
ALTER RESOURCE QUEUE, CREATE RESOURCE QUEUE, ALTER ROLE

**DROP ROLE**
Removes a database role.

Synopsis

```
DROP ROLE [IF EXISTS] name [, ...]
```

Description

DROP ROLE removes the specified role(s). To drop a superuser role, you must be a superuser yourself. To drop non-superuser roles, you must have CREATEROLE privilege.

A role cannot be removed if it is still referenced in any database; an error will be raised if so. Before dropping the role, you must drop all the objects it owns (or reassign their ownership) and revoke any privileges the role has been granted. The REASSIGN OWNED and DROP OWNED commands can be useful for this purpose.

However, it is not necessary to remove role memberships involving the role; DROP ROLE automatically revokes any memberships of the target role in other roles, and of other roles in the target role. The other roles are not dropped nor otherwise affected.

Parameters

**IF EXISTS**
Do not throw an error if the role does not exist. A notice is issued in this case.

**name**
The name of the role to remove.

Examples
Remove the roles named sally and bob:

```
DROP ROLE sally, bob;
```

Compatibility
The SQL standard defines DROP ROLE, but it allows only one role to be dropped at a time, and it specifies different privilege requirements than Greenplum Database uses.
**DROP RULE**

Removes a rewrite rule.

**Synopsis**

```
DROP RULE [IF EXISTS] name ON relation [CASCADE | RESTRICT]
```

**Description**

DROP RULE drops a rewrite rule from a table or view.

**Parameters**

- **IF EXISTS**
  - Do not throw an error if the rule does not exist. A notice is issued in this case.

- **name**
  - The name of the rule to remove.

- **relation**
  - The name (optionally schema-qualified) of the table or view that the rule applies to.

- **CASCADE**
  - Automatically drop objects that depend on the rule.

- **RESTRICT**
  - Refuse to drop the rule if any objects depend on it. This is the default.

**Examples**

Remove the rewrite rule `sales_2006` on the table `sales`:

```
DROP RULE sales_2006 ON sales;
```

**Compatibility**

There is no DROP RULE statement in the SQL standard.

**See Also**

CREATE RULE

---

**DROP SCHEMA**

Removes a schema.

**Synopsis**

```
DROP SCHEMA [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```
**Description**

DROP SCHEMA removes schemas from the database. A schema can only be dropped by its owner or a superuser. Note that the owner can drop the schema (and thereby all contained objects) even if he does not own some of the objects within the schema.

**Parameters**

**IF EXISTS**

Do not throw an error if the schema does not exist. A notice is issued in this case.

**name**

The name of the schema to remove.

**CASCADE**

Automatically drops any objects contained in the schema (tables, functions, etc.).

**RESTRICT**

Refuse to drop the schema if it contains any objects. This is the default.

**Examples**

Remove the schema mystuff from the database, along with everything it contains:

```
DROP SCHEMA mystuff CASCADE;
```

**Compatibility**

DROP SCHEMA is fully conforming with the SQL standard, except that the standard only allows one schema to be dropped per command. Also, the IF EXISTS option is a Greenplum Database extension.

**See Also**

CREATE SCHEMA, ALTER SCHEMA

---

**DROP SEQUENCE**

Removes a sequence.

**Synopsis**

```
DROP SEQUENCE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

**Description**

DROP SEQUENCE removes a sequence generator table. You must own the sequence to drop it (or be a superuser).

**Parameters**

**IF EXISTS**

Do not throw an error if the sequence does not exist. A notice is issued in this case.

**name**

The name (optionally schema-qualified) of the sequence to remove.

**CASCADE**

Automatically drop objects that depend on the sequence.

**RESTRICT**
Refuse to drop the sequence if any objects depend on it. This is the default.

**Examples**

Remove the sequence `myserial`:

```
DROP SEQUENCE myserial;
```

**Compatibility**

`DROP SEQUENCE` is fully conforming with the SQL standard, except that the standard only allows one sequence to be dropped per command. Also, the `IF EXISTS` option is a Greenplum Database extension.

**See Also**

`ALTER SEQUENCE`, `CREATE SEQUENCE`

---

**DROP TABLE**

Removes a table.

**Synopsis**

```
DROP TABLE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

**Description**

`DROP TABLE` removes tables from the database. Only its owner may drop a table. To empty a table of rows without removing the table definition, use `DELETE` or `TRUNCATE`.

`DROP TABLE` always removes any indexes, rules, triggers, and constraints that exist for the target table. However, to drop a table that is referenced by a view, `CASCADE` must be specified. `CASCADE` will remove a dependent view entirely.

**Parameters**

- **IF EXISTS**
  
  Do not throw an error if the table does not exist. A notice is issued in this case.

- **name**
  
  The name (optionally schema-qualified) of the table to remove.

- **CASCADE**
  
  Automatically drop objects that depend on the table (such as views).

- **RESTRICT**
  
  Refuse to drop the table if any objects depend on it. This is the default.

**Examples**

Remove the table `mytable`:

```
DROP TABLE mytable;
```

**Compatibility**

`DROP TABLE` is fully conforming with the SQL standard, except that the standard only allows one table to be dropped per command. Also, the `IF EXISTS` option is a Greenplum Database extension.
DROP TABLESPACE

Removes a tablespace.

Synopsis

```
DROP TABLESPACE [IF EXISTS] tablespacename
```

Description

DROP TABLESPACE removes a tablespace from the system.

A tablespace can only be dropped by its owner or a superuser. The tablespace must be empty of all database objects before it can be dropped. It is possible that objects in other databases may still reside in the tablespace even if no objects in the current database are using the tablespace.

Parameters

**IF EXISTS**

Do not throw an error if the tablespace does not exist. A notice is issued in this case.

**tablespacename**

The name of the tablespace to remove.

Examples

Remove the tablespace mystuff:

```
DROP TABLESPACE mystuff;
```

Compatibility

DROP TABLESPACE is a Greenplum Database extension.

See Also

CREATE TABLESPACE, ALTER TABLESPACE

DROP TYPE

Removes a data type.

Synopsis

```
DROP TYPE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

Description

DROP TYPE will remove a user-defined data type. Only the owner of a type can remove it.

Parameters

**IF EXISTS**

Do not throw an error if the type does not exist. A notice is issued in this case.
**DROP TYPE**

Removes a data type.

**Synopsis**

```
DROP TYPE [IF EXISTS] name [, ...]
```

**Description**

DROP TYPE is a synonym for DROP TYPE. See DROP TYPE for more information.

**Examples**

Remove the data type `box`;

```
DROP TYPE box;
```

**Compatibility**

This command is similar to the corresponding command in the SQL standard, apart from the IF EXISTS option, which is a Greenplum Database extension. But note that the CREATE TYPE command and the data type extension mechanisms in Greenplum Database differ from the SQL standard.

**See Also**

ALTER TYPE, CREATE TYPE

**DROP USER**

Removes a database role.

**Synopsis**

```
DROP USER [IF EXISTS] name [, ...]
```

**Description**

DROP USER is an alias for DROP ROLE. See DROP ROLE for more information.

**Compatibility**

There is no DROP USER statement in the SQL standard. The SQL standard leaves the definition of users to the implementation.

**See Also**

DROP ROLE

**DROP VIEW**

Removes a view.

**Synopsis**

```
DROP VIEW [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

**Examples**

Remove the view `box`;

```
DROP VIEW box;
```
**Description**

DROP VIEW will remove an existing view. Only the owner of a view can remove it.

**Parameters**

**IF EXISTS**

Do not throw an error if the view does not exist. A notice is issued in this case.

**name**

The name (optionally schema-qualified) of the view to remove.

**CASCADE**

Automatically drop objects that depend on the view (such as other views).

**RESTRICT**

Refuse to drop the view if any objects depend on it. This is the default.

**Examples**

Remove the view topten:

```sql
DROP VIEW topten;
```

**Compatibility**

DROP VIEW is fully conforming with the SQL standard, except that the standard only allows one view to be dropped per command. Also, the IF EXISTS option is a Greenplum Database extension.

**See Also**

CREATE VIEW

**END**

Commits the current transaction.

**Synopsis**

```sql
END [WORK | TRANSACTION]
```

**Description**

END commits the current transaction. All changes made by the transaction become visible to others and are guaranteed to be durable if a crash occurs. This command is a Greenplum Database extension that is equivalent to COMMIT.

**Parameters**

**WORK**

**TRANSACTION**

Optional keywords. They have no effect.

**Examples**

Commit the current transaction:

```sql
END;
```
**EXECUTE**

Executes a prepared SQL statement.

**Synopsis**

```
EXECUTE name [ (parameter [, ...] ) ]
```

**Description**

`EXECUTE` is used to execute a previously prepared statement. Since prepared statements only exist for the duration of a session, the prepared statement must have been created by a `PREPARE` statement executed earlier in the current session.

If the `PREPARE` statement that created the statement specified some parameters, a compatible set of parameters must be passed to the `EXECUTE` statement, or else an error is raised. Note that (unlike functions) prepared statements are not overloaded based on the type or number of their parameters; the name of a prepared statement must be unique within a database session.

For more information on the creation and usage of prepared statements, see `PREPARE`.

**Parameters**

- `name`
  The name of the prepared statement to execute.

- `parameter`
  The actual value of a parameter to the prepared statement. This must be an expression yielding a value that is compatible with the data type of this parameter, as was determined when the prepared statement was created.

**Examples**

Create a prepared statement for an `INSERT` statement, and then execute it:

```
PREPARE fooplan (int, text, bool, numeric) AS INSERT INTO foo VALUES($1, $2, $3, $4);
EXECUTE fooplan(1, 'Hunter Valley', 't', 200.00);
```

**Compatibility**

The SQL standard includes an `EXECUTE` statement, but it is only for use in embedded SQL. This version of the `EXECUTE` statement also uses a somewhat different syntax.

**See Also**

`DEALLOCATE`, `PREPARE`
**EXPLAIN**

Shows the query plan of a statement.

**Synopsis**

```plaintext
EXPLAIN [ANALYZE] [VERBOSE] statement
```

**Description**

`EXPLAIN` displays the query plan that the Greenplum planner generates for the supplied statement. Query plans are a tree plan of nodes. Each node in the plan represents a single operation, such as table scan, join, aggregation or a sort.

Plans should be read from the bottom up as each node feeds rows into the node directly above it. The bottom nodes of a plan are usually table scan operations (sequential, index or bitmap index scans). If the query requires joins, aggregations, or sorts (or other operations on the raw rows) then there will be additional nodes above the scan nodes to perform these operations. The topmost plan nodes are usually the Greenplum Database motion nodes (redistribute, explicit redistribute, broadcast, or gather motions). These are the operations responsible for moving rows between the segment instances during query processing.

The output of `EXPLAIN` has one line for each node in the plan tree, showing the basic node type plus the following cost estimates that the planner made for the execution of that plan node:

- **cost** — measured in units of disk page fetches; that is, 1.0 equals one sequential disk page read. The first estimate is the start-up cost (cost of getting to the first row) and the second is the total cost (cost of getting all rows). Note that the total cost assumes that all rows will be retrieved, which may not always be the case (if using `LIMIT` for example).
- **rows** — the total number of rows output by this plan node. This is usually less than the actual number of rows processed or scanned by the plan node, reflecting the estimated selectivity of any `WHERE` clause conditions. Ideally the top-level nodes estimate will approximate the number of rows actually returned, updated, or deleted by the query.
- **width** — total bytes of all the rows output by this plan node.

It is important to note that the cost of an upper-level node includes the cost of all its child nodes. The topmost node of the plan has the estimated total execution cost for the plan. This is this number that the planner seeks to minimize. It is also important to realize that the cost only reflects things that the query optimizer cares about. In particular, the cost does not consider the time spent transmitting result rows to the client.

`EXPLAIN ANALYZE` causes the statement to be actually executed, not only planned. The `EXPLAIN ANALYZE` plan shows the actual results along with the planner's estimates. This is useful for seeing whether the planner's estimates are close to reality. In addition to the information shown in the `EXPLAIN` plan, `EXPLAIN ANALYZE` will show the following additional information:

- The total elapsed time (in milliseconds) that it took to run the query.
- The number of `workers` (segments) involved in a plan node operation. Only segments that return rows are counted.
- The maximum number of rows returned by the segment that produced the most rows for an operation. If multiple segments produce an equal number of rows, the one with the longest `time to end` is the one chosen.
- The segment id number of the segment that produced the most rows for an operation.
• For relevant operations, the `work_mem` used by the operation. If `work_mem` was not sufficient to perform the operation in memory, the plan will show how much data was spilled to disk and how many passes over the data were required for the lowest performing segment. For example:

```
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
```

• The time (in milliseconds) it took to retrieve the first row from the segment that produced the most rows, and the total time taken to retrieve all rows from that segment. The `<time> to first row` may be omitted if it is the same as the `<time> to end`.

**Important:** Keep in mind that the statement is actually executed when `EXPLAIN ANALYZE` is used. Although `EXPLAIN ANALYZE` will discard any output that a `SELECT` would return, other side effects of the statement will happen as usual. If you wish to use `EXPLAIN ANALYZE` on a DML statement without letting the command affect your data, use this approach:

```
BEGIN;
EXPLAIN ANALYZE ...;
ROLLBACK;
```

**Parameters**

`name`

The name of the prepared statement to execute.

`parameter`

The actual value of a parameter to the prepared statement. This must be an expression yielding a value that is compatible with the data type of this parameter, as was determined when the prepared statement was created.

**Notes**

In order to allow the query optimizer to make reasonably informed decisions when optimizing queries, the `ANALYZE` statement should be run to record statistics about the distribution of data within the table. If you have not done this (or if the statistical distribution of the data in the table has changed significantly since the last time `ANALYZE` was run), the estimated costs are unlikely to conform to the real properties of the query, and consequently an inferior query plan may be chosen.

An SQL statement that is run during the execution of an `EXPLAIN ANALYZE` command is excluded from Greenplum Database resource queues.

For more information about query profiling, see "Query Profiling" in the Greenplum Database Administrator Guide. For more information about resource queues, see "Resource Management with Resource Queues" in the Greenplum Database Administrator Guide.

**Examples**

To illustrate how to read an `EXPLAIN` query plan, consider the following example for a very simple query:

```
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
```
If we read the plan from the bottom up, the query optimizer starts by doing a sequential scan of the names table. Notice that the WHERE clause is being applied as a filter condition. This means that the scan operation checks the condition for each row it scans, and outputs only the ones that pass the condition.

The results of the scan operation are passed up to a gather motion operation. In Greenplum Database, a gather motion is when segments send rows up to the master. In this case we have 2 segment instances sending to 1 master instance (2:1). This operation is working on slice1 of the parallel query execution plan. In Greenplum Database a query plan is divided into slices so that portions of the query plan can be worked on in parallel by the segments.

The estimated startup cost for this plan is 00.00 (no cost) and a total cost of 20.88 disk page fetches. The planner is estimating that this query will return one row.

**Compatibility**

There is no EXPLAIN statement defined in the SQL standard.

**See Also**

ANALYZE

**FETCH**

Retrieves rows from a query using a cursor.

**Synopsis**

```plaintext
FETCH [ forward_direction { FROM | IN } ] cursorname
```

where `forward_direction` can be empty or one of:

- NEXT
- FIRST
- LAST
- ABSOLUTE count
- RELATIVE count
- count
- ALL
- FORWARD
- FORWARD count
- FORWARD ALL

**Description**

`FETCH` retrieves rows using a previously-created cursor.

A cursor has an associated position, which is used by `FETCH`. The cursor position can be before the first row of the query result, on any particular row of the result, or after the last row of the result. When created, a cursor is positioned before the first row. After fetching some rows, the cursor is positioned on the row most recently retrieved. If `FETCH` runs off the end of the available rows then the cursor is left positioned after the last row. `FETCH ALL` will always leave the cursor positioned after the last row.

The forms `NEXT, FIRST, LAST, ABSOLUTE, RELATIVE` fetch a single row after moving the cursor appropriately. If there is no such row, an empty result is returned, and the cursor is left positioned before the first row or after the last row as appropriate.

The forms using `FORWARD` retrieve the indicated number of rows moving in the forward direction, leaving the cursor positioned on the last-returned row (or after all rows, if the count exceeds the number of rows available). Note that it is not possible to move a cursor position backwards in Greenplum Database, since scrollable cursors are not supported. You can only move a cursor forward in position using `FETCH`. 
RELATIVE 0 and FORWARD 0 request fetching the current row without moving the cursor, that is, re-fetching the most recently fetched row. This will succeed unless the cursor is positioned before the first row or after the last row, in which case no row is returned.

**Outputs**

On successful completion, a FETCH command returns a command tag of the form

```
FETCH count
```

The count is the number of rows fetched (possibly zero). Note that in `psql`, the command tag will not actually be displayed, since `psql` displays the fetched rows instead.

**Parameters**

`forward_direction`

Defines the fetch direction and number of rows to fetch. Only forward fetches are allowed in Greenplum Database. It can be one of the following:

**NEXT**

Fetch the next row. This is the default if direction is omitted.

**FIRST**

Fetch the first row of the query (same as `ABSOLUTE 1`). Only allowed if it is the first FETCH operation using this cursor.

**LAST**

Fetch the last row of the query (same as `ABSOLUTE -1`).

**ABSOLUTE count**

Fetch the specified row of the query. Position after last row if count is out of range. Only allowed if the row specified by `count` moves the cursor position forward.

**RELATIVE count**

Fetch the specified row of the query `count` rows ahead of the current cursor position. RELATIVE 0 re-fetches the current row, if any. Only allowed if `count` moves the cursor position forward.

**count**

Fetch the next `count` number of rows (same as FORWARD `count`).

**ALL**

Fetch all remaining rows (same as FORWARD ALL).

**FORWARD**

Fetch the next row (same as NEXT).

**FORWARD count**

Fetch the next `count` number of rows. FORWARD 0 re-fetches the current row.

**FORWARD ALL**

Fetch all remaining rows.

**cursorname**

The name of an open cursor.

**Notes**

Greenplum Database does not support scrollable cursors, so you can only use FETCH to move the cursor position forward.
ABSOLUTE fetches are not any faster than navigating to the desired row with a relative move: the underlying implementation must traverse all the intermediate rows anyway.

DECLARE is used to define a cursor. Use MOVE to change cursor position without retrieving data.

**Examples**

-- Start the transaction:

```
BEGIN;
```

-- Set up a cursor:

```
DECLARE mycursor CURSOR FOR SELECT * FROM films;
```

-- Fetch the first 5 rows in the cursor mycursor:

```
FETCH FORWARD 5 FROM mycursor;
```

```
<table>
<thead>
<tr>
<th>code</th>
<th>title</th>
<th>did</th>
<th>date_prod</th>
<th>kind</th>
<th>len</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL101</td>
<td>The Third Man</td>
<td>101</td>
<td>1949-12-23</td>
<td>Drama</td>
<td>01:44</td>
</tr>
<tr>
<td>BL102</td>
<td>The African Queen</td>
<td>101</td>
<td>1951-08-11</td>
<td>Romantic</td>
<td>01:43</td>
</tr>
<tr>
<td>JL201</td>
<td>Une Femme est une Femme</td>
<td>102</td>
<td>1961-03-12</td>
<td>Romantic</td>
<td>01:25</td>
</tr>
<tr>
<td>P_301</td>
<td>Vertigo</td>
<td>103</td>
<td>1958-11-14</td>
<td>Action</td>
<td>02:08</td>
</tr>
<tr>
<td>P_302</td>
<td>Becket</td>
<td>103</td>
<td>1964-02-03</td>
<td>Drama</td>
<td>02:28</td>
</tr>
</tbody>
</table>
```

-- Close the cursor and end the transaction:

```
CLOSE mycursor;
COMMIT;
```

Change the kind column of the table films in the row at the c_films cursor’s current position:

```
UPDATE films SET kind = 'Dramatic' WHERE CURRENT OF c_films;
```

**Compatibility**

SQL standard allows cursors only in embedded SQL and in modules. Greenplum Database permits cursors to be used interactively.

The variant of FETCH described here returns the data as if it were a SELECT result rather than placing it in host variables. Other than this point, FETCH is fully upward-compatible with the SQL standard.

The FETCH forms involving FORWARD, as well as the forms FETCH count and FETCHALL, in which FORWARD is implicit, are Greenplum Database extensions. BACKWARD is not supported.

The SQL standard allows only FROM preceding the cursor name; the option to use IN is an extension.

**See Also**

DECLARE, CLOSE, MOVE

**GRANT**

Defines access privileges.

**Synopsis**

```
GRANT { {SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRIGGER | TRUNCATE } [,....] | ALL [PRIVILEGES] }
```
ON [TABLE] tablename [, ...]
TO { rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT { {USAGE | SELECT | UPDATE} [, ...] ALL [PRIVILEGES] }
ON SEQUENCE sequencename [, ...]
TO { rolename | PUBLIC } [, ...] [WITH GRANT OPTION]

GRANT { {CREATE | CONNECT | TEMPORARY | TEMP} [, ...] ALL [PRIVILEGES] }
ON DATABASE dbname [, ...]
TO { rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT { EXECUTE | ALL [PRIVILEGES] }
ON FUNCTION funcname ( [ [argmode] [argname] argtype [, ...] ] ) [, ...]
TO { rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT { USAGE | ALL [PRIVILEGES] }
ON LANGUAGE langname [, ...]
TO { rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT { {CREATE | USAGE} [, ...] | ALL [PRIVILEGES] }
ON SCHEMA schemaname [, ...]
TO { rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT { CREATE | ALL [PRIVILEGES] }
ON TABLESPACE tablespacename [, ...]
TO { rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT parent_role [, ...]
TO member_role [, ...] [WITH ADMIN OPTION]

GRANT { SELECT | INSERT | ALL [PRIVILEGES] }
ON PROTOCOL protocolname
TO username

Description

The GRANT command has two basic variants: one that grants privileges on a database object (table, view, sequence, database, function, procedural language, schema, or tablespace), and one that grants membership in a role.

GRANT on Database Objects

This variant of the GRANT command gives specific privileges on a database object to one or more roles. These privileges are added to those already granted, if any.

The key word PUBLIC indicates that the privileges are to be granted to all roles, including those that may be created later. PUBLIC may be thought of as an implicitly defined group-level role that always includes all roles. Any particular role will have the sum of privileges granted directly to it, privileges granted to any role it is presently a member of, and privileges granted to PUBLIC.

If WITH GRANT OPTION is specified, the recipient of the privilege may in turn grant it to others. Without a grant option, the recipient cannot do that. Grant options cannot be granted to PUBLIC.

There is no need to grant privileges to the owner of an object (usually the role that created it), as the owner has all privileges by default. The right to drop an object, or to alter its definition in any way is not described by a grantable privilege; it is inherent in the owner, and cannot be granted or revoked. The owner implicitly has all grant options for the object, too.

Depending on the type of object, the initial default privileges may include granting some privileges to PUBLIC. The default is no public access for tables, schemas, and tablespaces; CONNECT privilege and
TEMP table creation privilege for databases; EXECUTE privilege for functions; and USAGE privilege for languages. The object owner may of course revoke these privileges.

**GRANT on Roles**

This variant of the GRANT command grants membership in a role to one or more other roles. Membership in a role is significant because it conveys the privileges granted to a role to each of its members.

If WITH ADMIN OPTION is specified, the member may in turn grant membership in the role to others, and revoke membership in the role as well. Database superusers can grant or revoke membership in any role to anyone. Roles having CREATEROLE privilege can grant or revoke membership in any role that is not a superuser.

Unlike the case with privileges, membership in a role cannot be granted to PUBLIC.

**GRANT on Protocols**

After creating a custom protocol, specify CREATE TRUSTED PROTOCOL to be allowed to allowing any user besides the owner to access it. If the protocol is not trusted, you cannot give any other user permission to use it to read or write data. After a TRUSTED protocol is created, you can specify which other users can access it with the GRANT command.

- To allow a user to create a readable external table with a trusted protocol
  
  ```sql
  GRANT SELECT ON PROTOCOL protocolname TO username
  ```

- To allow a user to create a writable external table with a trusted protocol
  
  ```sql
  GRANT INSERT ON PROTOCOL protocolname TO username
  ```

- To allow a user to create both readable and writable external table with a trusted protocol
  
  ```sql
  GRANT ALL ON PROTOCOL protocolname TO username
  ```

**Parameters**

**SELECT**

Allows SELECT from any column of the specified table, view, or sequence. Also allows the use of COPY TO.

**INSERT**

Allows INSERT of a new row into the specified table. Also allows COPY FROM.

**UPDATE**

Allows UPDATE of any column of the specified table. SELECT ... FOR UPDATE and SELECT ... FOR SHARE also require this privilege (as well as the SELECT privilege). For sequences, this privilege allows the use of the `nextval()` and `setval()` functions.

**DELETE**

Allows DELETE of a row from the specified table.

**REFERENCES**

This keyword is accepted, although foreign key constraints are currently not supported in Greenplum Database. To create a foreign key constraint, it is necessary to have this privilege on both the referencing and referenced tables.

**TRIGGER**

Allows the creation of a trigger on the specified table.

  **Note:** Greenplum Database does not support triggers.

**TRUNCATE**

Allows TRUNCATE of all rows from the specified table.
CREATE

For databases, allows new schemas to be created within the database.
For schemas, allows new objects to be created within the schema. To rename an existing object, you must own the object and have this privilege for the containing schema.
For tablespaces, allows tables and indexes to be created within the tablespace, and allows databases to be created that have the tablespace as their default tablespace. (Note that revoking this privilege will not alter the placement of existing objects.)

CONNECT

Allows the user to connect to the specified database. This privilege is checked at connection startup (in addition to checking any restrictions imposed by pg_hba.conf).

TEMPORARY

TEMP

Allows temporary tables to be created while using the database.

EXECUTE

Allows the use of the specified function and the use of any operators that are implemented on top of the function. This is the only type of privilege that is applicable to functions. (This syntax works for aggregate functions, as well.)

USAGE

For procedural languages, allows the use of the specified language for the creation of functions in that language. This is the only type of privilege that is applicable to procedural languages.
For schemas, allows access to objects contained in the specified schema (assuming that the objects' own privilege requirements are also met). Essentially this allows the grantee to look up objects within the schema.
For sequences, this privilege allows the use of the nextval() function.

ALL PRIVILEGES

Grant all of the available privileges at once. The PRIVILEGES key word is optional in Greenplum Database, though it is required by strict SQL.

PUBLIC

A special group-level role that denotes that the privileges are to be granted to all roles, including those that may be created later.

WITH GRANT OPTION

The recipient of the privilege may in turn grant it to others.

WITH ADMIN OPTION

The member of a role may in turn grant membership in the role to others.

Notes

Database superusers can access all objects regardless of object privilege settings. One exception to this rule is view objects. Access to tables referenced in the view is determined by permissions of the view owner not the current user (even if the current user is a superuser).

If a superuser chooses to issue a GRANT or REVOKE command, the command is performed as though it were issued by the owner of the affected object. In particular, privileges granted via such a command will appear to have been granted by the object owner. For role membership, the membership appears to have been granted by the containing role itself.

GRANT and REVOKE can also be done by a role that is not the owner of the affected object, but is a member of the role that owns the object, or is a member of a role that holds privileges WITH GRANT
OPTION on the object. In this case the privileges will be recorded as having been granted by the role that actually owns the object or holds the privileges WITH GRANT OPTION.

Granting permission on a table does not automatically extend permissions to any sequences used by the table, including sequences tied to SERIAL columns. Permissions on a sequence must be set separately.

Greenplum Database does not support granting or revoking privileges for individual columns of a table. One possible workaround is to create a view having just the desired columns and then grant privileges to that view.

Use psql's \z meta-command to obtain information about existing privileges for an object.

Examples

Grant insert privilege to all roles on table mytable:

GRANT INSERT ON mytable TO PUBLIC;

Grant all available privileges to role sally on the view topten. Note that while the above will indeed grant all privileges if executed by a superuser or the owner of topten, when executed by someone else it will only grant those permissions for which the granting role has grant options.

GRANT ALL PRIVILEGES ON topten TO sally;

Grant membership in role admins to user joe:

GRANT admins TO joe;

Compatibility

The PRIVILEGES key word in is required in the SQL standard, but optional in Greenplum Database. The SQL standard does not support setting the privileges on more than one object per command.

Greenplum Database allows an object owner to revoke his own ordinary privileges: for example, a table owner can make the table read-only to himself by revoking his own INSERT, UPDATE, DELETE, and TRUNCATE privileges. This is not possible according to the SQL standard. Greenplum Database treats the owner's privileges as having been granted by the owner to himself; therefore he can revoke them too. In the SQL standard, the owner's privileges are granted by an assumed system entity.

The SQL standard allows setting privileges for individual columns within a table.

The SQL standard provides for a USAGE privilege on other kinds of objects: character sets, collations, translations, domains.

Privileges on databases, tablespaces, schemas, and languages are Greenplum Database extensions.

See Also

REVOKE

INSERT

Creates new rows in a table.

Synopsis

INSERT INTO table [( column [, ...] )]
{DEFAULT VALUES | VALUES ( {expression | DEFAULT} [, ...] )}
[, ...] | query}
Description

`INSERT` inserts new rows into a table. One can insert one or more rows specified by value expressions, or zero or more rows resulting from a query.

The target column names may be listed in any order. If no list of column names is given at all, the default is the columns of the table in their declared order. The values supplied by the `VALUES` clause or query are associated with the explicit or implicit column list left-to-right.

Each column not present in the explicit or implicit column list will be filled with a default value, either its declared default value or null if there is no default.

If the expression for any column is not of the correct data type, automatic type conversion will be attempted.

You must have `INSERT` privilege on a table in order to insert into it.

Outputs

On successful completion, an `INSERT` command returns a command tag of the form:

```
INSERT oid count
```

The `count` is the number of rows inserted. If count is exactly one, and the target table has OIDs, then `oid` is the OID assigned to the inserted row. Otherwise `oid` is zero.

Parameters

`table`

The name (optionally schema-qualified) of an existing table.

`column`

The name of a column in table. The column name can be qualified with a subfield name or array subscript, if needed. (Inserting into only some fields of a composite column leaves the other fields null.)

`DEFAULT VALUES`

All columns will be filled with their default values.

`expression`

An expression or value to assign to the corresponding column.

`DEFAULT`

The corresponding column will be filled with its default value.

`query`

A query (SELECT statement) that supplies the rows to be inserted. Refer to the SELECT statement for a description of the syntax.

Notes

To insert data into a partitioned table, you specify the root partitioned table, the table created with the `CREATE TABLE` command. You also can specify a leaf child table of the partitioned table in an `INSERT` command. An error is returned if the data is not valid for the specified leaf child table. Specifying a child table that is not a leaf child table in the `INSERT` command is not supported. Execution of other DML commands such as `UPDATE` and `DELETE` on any child table of a partitioned table is not supported. These commands must be executed on the root partitioned table, the table created with the `CREATE TABLE` command.

For append-optimized tables, Greenplum Database supports a maximum of 127 concurrent `INSERT` transactions into a single append-optimized table.
For writable S3 external tables, the INSERT operation uploads to one or more files in the configured S3 bucket, as described in s3:// Protocol. Pressing Ctrl-c cancels the INSERT and stops uploading to S3.

**Examples**

Insert a single row into table films:

```sql
INSERT INTO films VALUES ('UA502', 'Bananas', 105, '1971-07-13', 'Comedy', '82 minutes');
```

In this example, the length column is omitted and therefore it will have the default value:

```sql
INSERT INTO films (code, title, did, date_prod, kind) VALUES ('T_601', 'Yojimbo', 106, '1961-06-16', 'Drama');
```

This example uses the DEFAULT clause for the date_prod column rather than specifying a value:

```sql
INSERT INTO films VALUES ('UA502', 'Bananas', 105, DEFAULT, 'Comedy', '82 minutes');
```

To insert a row consisting entirely of default values:

```sql
INSERT INTO films DEFAULT VALUES;
```

To insert multiple rows using the multirow VALUES syntax:

```sql
INSERT INTO films (code, title, did, date_prod, kind) VALUES ('B6717', 'Tampopo', 110, '1985-02-10', 'Comedy'), ('HG120', 'The Dinner Game', 140, DEFAULT, 'Comedy');
```

This example inserts some rows into table films from a table tmp_films with the same column layout as films:

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

**Compatibility**

INSERT conforms to the SQL standard. The case in which a column name list is omitted, but not all the columns are filled from the VALUES clause or query, is disallowed by the standard.

Possible limitations of the query clause are documented under SELECT.

**See Also**

COPY, SELECT, CREATE EXTERNAL TABLE, s3:// Protocol

**LOAD**

Loads or reloads a shared library file.

**Synopsis**

LOAD 'filename'
Description
This command loads a shared library file into the Greenplum Database server address space. If the file had been loaded previously, it is first unloaded. This command is primarily useful to unload and reload a shared library file that has been changed since the server first loaded it. To make use of the shared library, function(s) in it need to be declared using the `CREATE FUNCTION` command.

The file name is specified in the same way as for shared library names in `CREATE FUNCTION`; in particular, one may rely on a search path and automatic addition of the system's standard shared library file name extension.

Note that in Greenplum Database the shared library file (.so file) must reside in the same path location on every host in the Greenplum Database array (masters, segments, and mirrors).

Only database superusers can load shared library files.

Parameters
`filename`

The path and file name of a shared library file. This file must exist in the same location on all hosts in your Greenplum Database array.

Examples
Load a shared library file:

```sql
LOAD '/usr/local/greenplum-db/lib/myfuncs.so';
```

Compatibility
`LOAD` is a Greenplum Database extension.

See Also
`CREATE FUNCTION`

LOCK

Locks a table.

Synopsis

```
LOCK [TABLE] name [, ...] [IN lockmode MODE] [NOWAIT]
```

where `lockmode` is one of:

```
ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE
| SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE
```

Description

`LOCK TABLE` obtains a table-level lock, waiting if necessary for any conflicting locks to be released. If `NOWAIT` is specified, `LOCK TABLE` does not wait to acquire the desired lock: if it cannot be acquired immediately, the command is aborted and an error is emitted. Once obtained, the lock is held for the remainder of the current transaction. There is no `UNLOCK TABLE` command; locks are always released at transaction end.

When acquiring locks automatically for commands that reference tables, Greenplum Database always uses the least restrictive lock mode possible. `LOCK TABLE` provides for cases when you might need more
restrictive locking. For example, suppose an application runs a transaction at the Read Committed isolation level and needs to ensure that data in a table remains stable for the duration of the transaction. To achieve this, you could obtain SHARE lock mode over the table before querying. This will prevent concurrent data changes and ensure subsequent reads of the table see a stable view of committed data, because SHARE lock mode conflicts with the ROW EXCLUSIVE lock acquired by writers, and your LOCK TABLE name IN SHARE MODE statement will wait until any concurrent holders of ROW EXCLUSIVE mode locks commit or roll back. Thus, once you obtain the lock, there are no uncommitted writes outstanding; furthermore none can begin until you release the lock.

To achieve a similar effect when running a transaction at the Serializable isolation level, you have to execute the LOCK TABLE statement before executing any SELECT or data modification statement. A serializable transaction’s view of data will be frozen when its first SELECT or data modification statement begins. A LOCK TABLE later in the transaction will still prevent concurrent writes — but it won’t ensure that what the transaction reads corresponds to the latest committed values.

If a transaction of this sort is going to change the data in the table, then it should use SHARE ROW EXCLUSIVE lock mode instead of SHARE mode. This ensures that only one transaction of this type runs at a time. Without this, a deadlock is possible: two transactions might both acquire SHARE mode, and then be unable to also acquire ROW EXCLUSIVE mode to actually perform their updates. Note that a transaction’s own locks never conflict, so a transaction can acquire ROW EXCLUSIVE mode when it holds SHARE mode — but not if anyone else holds SHARE mode. To avoid deadlocks, make sure all transactions acquire locks on the same objects in the same order, and if multiple lock modes are involved for a single object, then transactions should always acquire the most restrictive mode first.

**Parameters**

**name**

The name (optionally schema-qualified) of an existing table to lock.

If multiple tables are given, tables are locked one-by-one in the order specified in the LOCK TABLE command.

**lockmode**

The lock mode specifies which locks this lock conflicts with. If no lock mode is specified, then ACCESS EXCLUSIVE, the most restrictive mode, is used. Lock modes are as follows:

- **ACCESS SHARE** — Conflicts with the ACCESS EXCLUSIVE lock mode only. The commands SELECT and ANALYZE automatically acquire a lock of this mode on referenced tables. In general, any query that only reads a table and does not modify it will acquire this lock mode.
- **ROW SHARE** — Conflicts with the EXCLUSIVE and ACCESS EXCLUSIVE lock modes. The SELECT FOR SHARE command automatically acquires a lock of this mode on the target table(s) (in addition to ACCESS SHARE locks on any other tables that are referenced but not selected FOR SHARE).
- **ROW EXCLUSIVE** — Conflicts with the SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, and ACCESS EXCLUSIVE lock modes. The commands INSERT and COPY automatically acquire this lock mode on the target table (in addition to ACCESS SHARE locks on any other referenced tables).
- **SHARE UPDATE EXCLUSIVE** — Conflicts with the SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, and ACCESS EXCLUSIVE lock modes. This mode protects a table against concurrent schema changes and VACUUM runs. Acquired automatically by VACUUM (without FULL) on heap tables.
- **SHARE** — Conflicts with the ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE ROW EXCLUSIVE, EXCLUSIVE, and ACCESS EXCLUSIVE lock modes. This mode protects a table against concurrent data changes. Acquired automatically by CREATE INDEX.
- **SHARE ROW EXCLUSIVE** — Conflicts with the ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, and ACCESS EXCLUSIVE lock modes.
EXCLUSIVE lock modes. This lock mode is not automatically acquired by any Greenplum Database command.

- **EXCLUSIVE** — Conflicts with the ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, and ACCESS EXCLUSIVE lock modes. This mode allows only concurrent ACCESS SHARE locks, i.e., only reads from the table can proceed in parallel with a transaction holding this lock mode. This lock mode is automatically acquired for UPDATE, SELECT FOR UPDATE, and DELETE in Greenplum Database (which is more restrictive locking than in regular PostgreSQL).

- **ACCESS EXCLUSIVE** — Conflicts with locks of all modes (ACCESS SHARE, ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, and ACCESS EXCLUSIVE). This mode guarantees that the holder is the only transaction accessing the table in any way. Acquired automatically by the ALTER TABLE, DROP TABLE, TRUNCATE, REINDEX, CLUSTER, and VACUUM FULL commands. This is the default lock mode for LOCK TABLE statements that do not specify a mode explicitly. This lock is also briefly acquired by VACUUM (without FULL) on append-optimized tables during processing.

**NOWAIT**

Specifies that LOCK TABLE should not wait for any conflicting locks to be released: if the specified lock(s) cannot be acquired immediately without waiting, the transaction is aborted.

**Notes**

LOCK TABLE ... IN ACCESS SHARE MODE requires SELECT privileges on the target table. All other forms of LOCK require UPDATE and/or DELETE privileges.

LOCK TABLE is useful only inside a transaction block (BEGIN/COMMIT pair), since the lock is dropped as soon as the transaction ends. A LOCK TABLE command appearing outside any transaction block forms a self-contained transaction, so the lock will be dropped as soon as it is obtained.

LOCK TABLE only deals with table-level locks, and so the mode names involving ROW are all misnomers. These mode names should generally be read as indicating the intention of the user to acquire row-level locks within the locked table. Also, ROW EXCLUSIVE mode is a sharable table lock. Keep in mind that all the lock modes have identical semantics so far as LOCK TABLE is concerned, differing only in the rules about which modes conflict with which. For information on how to acquire an actual row-level lock, see the FOR UPDATE/FOR SHARE clause in the SELECT reference documentation.

**Examples**

Obtain a SHARE lock on the films table when going to perform inserts into the films_user_comments table:

```sql
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;
```

Take a SHARE ROW EXCLUSIVE lock on a table when performing a delete operation:

```sql
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;

Compatibility

There is no `LOCK TABLE` in the SQL standard, which instead uses `SET TRANSACTION` to specify concurrency levels on transactions. Greenplum Database supports that too.

Except for `ACCESS SHARE`, `ACCESS EXCLUSIVE`, and `SHARE UPDATE EXCLUSIVE` lock modes, the Greenplum Database lock modes and the `LOCK TABLE` syntax are compatible with those present in Oracle.

See Also

`BEGIN`, `SET TRANSACTION`, `SELECT`

MOVE

Positions a cursor.

Synopsis

```
MOVE [ forward_direction {FROM | IN} ] cursorname
```

where `forward_direction` can be empty or one of:

```
NEXT
FIRST
LAST
ABSOLUTE count
RELATIVE count
count
ALL
FORWARD
FORWARD count
FORWARD ALL
```

Description

`MOVE` repositions a cursor without retrieving any data. `MOVE` works exactly like the `FETCH` command, except it only positions the cursor and does not return rows.

Note that it is not possible to move a cursor position backwards in Greenplum Database, since scrollable cursors are not supported. You can only move a cursor forward in position using `MOVE`.

Outputs

On successful completion, a `MOVE` command returns a command tag of the form

```
MOVE count
```

The count is the number of rows that a `FETCH` command with the same parameters would have returned (possibly zero).

Parameters

`forward_direction`

See `FETCH` for more information.

`cursorname`
The name of an open cursor.

**Examples**

-- Start the transaction:

```
BEGIN;
```

-- Set up a cursor:

```
DECLARE mycursor CURSOR FOR SELECT * FROM films;
```

-- Move forward 5 rows in the cursor mycursor:

```
MOVE FORWARD 5 IN mycursor;
MOVE 5
```

-- Fetch the next row after that (row 6):

```
FETCH 1 FROM mycursor;
```

<table>
<thead>
<tr>
<th>code</th>
<th>title</th>
<th>did</th>
<th>date_prod</th>
<th>kind</th>
<th>len</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_303</td>
<td>48 Hrs</td>
<td>103</td>
<td>1982-10-22</td>
<td>Action</td>
<td>01:37</td>
</tr>
</tbody>
</table>

(1 row)

-- Close the cursor and end the transaction:

```
CLOSE mycursor;
COMMIT;
```

**Compatibility**

There is no MOVE statement in the SQL standard.

**See Also**

`DECLARE`, `FETCH`, `CLOSE`

**PREPARE**

Prepare a statement for execution.

**Synopsis**

```
PREPARE name [ (datatype [, ...] ) ] AS statement
```

**Description**

`PREPARE` creates a prepared statement, possibly with unbound parameters. A prepared statement is a server-side object that can be used to optimize performance. A prepared statement may be subsequently executed with a binding for its parameters. Greenplum Database may choose to replan the query for different executions of the same prepared statement.

Prepared statements can take parameters: values that are substituted into the statement when it is executed. When creating the prepared statement, refer to parameters by position, using $1, $2, etc. A corresponding list of parameter data types can optionally be specified. When a parameter's data type is not specified or is declared as unknown, the type is inferred from the context in which the parameter is used (if
When executing the statement, specify the actual values for these parameters in the `EXECUTE` statement.

Prepared statements only last for the duration of the current database session. When the session ends, the prepared statement is forgotten, so it must be recreated before being used again. This also means that a single prepared statement cannot be used by multiple simultaneous database clients; however, each client can create their own prepared statement to use. The prepared statement can be manually cleaned up using the `DEALLOCATE` command.

Prepared statements have the largest performance advantage when a single session is being used to execute a large number of similar statements. The performance difference will be particularly significant if the statements are complex to plan or rewrite, for example, if the query involves a join of many tables or requires the application of several rules. If the statement is relatively simple to plan and rewrite but relatively expensive to execute, the performance advantage of prepared statements will be less noticeable.

### Parameters

- **name**
  - An arbitrary name given to this particular prepared statement. It must be unique within a single session and is subsequently used to execute or deallocate a previously prepared statement.

- **datatype**
  - The data type of a parameter to the prepared statement. If the data type of a particular parameter is unspecified or is specified as unknown, it will be inferred from the context in which the parameter is used. To refer to the parameters in the prepared statement itself, use `$1`, `$2`, etc.

- **statement**
  - Any `SELECT`, `INSERT`, `UPDATE`, `DELETE`, or `VALUES` statement.

### Notes

In some situations, the query plan produced for a prepared statement will be inferior to the query plan that would have been chosen if the statement had been submitted and executed normally. This is because when the statement is planned and the planner attempts to determine the optimal query plan, the actual values of any parameters specified in the statement are unavailable. Greenplum Database collects statistics on the distribution of data in the table, and can use constant values in a statement to make guesses about the likely result of executing the statement. Since this data is unavailable when planning prepared statements with parameters, the chosen plan may be suboptimal. To examine the query plan Greenplum Database has chosen for a prepared statement, use `EXPLAIN`.

For more information on query planning and the statistics collected by Greenplum Database for that purpose, see the `ANALYZE` documentation.

You can see all available prepared statements of a session by querying the `pg_prepared_statements` system view.

### Examples

Create a prepared statement for an `INSERT` statement, and then execute it:

```sql
PREPARE fooplan (int, text, bool, numeric) AS INSERT INTO foo VALUES($1, $2, $3, $4);
EXECUTE fooplan(1, 'Hunter Valley', 't', 200.00);
```

Create a prepared statement for a `SELECT` statement, and then execute it. Note that the data type of the second parameter is not specified, so it is inferred from the context in which `$2` is used:

```sql
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(l, current_date);

Compatibility
The SQL standard includes a PREPARE statement, but it is only for use in embedded SQL. This version of the PREPARE statement also uses a somewhat different syntax.

See Also
EXECUTE, DEALLOCATE

REASSIGN OWNED
Changes the ownership of database objects owned by a database role.

Synopsis

REASSIGN OWNED BY old_role [, ...] TO new_role

Description
REASSIGN OWNED reassigns all the objects in the current database that are owned by old_role to new_role. Note that it does not change the ownership of the database itself.

Parameters

old_role
The name of a role. The ownership of all the objects in the current database owned by this role will be reassigned to new_role.

new_role
The name of the role that will be made the new owner of the affected objects.

Notes
REASSIGN OWNED is often used to prepare for the removal of one or more roles. Because REASSIGN OWNED only affects the objects in the current database, it is usually necessary to execute this command in each database that contains objects owned by a role that is to be removed.

The DROP OWNED command is an alternative that drops all the database objects owned by one or more roles.

The REASSIGN OWNED command does not affect the privileges granted to the old roles in objects that are not owned by them. Use DROP OWNED to revoke those privileges.

Examples
Reassign any database objects owned by the role named sally and bob to admin;

REASSIGN OWNED BY sally, bob TO admin;

Compatibility
The REASSIGN OWNED statement is a Greenplum Database extension.
### See Also

DROP OWNED, DROP ROLE

---

## REINDEX

Rebuilds indexes.

### Synopsis

```sql
REINDEX {INDEX | TABLE | DATABASE | SYSTEM} name
```

### Description

REINDEX rebuilds an index using the data stored in the index's table, replacing the old copy of the index. There are several scenarios in which to use REINDEX:

- An index has become bloated, that it is contains many empty or nearly-empty pages. This can occur with B-tree indexes in Greenplum Database under certain uncommon access patterns. REINDEX provides a way to reduce the space consumption of the index by writing a new version of the index without the dead pages.
- You have altered the fillfactor storage parameter for an index, and wish to ensure that the change has taken full effect.

### Parameters

**INDEX**

Recreate the specified index.

**TABLE**

Recreate all indexes of the specified table. If the table has a secondary TOAST table, that is reindexed as well.

**DATABASE**

Recreate all indexes within the current database. Indexes on shared system catalogs are skipped. This form of REINDEX cannot be executed inside a transaction block.

**SYSTEM**

Recreate all indexes on system catalogs within the current database. Indexes on user tables are not processed. Also, indexes on shared (global) system catalogs are skipped. This form of REINDEX cannot be executed inside a transaction block.

**name**

The name of the specific index, table, or database to be reindexed. Index and table names may be schema-qualified. Presently, `REINDEX DATABASE` and `REINDEX SYSTEM` can only reindex the current database, so their parameter must match the current database's name.

### Notes

REINDEX is similar to a drop and recreate of the index in that the index contents are rebuilt from scratch. However, the locking considerations are rather different. REINDEX locks out writes but not reads of the index's parent table. It also takes an exclusive lock on the specific index being processed, which will block reads that attempt to use that index. In contrast, DROP INDEX momentarily takes exclusive lock on the parent table, blocking both writes and reads. The subsequent CREATE INDEX locks out writes but not reads; since the index is not there, no read will attempt to use it, meaning that there will be no blocking but reads may be forced into expensive sequential scans.
Reindexing a single index or table requires being the owner of that index or table. Reindexing a database requires being the owner of the database (note that the owner can therefore rebuild indexes of tables owned by other users). Of course, superusers can always reindex anything.

If you suspect that shared global system catalog indexes are corrupted, they can only be reindexed in Greenplum utility mode. The typical symptom of a corrupt shared index is "index is not a btree" errors, or else the server crashes immediately at startup due to reliance on the corrupted indexes. Contact Greenplum Customer Support for assistance in this situation.

**Examples**

Rebuild a single index:

```
REINDEX INDEX my_index;
```

Rebuild all the indexes on the table `my_table`:

```
REINDEX TABLE my_table;
```

**Compatibility**

There is no `REINDEX` command in the SQL standard.

**See Also**

`CREATE INDEX`, `DROP INDEX`, `VACUUM`

---

**RELEASE SAVEPOINT**

Destroys a previously defined savepoint.

**Synopsis**

```
RELEASE [SAVEPOINT] savepoint_name
```

**Description**

`RELEASE SAVEPOINT` destroys a savepoint previously defined in the current transaction.

Destroying a savepoint makes it unavailable as a rollback point, but it has no other user visible behavior. It does not undo the effects of commands executed after the savepoint was established. (To do that, see `ROLLBACK TO SAVEPOINT`.) Destroying a savepoint when it is no longer needed may allow the system to reclaim some resources earlier than transaction end.

`RELEASE SAVEPOINT` also destroys all savepoints that were established *after* the named savepoint was established.

**Parameters**

*savepoint_name*

The name of the savepoint to destroy.

**Examples**

To establish and later destroy a savepoint:

```
BEGIN;
    INSERT INTO table1 VALUES (3);
SAVEPOINT my_savepoint;
```
The above transaction will insert both 3 and 4.

Compatibility
This command conforms to the SQL standard. The standard specifies that the key word `SAVEPOINT` is mandatory, but Greenplum Database allows it to be omitted.

See Also
`BEGIN`, `SAVEPOINT`, `ROLLBACK TO SAVEPOINT`, `COMMIT`

**RESET**
Restores the value of a system configuration parameter to the default value.

**Synopsis**

```sql
RESET configuration_parameter
RESET ALL
```

**Description**
`RESET` restores system configuration parameters to their default values. `RESET` is an alternative spelling for `SET configuration_parameter TO DEFAULT`.

The default value is defined as the value that the parameter would have had, had no `SET` ever been issued for it in the current session. The actual source of this value might be a compiled-in default, the master `postgresql.conf` configuration file, command-line options, or per-database or per-user default settings. See Server Configuration Parameters for more information.

**Parameters**

- `configuration_parameter`
  The name of a system configuration parameter. See Server Configuration Parameters for details.

- `ALL`
  Resets all settable configuration parameters to their default values.

**Examples**
Set the `statement_mem` configuration parameter to its default value:

```sql
RESET statement_mem;
```

**Compatibility**
`RESET` is a Greenplum Database extension.

**See Also**
`SET`
**REVOKE**

Removes access privileges.

### Synopsis

```sql
REVOKE [GRANT OPTION FOR] { {SELECT | INSERT | UPDATE | DELETE
 | REFERENCES | TRIGGER | TRUNCATE } [, ...] | ALL [PRIVILEGES] }
   ON [TABLE] tablename [, ...]
   FROM { rolename | PUBLIC } [, ...]
   [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { {USAGE | SELECT | UPDATE} [, ...]
 | ALL [PRIVILEGES] }
   ON SEQUENCE sequencename [, ...]
   FROM { rolename | PUBLIC } [, ...]
   [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { {CREATE | CONNECT
 | TEMPORARY | TEMP} [, ...] | ALL [PRIVILEGES] }
   ON DATABASE dbname [, ...]
   FROM { rolename | PUBLIC } [, ...]
   [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] {EXECUTE | ALL [PRIVILEGES]}
   ON FUNCTION funcname ( [[argmode] argname argtype
   [, ...]] ) [, ...]
   FROM { rolename | PUBLIC } [, ...]
   [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { {USAGE | ALL [PRIVILEGES]}
   ON LANGUAGE langname [, ...]
   FROM { rolename | PUBLIC } [, ...]
   [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { {CREATE | USAGE} [, ...]
 | ALL [PRIVILEGES] }
   ON SCHEMA schemaname [, ...]
   FROM { rolename | PUBLIC } [, ...]
   [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { CREATE | ALL [PRIVILEGES] }
   ON TABLESPACE tablespacename [, ...]
   FROM { rolename | PUBLIC } [, ...]
   [CASCADE | RESTRICT]

REVOKE [ADMIN OPTION FOR] parent_role [, ...]
   FROM member_role [, ...]
   [CASCADE | RESTRICT]
```

### Description

The **REVOKE** command revokes previously granted privileges from one or more roles. The key word **PUBLIC** refers to the implicitly defined group of all roles.

See the description of the **GRANT** command for the meaning of the privilege types.

Note that any particular role will have the sum of privileges granted directly to it, privileges granted to any role it is presently a member of, and privileges granted to **PUBLIC**. Thus, for example, revoking **SELECT**
privilege from PUBLIC does not necessarily mean that all roles have lost SELECT privilege on the object: those who have it granted directly or via another role will still have it.

If GRANT OPTION FOR is specified, only the grant option for the privilege is revoked, not the privilege itself. Otherwise, both the privilege and the grant option are revoked.

If a role holds a privilege with grant option and has granted it to other roles then the privileges held by those other roles are called dependent privileges. If the privilege or the grant option held by the first role is being revoked and dependent privileges exist, those dependent privileges are also revoked if CASCADE is specified, else the revoke action will fail. This recursive revocation only affects privileges that were granted through a chain of roles that is traceable to the role that is the subject of this REVOKE command. Thus, the affected roles may effectively keep the privilege if it was also granted through other roles.

When revoking membership in a role, GRANT OPTION is instead called ADMIN OPTION, but the behavior is similar.

Parameters
See GRANT.

Examples
Revoke insert privilege for the public on table films:

```
REVOKE INSERT ON films FROM PUBLIC;
```

Revoke all privileges from role sally on view topten. Note that this actually means revoke all privileges that the current role granted (if not a superuser).

```
REVOKE ALL PRIVILEGES ON topten FROM sally;
```

Revoke membership in role admins from user joe:

```
REVOKE admins FROM joe;
```

Compatibility
The compatibility notes of the GRANT command also apply to REVOKE.

Either RESTRICT or CASCADE is required according to the standard, but Greenplum Database assumes RESTRICT by default.

See Also
GRANT

ROLLBACK
Aborts the current transaction.

Synopsis

```
ROLLBACK [WORK | TRANSACTION]
```

Description

ROLLBACK rolls back the current transaction and causes all the updates made by the transaction to be discarded.
Parameters

WORK
TRANSACTION

Optional key words. They have no effect.

Notes

Use COMMIT to successfully end the current transaction.

Issuing ROLLBACK when not inside a transaction does no harm, but it will provoke a warning message.

Examples

To discard all changes made in the current transaction:

ROLLBACK;

Compatibility

The SQL standard only specifies the two forms ROLLBACK and ROLLBACK WORK. Otherwise, this command is fully conforming.

See Also

BEGIN, COMMIT, SAVEPOINT, ROLLBACK TO SAVEPOINT

**ROLLBACK TO SAVEPOINT**

Rolls back the current transaction to a savepoint.

Synopsis

ROLLBACK [WORK | TRANSACTION] TO [SAVEPOINT] savepoint_name

Description

This command will roll back all commands that were executed after the savepoint was established. The savepoint remains valid and can be rolled back to again later, if needed.

ROLLBACK TO SAVEPOINT implicitly destroys all savepoints that were established after the named savepoint.

Parameters

WORK
TRANSACTION

Optional key words. They have no effect.

savepoint_name

The name of a savepoint to roll back to.

Notes

Use RELEASE SAVEPOINT to destroy a savepoint without discarding the effects of commands executed after it was established.

Specifying a savepoint name that has not been established is an error.
Cursors have somewhat non-transactional behavior with respect to savepoints. Any cursor that is opened inside a savepoint will be closed when the savepoint is rolled back. If a previously opened cursor is affected by a \texttt{FETCH} command inside a savepoint that is later rolled back, the cursor position remains at the position that \texttt{FETCH} left it pointing to (that is, \texttt{FETCH} is not rolled back). Closing a cursor is not undone by rolling back, either. A cursor whose execution causes a transaction to abort is put in a can't-execute state, so while the transaction can be restored using \texttt{ROLLBACK TO SAVEPOINT}, the cursor can no longer be used.

**Examples**

To undo the effects of the commands executed after my\_savepoint was established:

\begin{verbatim}
ROLLBACK TO SAVEPOINT my\_savepoint;
\end{verbatim}

Cursor positions are not affected by a savepoint rollback:

\begin{verbatim}
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;
\end{verbatim}

**Compatibility**

The SQL standard specifies that the key word \texttt{SAVEPOINT} is mandatory, but Greenplum Database (and Oracle) allow it to be omitted. SQL allows only \texttt{WORK}, not \texttt{TRANSACTION}, as a noise word after \texttt{ROLLBACK}. Also, SQL has an optional clause \texttt{AND [NO] CHAIN} which is not currently supported by Greenplum Database. Otherwise, this command conforms to the SQL standard.

**See Also**

\texttt{BEGIN, COMMIT, SAVEPOINT, RELEASE SAVEPOINT, ROLLBACK}

**SAVEPOINT**

Defines a new savepoint within the current transaction.

**Synopsis**

\begin{verbatim}
SAVEPOINT savepoint_name
\end{verbatim}

**Description**

\texttt{SAVEPOINT} establishes a new savepoint within the current transaction.

A savepoint is a special mark inside a transaction that allows all commands that are executed after it was established to be rolled back, restoring the transaction state to what it was at the time of the savepoint.

**Parameters**

\begin{verbatim}
savepoint_name
\end{verbatim}
The name of the new savepoint.

**Notes**

Use `ROLLBACK TO SAVEPOINT` to rollback to a savepoint. Use `RELEASE SAVEPOINT` to destroy a savepoint, keeping the effects of commands executed after it was established.

Savepoints can only be established when inside a transaction block. There can be multiple savepoints defined within a transaction.

**Examples**

To establish a savepoint and later undo the effects of all commands executed after it was established:

```sql
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;
```

The above transaction will insert the values 1 and 3, but not 2.

To establish and later destroy a savepoint:

```sql
BEGIN;
  INSERT INTO table1 VALUES (3);
  SAVEPOINT my_savepoint;
  INSERT INTO table1 VALUES (4);
  RELEASE SAVEPOINT my_savepoint;
COMMIT;
```

The above transaction will insert both 3 and 4.

**Compatibility**

SQL requires a savepoint to be destroyed automatically when another savepoint with the same name is established. In Greenplum Database, the old savepoint is kept, though only the more recent one will be used when rolling back or releasing. (Releasing the newer savepoint will cause the older one to again become accessible to `ROLLBACK TO SAVEPOINT` and `RELEASE SAVEPOINT`.) Otherwise, `SAVEPOINT` is fully SQL conforming.

**See Also**

`BEGIN`, `COMMIT`, `ROLLBACK`, `RELEASE SAVEPOINT`, `ROLLBACK TO SAVEPOINT`

**SELECT**

Retrieves rows from a table or view.

**Synopsis**

```
[ WITH [ RECURSIVE ] 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 specification)]
[UNION | INTERSECT | EXCEPT] [ALL] select
[ORDER BY expression [ASC | DESC | USING operator] [NULLS [FIRST | LAST]]
[, ...]
[LIMIT {count | ALL}]
OFFSET start
[FOR {UPDATE | SHARE} [OF table_name [, ...]] [NOWAIT] [...]]

where with_query: is:

with_query_name [( column_name [, ...] )] AS ( select )

where grouping_element can be one of:

() expression
ROLLUP (expression [, ...])
CUBE (expression [, ...])
GROUPING SETS ( (grouping_element [, ...]) )

where window_specification can be:

[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 ]]]]
where window_frame_bound can be one of:
UNBOUNDED PRECEDING
expression PRECEDING
CURRENT ROW
expression FOLLOWING
UNBOUNDED FOLLOWING

where from_item can be one of:

[ONLY] table_name [[AS] alias [( column_alias [, ...] )]]
(select) [AS] alias [( column_alias [, ...] )]
with_query_name [ AS] alias [( column_alias [, ...] )]
function_name ( [argument [, ...]] ) [AS] alias
[( column_alias [, ...] )
 | column_definition [, ...] ]
function_name ( [argument [, ...]] ) AS
( column_definition [, ...] )
from_item [NATURAL] join_type from_item
[ON join_condition | USING ( join_column [, ...] )]

Note: The RECURSIVE keyword is an experimental feature and is not recommended for production use.

Description

SELECT retrieves rows from zero or more tables. The general processing of SELECT is as follows:

1. All queries in the WITH clause are computed. These effectively serve as temporary tables that can be referenced in the FROM list.
2. All elements in the FROM list are computed. (Each element in the FROM list is a real or virtual table.) If more than one element is specified in the FROM list, they are cross-joined together.
3. If the **WHERE** clause is specified, all rows that do not satisfy the condition are eliminated from the output.

4. If the **GROUP BY** clause is specified, the output is divided into groups of rows that match on one or more of the defined grouping elements. If the **HAVING** clause is present, it eliminates groups that do not satisfy the given condition.

5. If a window expression is specified (and optional **WINDOW** clause), the output is organized according to the positional (row) or value-based (range) window frame.

6. **DISTINCT** eliminates duplicate rows from the result. **DISTINCT ON** eliminates rows that match on all the specified expressions. **ALL** (the default) will return all candidate rows, including duplicates.

7. The actual output rows are computed using the **SELECT** output expressions for each selected row.

8. Using the operators **UNION**, **INTERSECT**, and **EXCEPT**, the output of more than one **SELECT** statement can be combined to form a single result set. The **UNION** operator returns all rows that are in one or both of the result sets. The **INTERSECT** operator returns all rows that are strictly in both result sets. The **EXCEPT** operator returns the rows that are in the first result set but not in the second. In all three cases, duplicate rows are eliminated unless **ALL** is specified.

9. If the **ORDER BY** clause is specified, the returned rows are sorted in the specified order. If **ORDER BY** is not given, the rows are returned in whatever order the system finds fastest to produce.

10. If the **LIMIT** or **OFFSET** clause is specified, the **SELECT** statement only returns a subset of the result rows.

11. If **FOR UPDATE** or **FOR SHARE** is specified, the **SELECT** statement locks the entire table against concurrent updates.

You must have **SELECT** privilege on a table to read its values. The use of **FOR UPDATE** or **FOR SHARE** requires **UPDATE** privilege as well.

### Parameters

#### The WITH Clause

The optional **WITH** clause allows you to specify one or more subqueries that can be referenced by name in the primary query. The subqueries effectively act as temporary tables or views for the duration of the primary query. Each subquery can be a **SELECT**, or **VALUES** command.

A **with_query_name** without schema qualification must be specified for each query in the **WITH** clause. Optionally, a list of column names can be specified; if the list of column names is omitted, the names are inferred from the subquery. The primary query and the **WITH** queries are all (notionally) executed at the same time.

The **RECURSIVE** keyword can be enabled by setting the server configuration parameter **gp_recursive_cte_prototype** to **true**. For information about the parameter, see **Server Configuration Parameters**.

**Warning:** The **RECURSIVE** keyword is an experimental feature and is not intended for use in a production environment. Experimental features are subject to change without notice in future releases.

If **RECURSIVE** is specified, it allows a subquery to reference itself by name. Such a subquery, the **select** portion of the **with_query**, must have the form

```
non_recursive_term UNION [ALL] recursive_term
```

The recursive self-reference must appear on the right-hand side of the **UNION [ALL]**. Only one recursive self-reference is permitted per query.

If the **RECURSIVE** keyword is specified, the **WITH** queries need not be ordered: a query can reference another query that is later in the list. However, circular references, or mutual recursion, are not supported.

Without the **RECURSIVE** keyword, **WITH** queries can only reference sibling **WITH** queries that are earlier in the **WITH** list.

**WITH RECURSIVE** limitations. These items are not supported,
• A recursive WITH clause that contains the following in the recursive_term.
  • Subqueries with a self-reference
  • DISTINCT clause
  • GROUP BY clause
  • A window function
• A recursive WITH clause where the with_query_name is a part of a set operation.

An example the set operation limitation. This query returns an error because the set operation UNION contains a reference to the table foo.

```sql
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;
```

This recursive CTE is allowed because the set operation UNION does not have a reference to the CTE foo.

```sql
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;
```

See WITH Queries (Common Table Expressions) in the Greenplum Database Administrator Guide for additional information.

The SELECT List

The SELECT list (between the key words SELECT and FROM) specifies expressions that form the output rows of the SELECT statement. The expressions can (and usually do) refer to columns computed in the FROM clause.

Using the clause [AS] output_name, another name can be specified for an output column. This name is primarily used to label the column for display. It can also be used to refer to the column's value in ORDER BY and GROUP BY clauses, but not in the WHERE or HAVING clauses; there you must write out the expression instead. The AS keyword is optional in most cases (such as when declaring an alias for column names, constants, function calls, and simple unary operator expressions). In cases where the declared alias is a reserved SQL keyword, the output_name must be enclosed in double quotes to avoid ambiguity.

An expression in the SELECT list can be a constant value, a column reference, an operator invocation, a function call, an aggregate expression, a window expression, a scalar subquery, and so on. A number of constructs can be classified as an expression but do not follow any general syntax rules. These generally have the semantics of a function or operator. For information about SQL value expressions and function calls, see “Querying Data” in the Greenplum Database Administrator Guide.

Instead of an expression, * can be written in the output list as a shorthand for all the columns of the selected rows. Also, you can write table_name.* as a shorthand for the columns coming from just that table.

The FROM Clause

The FROM clause specifies one or more source tables for the SELECT. If multiple sources are specified, the result is the Cartesian product (cross join) of all the sources. But usually qualification conditions are added to restrict the returned rows to a small subset of the Cartesian product. The FROM clause can contain the following elements:
**table_name**
The name (optionally schema-qualified) of an existing table or view. If **ONLY** is specified, only that table is scanned. If **ONLY** is not specified, the table and all its descendant tables (if any) are scanned.

**alias**
A substitute name for the **FROM** item containing the alias. An alias is used for brevity or to eliminate ambiguity for self-joins (where the same table is scanned multiple times). When an alias is provided, it completely hides the actual name of the table or function; for example given **FROM foo AS f**, the remainder of the **SELECT** must refer to this **FROM** item as **f** not **foo**. If an alias is written, a column alias list can also be written to provide substitute names for one or more columns of the table.

**select**
A sub-**SELECT** can appear in the **FROM** clause. This acts as though its output were created as a temporary table for the duration of this single **SELECT** command. Note that the sub-**SELECT** must be surrounded by parentheses, and an alias must be provided for it. A **VALUES** command can also be used here. See "Non-standard Clauses" in the Compatibility section for limitations of using correlated sub-selects in Greenplum Database.

**with_query_name**
A **with_query** is referenced in the **FROM** clause by specifying its **with_query_name**, just as though the name were a table name. The **with_query_name** cannot contain a schema qualifier. An alias can be provided in the same way as for a table.

The **with_query** hides a table of the same name for the purposes of the primary query. If necessary, you can refer to a table of the same name by qualifying the table name with the schema.

**function_name**
Function calls can appear in the **FROM** clause. (This is especially useful for functions that return result sets, but any function can be used.) This acts as though its output were created as a temporary table for the duration of this single **SELECT** command. An alias may also be used. If an alias is written, a column alias list can also be written to provide substitute names for one or more attributes of the function's composite return type. If the function has been defined as returning the record data type, then an alias or the key word **AS** must be present, followed by a column definition list in the form (column_name data_type [, ... ] ). The column definition list must match the actual number and types of columns returned by the function.

**join_type**
One of:
- **[INNER] JOIN**
- **LEFT [OUTER] JOIN**
- **RIGHT [OUTER] JOIN**
- **FULL [OUTER] JOIN**
- **CROSS JOIN**

For the **INNER** and **OUTER** join types, a join condition must be specified, namely exactly one of **NATURAL**, **ON join_condition**, or **USING (join_column [, ...])**. See below for the meaning. For **CROSS JOIN**, none of these clauses may appear.

A **JOIN** clause combines two **FROM** items. Use parentheses if necessary to determine the order of nesting. In the absence of parentheses, **JOINs** nest left-to-right. In any case **JOIN** binds more tightly than the commas separating **FROM** items.

**CROSS JOIN** and **INNER JOIN** produce a simple Cartesian product, the same result as you get from listing the two items at the top level of **FROM**, but restricted by the join
condition (if any). CROSS JOIN is equivalent to INNER JOIN ON(TRUE), that is, no rows are removed by qualification. These join types are just a notational convenience, since they do nothing you could not do with plain FROM and WHERE.

LEFT OUTER JOIN returns all rows in the qualified Cartesian product (i.e., all combined rows that pass its join condition), plus one copy of each row in the left-hand table for which there was no right-hand row that passed the join condition. This left-hand row is extended to the full width of the joined table by inserting null values for the right-hand columns. Note that only the JOIN clause's own condition is considered while deciding which rows have matches. Outer conditions are applied afterwards.

Conversely, RIGHT OUTER JOIN returns all the joined rows, plus one row for each unmatched right-hand row (extended with nulls on the left). This is just a notational convenience, since you could convert it to a LEFT OUTER JOIN by switching the left and right inputs.

FULL OUTER JOIN returns all the joined rows, plus one row for each unmatched left-hand row (extended with nulls on the right), plus one row for each unmatched right-hand row (extended with nulls on the left).

ON join_condition
  join_condition is an expression resulting in a value of type boolean (similar to a WHERE clause) that specifies which rows in a join are considered to match.

USING (join_column [, ...])
  A clause of the form USING ( a, b, ... ) is shorthand for ON left_table.a = right_table.a AND left_table.b = right_table.b .... Also, USING implies that only one of each pair of equivalent columns will be included in the join output, not both.

NATURAL
  NATURAL is shorthand for a USING list that mentions all columns in the two tables that have the same names.

The WHERE Clause
The optional WHERE clause has the general form:

WHERE condition

where condition is any expression that evaluates to a result of type boolean. Any row that does not satisfy this condition will be eliminated from the output. A row satisfies the condition if it returns true when the actual row values are substituted for any variable references.

The GROUP BY Clause
The optional GROUP BY clause has the general form:

GROUP BY grouping_element [, ...]

where grouping_element can be one of:

() 
expression 
ROLLUP (expression [, ...])
CUBE (expression [, ...])
GROUPING SETS ((grouping_element [, ...]))

GROUP BY will condense into a single row all selected rows that share the same values for the grouped expressions. expression can be an input column name, or the name or ordinal number of an output column (SELECT list item), or an arbitrary expression formed from input-column values. In case of ambiguity, a GROUP BY name will be interpreted as an input-column name rather than an output column name.
Aggregate functions, if any are used, are computed across all rows making up each group, producing a separate value for each group (whereas without GROUP BY, an aggregate produces a single value computed across all the selected rows). When GROUP BY is present, it is not valid for the SELECT list expressions to refer to ungrouped columns except within aggregate functions, since there would be more than one possible value to return for an ungrouped column.

Greenplum Database has the following additional OLAP grouping extensions (often referred to as supergroups):

**ROLLUP**

A ROLLUP grouping is an extension to the GROUP BY clause that creates aggregate subtotals that roll up from the most detailed level to a grand total, following a list of grouping columns (or expressions). ROLLUP takes an ordered list of grouping columns, calculates the standard aggregate values specified in the GROUP BY clause, then creates progressively higher-level subtotals, moving from right to left through the list. Finally, it creates a grand total. A ROLLUP grouping can be thought of as a series of grouping sets. For example:

GROUP BY ROLLUP (a,b,c)

is equivalent to:

GROUP BY GROUPING SETS( (a,b,c), (a,b), (a), () )

Notice that the $n$ elements of a ROLLUP translate to $n+1$ grouping sets. Also, the order in which the grouping expressions are specified is significant in a ROLLUP.

**CUBE**

A CUBE grouping is an extension to the GROUP BY clause that creates subtotals for all of the possible combinations of the given list of grouping columns (or expressions). In terms of multidimensional analysis, CUBE generates all the subtotals that could be calculated for a data cube with the specified dimensions. For example:

GROUP BY CUBE (a,b,c)

is equivalent to:

GROUP BY GROUPING SETS( (a,b,c), (a,b), (a,c), (b,c), (a), (b), (c), () )

Notice that $n$ elements of a CUBE translate to $2n$ grouping sets. Consider using CUBE in any situation requiring cross-tabular reports. CUBE is typically most suitable in queries that use columns from multiple dimensions rather than columns representing different levels of a single dimension. For instance, a commonly requested cross-tabulation might need subtotals for all the combinations of month, state, and product.

**GROUPING SETS**

You can selectively specify the set of groups that you want to create using a GROUPING SETS expression within a GROUP BY clause. This allows precise specification across multiple dimensions without computing a whole ROLLUP or CUBE. For example:

GROUP BY GROUPING SETS( (a,c), (a,b) )

If using the grouping extension clauses ROLLUP, CUBE, or GROUPING SETS, two challenges arise. First, how do you determine which result rows are subtotals, and then the exact level of aggregation for a given subtotal. Or, how do you differentiate between result rows that contain both stored NULL values and "NULL" values created by the ROLLUP or CUBE. Secondly, when duplicate grouping sets are specified in the GROUP BY clause, how
do you determine which result rows are duplicates? There are two additional grouping functions you can use in the SELECT list to help with this:

- **grouping(column [, ...])** — The grouping function can be applied to one or more grouping attributes to distinguish super-aggregated rows from regular grouped rows. This can be helpful in distinguishing a "NULL" representing the set of all values in a super-aggregated row from a NULL value in a regular row. Each argument in this function produces a bit — either 1 or 0, where 1 means the result row is super-aggregated, and 0 means the result row is from a regular grouping. The grouping function returns an integer by treating these bits as a binary number and then converting it to a base-10 integer.

- **group_id()** — For grouping extension queries that contain duplicate grouping sets, the group_id function is used to identify duplicate rows in the output. All unique grouping set output rows will have a group_id value of 0. For each duplicate grouping set detected, the group_id function assigns a group_id number greater than 0. All output rows in a particular duplicate grouping set are identified by the same group_id number.

**The WINDOW Clause**

The WINDOW clause is used to define a window that can be used in the OVER() expression of a window function such as rank or avg. For example:

```sql
SELECT vendor, rank() OVER (mywindow) FROM sale
GROUP BY vendor
WINDOW mywindow AS (ORDER BY sum(prc*qty));
```

A WINDOW clause has this general form:

```sql
WINDOW window_name AS (window_specification)
```

where **window_specification** can be:

```sql
[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 ]]
```

where **window_frame_bound** can be one of:

- UNBOUNDED PRECEDING
- expression PRECEDING
- CURRENT ROW
- expression FOLLOWING
- UNBOUNDED FOLLOWING

**window_name**

Gives a name to the window specification.

**PARTITION BY**

The PARTITION BY clause organizes the result set into logical groups based on the unique values of the specified expression. When used with window functions, the functions are applied to each partition independently. For example, if you follow PARTITION BY with a column name, the result set is partitioned by the distinct values of that column. If omitted, the entire result set is considered one partition.

**ORDER BY**
The ORDER BY clause defines how to sort the rows in each partition of the result set. If omitted, rows are returned in whatever order is most efficient and may vary. **Note:** Columns of data types that lack a coherent ordering, such as `time`, are not good candidates for use in the ORDER BY clause of a window specification. Time, with or without time zone, lacks a coherent ordering because addition and subtraction do not have the expected effects. For example, the following is not generally true:  

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

**ROWS | RANGE**

Use either a **ROWS** or **RANGE** clause to express the bounds of the window. The window bound can be one, many, or all rows of a partition. You can express the bound of the window either in terms of a range of data values offset from the value in the current row (RANGE), or in terms of the number of rows offset from the current row (ROWS). When using the RANGE clause, you must also use an ORDER BY clause. This is because the calculation performed to produce the window requires that the values be sorted. Additionally, the ORDER BY clause cannot contain more than one expression, and the expression must result in either a date or a numeric value. When using the ROWS or RANGE clauses, if you specify only a starting row, the current row is used as the last row in the window.

**PRECEDING** — The PRECEDING clause defines the first row of the window using the current row as a reference point. The starting row is expressed in terms of the number of rows preceding the current row. For example, in the case of ROWS framing, 5 PRECEDING sets the window to start with the fifth row preceding the current row. In the case of RANGE framing, it sets the window to start with the first row whose ordering column value precedes that of the current row by 5 in the given order. If the specified order is ascending by date, this will be the first row within 5 days before the current row. UNBOUNDED PRECEDING sets the first row in the window to be the first row in the partition.

**BETWEEN** — The BETWEEN clause defines the first and last row of the window, using the current row as a reference point. First and last rows are expressed in terms of the number of rows preceding and following the current row, respectively. For example, BETWEEN 3 PRECEDING AND 5 FOLLOWING sets the window to start with the third row preceding the current row, and end with the fifth row following the current row. Use BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING to set the first and last rows in the window to be the first and last row in the partition, respectively. This is equivalent to the default behavior if no ROW or RANGE clause is specified.

**FOLLOWING** — The FOLLOWING clause defines the last row of the window using the current row as a reference point. The last row is expressed in terms of the number of rows following the current row. For example, in the case of ROWS framing, 5 FOLLOWING sets the window to end with the fifth row following the current row. In the case of RANGE framing, it sets the window to end with the last row whose ordering column value follows that of the current row by 5 in the given order. If the specified order is ascending by date, this will be the last row within 5 days after the current row. Use UNBOUNDED FOLLOWING to set the last row in the window to be the last row in the partition.

If you do not specify a ROW or a RANGE clause, the window bound starts with the first row in the partition (UNBOUNDED PRECEDING) and ends with the current row (CURRENT ROW) if ORDER BY is used. If an ORDER BY is not specified, the window starts with the first row in the partition (UNBOUNDED PRECEDING) and ends with last row in the partition (UNBOUNDED FOLLOWING).

**The HAVING Clause**

The optional **HAVING** clause has the general form:

```
HAVING condition
```

where condition is the same as specified for the **WHERE** clause. **HAVING** eliminates group rows that do not satisfy the condition. **HAVING** is different from **WHERE**: **WHERE** filters individual rows before the application
of GROUP BY, while HAVING filters group rows created by GROUP BY. Each column referenced in condition must unambiguously reference a grouping column, unless the reference appears within an aggregate function.

The presence of HAVING turns a query into a grouped query even if there is no GROUP BY clause. This is the same as what happens when the query contains aggregate functions but no GROUP BY clause. All the selected rows are considered to form a single group, and the SELECT list and HAVING clause can only reference table columns from within aggregate functions. Such a query will emit a single row if the HAVING condition is true, zero rows if it is not true.

The UNION Clause

The UNION clause has this general form:

```
select_statement UNION [ALL] select_statement
```

where `select_statement` is any SELECT statement without an ORDER BY, LIMIT, FOR UPDATE, or FOR SHARE clause. (ORDER BY and LIMIT can be attached to a subquery expression if it is enclosed in parentheses. Without parentheses, these clauses will be taken to apply to the result of the UNION, not to its right-hand input expression.)

The UNION operator computes the set union of the rows returned by the involved SELECT statements. A row is in the set union of two result sets if it appears in at least one of the result sets. The two SELECT statements that represent the direct operands of the UNION must produce the same number of columns, and corresponding columns must be of compatible data types.

The result of UNION does not contain any duplicate rows unless the ALL option is specified. ALL prevents elimination of duplicates. (Therefore, UNION ALL is usually significantly quicker than UNION; use ALL when you can.)

Multiple UNION operators in the same SELECT statement are evaluated left to right, unless otherwise indicated by parentheses.

Currently, FOR UPDATE and FOR SHARE may not be specified either for a UNION result or for any input of a UNION.

The INTERSECT Clause

The INTERSECT clause has this general form:

```
select_statement INTERSECT [ALL] select_statement
```

where `select_statement` is any SELECT statement without an ORDER BY, LIMIT, FOR UPDATE, or FOR SHARE clause.

The INTERSECT operator computes the set intersection of the rows returned by the involved SELECT statements. A row is in the intersection of two result sets if it appears in both result sets.

The result of INTERSECT does not contain any duplicate rows unless the ALL option is specified. With ALL, a row that has \( m \) duplicates in the left table and \( n \) duplicates in the right table will appear \( \min(m, n) \) times in the result set.

Multiple INTERSECT operators in the same SELECT statement are evaluated left to right, unless parentheses dictate otherwise. INTERSECT binds more tightly than UNION. That is, \( \text{A UNION B INTERSECT C} \) will be read as \( \text{A UNION (B INTERSECT C)} \).

Currently, FOR UPDATE and FOR SHARE may not be specified either for an INTERSECT result or for any input of an INTERSECT.

The EXCEPT Clause
The \texttt{EXCEPT} clause has this general form:

\begin{verbatim}
select_statement EXCEPT [\texttt{ALL}] select_statement
\end{verbatim}

where \texttt{select_statement} is any \texttt{SELECT} statement without an \texttt{ORDER BY}, \texttt{LIMIT}, \texttt{FOR UPDATE}, or \texttt{FOR SHARE} clause.

The \texttt{EXCEPT} operator computes the set of rows that are in the result of the left \texttt{SELECT} statement but not in the result of the right one.

The result of \texttt{EXCEPT} does not contain any duplicate rows unless the \texttt{ALL} option is specified. With \texttt{ALL}, a row that has \(m\) duplicates in the left table and \(n\) duplicates in the right table will appear \(\max(m-n,0)\) times in the result set.

Multiple \texttt{EXCEPT} operators in the same \texttt{SELECT} statement are evaluated left to right, unless parentheses dictate otherwise. \texttt{EXCEPT} binds at the same level as \texttt{UNION}.

Currently, \texttt{FOR UPDATE} and \texttt{FOR SHARE} may not be specified either for an \texttt{EXCEPT} result or for any input of an \texttt{EXCEPT}.

\textbf{The ORDER BY Clause}

The optional \texttt{ORDER BY} clause has this general form:

\begin{verbatim}
ORDER BY expression [\texttt{ASC} | \texttt{DESC} | \texttt{USING} operator] [\texttt{NULLS} { \texttt{FIRST} | \texttt{LAST}}] [, ...]
\end{verbatim}

where \texttt{expression} can be the name or ordinal number of an output column (\texttt{SELECT} list item), or it can be an arbitrary expression formed from input-column values.

The \texttt{ORDER BY} clause causes the result rows to be sorted according to the specified expressions. If two rows are equal according to the left-most expression, they are compared according to the next expression and so on. If they are equal according to all specified expressions, they are returned in an implementation-dependent order.

The ordinal number refers to the ordinal (left-to-right) position of the result column. This feature makes it possible to define an ordering on the basis of a column that does not have a unique name. This is never absolutely necessary because it is always possible to assign a name to a result column using the \texttt{AS} clause.

It is also possible to use arbitrary expressions in the \texttt{ORDER BY} clause, including columns that do not appear in the \texttt{SELECT} result list. Thus the following statement is valid:

\begin{verbatim}
SELECT name FROM distributors ORDER BY code;
\end{verbatim}

A limitation of this feature is that an \texttt{ORDER BY} clause applying to the result of a \texttt{UNION}, \texttt{INTERSECT}, or \texttt{EXCEPT} clause may only specify an output column name or number, not an expression.

If an \texttt{ORDER BY} expression is a simple name that matches both a result column name and an input column name, \texttt{ORDER BY} will interpret it as the result column name. This is the opposite of the choice that \texttt{GROUP BY} will make in the same situation. This inconsistency is made to be compatible with the SQL standard.

Optionally one may add the key word \texttt{ASC} (ascending) or \texttt{DESC} (descending) after any expression in the \texttt{ORDER BY} clause. If not specified, \texttt{ASC} is assumed by default. Alternatively, a specific ordering operator name may be specified in the \texttt{USING} clause. \texttt{ASC} is usually equivalent to \texttt{USING <} and \texttt{DESC} is usually equivalent to \texttt{USING >}. (But the creator of a user-defined data type can define exactly what the default sort ordering is, and it might correspond to operators with other names.)

If \texttt{NULLS LAST} is specified, null values sort after all non-null values; if \texttt{NULLS FIRST} is specified, null values sort before all non-null values. If neither is specified, the default behavior is \texttt{NULLS LAST} when \texttt{ASC} is specified or implied, and \texttt{NULLS FIRST} when \texttt{DESC} is specified (thus, the default is to act as though nulls are larger than non-nulls). When \texttt{USING} is specified, the default nulls ordering depends upon whether the operator is a less-than or greater-than operator.
Note that ordering options apply only to the expression they follow; for example `ORDER BY x, y DESC` does not mean the same thing as `ORDER BY x DESC, y DESC`.

Character-string data is sorted according to the locale-specific collation order that was established when the Greenplum Database system was initialized.

### The DISTINCT Clause

If `DISTINCT` is specified, all duplicate rows are removed from the result set (one row is kept from each group of duplicates). `ALL` specifies the opposite: all rows are kept. `ALL` is the default.

`DISTINCT ON ( expression [, ...] )` keeps only the first row of each set of rows where the given expressions evaluate to equal. The `DISTINCT ON` expressions are interpreted using the same rules as for `ORDER BY`. Note that the 'first row' of each set is unpredictable unless `ORDER BY` is used to ensure that the desired row appears first. For example:

```sql
SELECT DISTINCT ON (location) location, time, report FROM weather_reports ORDER BY location, time DESC;
```

retrieves the most recent weather report for each location. But if we had not used `ORDER BY` to force descending order of time values for each location, we would have gotten a report from an unpredictable time for each location.

The `DISTINCT ON` expression(s) must match the left-most `ORDER BY` expression(s). The `ORDER BY` clause will normally contain additional expression(s) that determine the desired precedence of rows within each `DISTINCT ON` group.

When Greenplum Database processes queries that contain the `DISTINCT` clause, the queries are transformed into `GROUP BY` queries. In many cases, the transformation provides significant performance gains. However, when the number of distinct values is close to the total number of rows, the transformation might result in the generation of a multi-level grouping plan. In this case, there is an expected performance degradation because of the overhead introduced by the lower aggregation level.

### The LIMIT Clause

The `LIMIT` clause consists of two independent sub-clauses:

```sql
LIMIT {count | ALL}
OFFSET start
```

where `count` specifies the maximum number of rows to return, while `start` specifies the number of rows to skip before starting to return rows. When both are specified, start rows are skipped before starting to count the count rows to be returned.

When using `LIMIT`, it is a good idea to use an `ORDER BY` clause that constrains the result rows into a unique order. Otherwise you will get an unpredictable subset of the query’s rows — you may be asking for the tenth through twentieth rows, but tenth through twentieth in what ordering? You don't know what ordering unless you specify `ORDER BY`.

The query optimizer takes `LIMIT` into account when generating a query plan, so you are very likely to get different plans (yielding different row orders) depending on what you use for `LIMIT` and `OFFSET`. Thus, using different `LIMIT/OFFSET` values to select different subsets of a query result will give inconsistent results unless you enforce a predictable result ordering with `ORDER BY`. This is not a defect; it is an inherent consequence of the fact that SQL does not promise to deliver the results of a query in any particular order unless `ORDER BY` is used to constrain the order.

### The FOR UPDATE/FOR SHARE Clause

The `FOR UPDATE` clause has this form:

```sql
FOR UPDATE [OF table_name [, ...]] [NOWAIT]
```
The closely related `FOR SHARE` clause has this form:

```
FOR SHARE [OF table_name [, ...]] [NOWAIT]
```

`FOR UPDATE` causes the tables accessed by the `SELECT` statement to be locked as though for update. This prevents the table from being modified or deleted by other transactions until the current transaction ends. That is, other transactions that attempt `UPDATE`, `DELETE`, or `SELECT FOR UPDATE` of this table will be blocked until the current transaction ends. Also, if an `UPDATE`, `DELETE`, or `SELECT FOR UPDATE` from another transaction has already locked a selected table, `SELECT FOR UPDATE` will wait for the other transaction to complete, and will then lock and return the updated table.

To prevent the operation from waiting for other transactions to commit, use the `NOWAIT` option. `SELECT FOR UPDATE NOWAIT` reports an error, rather than waiting, if a selected row cannot be locked immediately. Note that `NOWAIT` applies only to the row-level lock(s) — the required `ROW SHARE` table-level lock is still taken in the ordinary way. You can use the `NOWAIT` option of `LOCK` if you need to acquire the table-level lock without waiting (see `LOCK`).

`FOR SHARE` behaves similarly, except that it acquires a shared rather than exclusive lock on the table. A shared lock blocks other transactions from performing `UPDATE`, `DELETE`, or `SELECT FOR UPDATE` on the table, but it does not prevent them from performing `SELECT FOR SHARE`.

If specific tables are named in `FOR UPDATE` or `FOR SHARE`, then only those tables are locked; any other tables used in the `SELECT` are simply read as usual. A `FOR UPDATE` or `FOR SHARE` clause without a table list affects all tables used in the command. If `FOR UPDATE` or `FOR SHARE` is applied to a view or subquery, it affects all tables used in the view or subquery.

`FOR UPDATE` or `FOR SHARE` do not apply to a `with_query` referenced by the primary query. If you want row locking to occur within a `with_query`, specify `FOR UPDATE` or `FOR SHARE` within the `with_query`.

Multiple `FOR UPDATE` and `FOR SHARE` clauses can be written if it is necessary to specify different locking behavior for different tables. If the same table is mentioned (or implicitly affected) by both `FOR UPDATE` and `FOR SHARE` clauses, then it is processed as `FOR UPDATE`. Similarly, a table is processed as `NOWAIT` if that is specified in any of the clauses affecting it.

### Examples

To join the table `films` with the table `distributors`:

```
SELECT f.title, f.did, d.name, f.date_prod, f.kind FROM distributors d, films f WHERE f.did = d.did
```

To sum the column `length` of all films and group the results by `kind`:

```
SELECT kind, sum(length) AS total FROM films GROUP BY kind;
```

To sum the column `length` of all films, group the results by `kind` and show those group totals that are less than 5 hours:

```
SELECT kind, sum(length) AS total FROM films GROUP BY kind
HAVING sum(length) < interval '5 hours';
```

Calculate the subtotals and grand totals of all sales for movie `kind` and `distributor`.

```
SELECT kind, distributor, sum(prc*qty) FROM sales
GROUP BY ROLLUP(kind, distributor)
ORDER BY 1,2,3;
```

Calculate the rank of movie distributors based on total sales:

```
SELECT distributor, sum(prc*qty),
```
rank() OVER (ORDER BY sum(prc*qty) DESC)
FROM sale
GROUP BY distributor ORDER BY 2 DESC;

The following two examples are identical ways of sorting the individual results according to the contents of
the second column (name):

SELECT * FROM distributors ORDER BY name;
SELECT * FROM distributors ORDER BY 2;

The next example shows how to obtain the union of the tables distributors and actors, restricting the
results to those that begin with the letter W in each table. Only distinct rows are wanted, so the key word
ALL is omitted:

SELECT distributors.name FROM distributors WHERE
distributors.name LIKE 'W%' UNION SELECT actors.name FROM
actors WHERE actors.name LIKE 'W';

This example shows how to use a function in the FROM clause, both with and without a column definition
list:

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);

This example uses a simple WITH clause:

WITH test AS (
    SELECT random() as x FROM generate_series(1, 3)
  )
SELECT * FROM test
UNION ALL
SELECT * FROM test;

This example uses the WITH clause to display per-product sales totals in only the top sales regions.

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;

The example could have been written without the WITH clause but would have required two levels of
nested sub-SELECT statements.
This example uses the `WITH RECURSIVE` clause to find all subordinates (direct or indirect) of the employee Mary, and their level of indirectness, from a table that shows only direct subordinates:

```sql
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;
```

The typical form of recursive queries: an initial condition, followed by `UNION [ALL]`, followed by the recursive part of the query. Be sure that the recursive part of the query will eventually return no tuples, or else the query will loop indefinitely. See `WITH Queries (Common Table Expressions)` in the `Greenplum Database Administrator Guide` for more examples.

**Compatibility**

The `SELECT` statement is compatible with the SQL standard, but there are some extensions and some missing features.

**Omitted FROM Clauses**

Greenplum Database allows one to omit the `FROM` clause. It has a straightforward use to compute the results of simple expressions. For example:

```sql
SELECT 2+2;
```

Some other SQL databases cannot do this except by introducing a dummy one-row table from which to do the `SELECT`.

Note that if a `FROM` clause is not specified, the query cannot reference any database tables. For compatibility with applications that rely on this behavior the `add_missing_from` configuration variable can be enabled.

**The AS Key Word**

In the SQL standard, the optional key word `AS` is just noise and can be omitted without affecting the meaning. The Greenplum Database parser requires this key word when renaming output columns because the type extensibility features lead to parsing ambiguities without it. `AS` is optional in `FROM` items, however.

**Namespace Available to GROUP BY and ORDER BY**

In the SQL-92 standard, an `ORDER BY` clause may only use result column names or numbers, while a `GROUP BY` clause may only use expressions based on input column names. Greenplum Database extends each of these clauses to allow the other choice as well (but it uses the standard's interpretation if there is ambiguity). Greenplum Database also allows both clauses to specify arbitrary expressions. Note that names appearing in an expression are always taken as input-column names, not as result-column names.

SQL:1999 and later use a slightly different definition which is not entirely upward compatible with SQL-92. In most cases, however, Greenplum Database interprets an `ORDER BY` or `GROUP BY` expression the same way SQL:1999 does.

**Nonstandard Clauses**

The clauses `DISTINCT ON`, `LIMIT`, and `OFFSET` are not defined in the SQL standard.

**Limited Use of STABLE and VOLATILE Functions**
To prevent data from becoming out-of-sync across the segments in Greenplum Database, any function classified as STABLE or VOLATILE cannot be executed at the segment database level if it contains SQL or modifies the database in any way. See CREATE FUNCTION for more information.

See Also

EXPLAIN

SELECT INTO

Defines a new table from the results of a query.

Synopsis

```sql
[ WITH [ RECURSIVE¹ ] with_query [, ...] ]
SELECT [ALL | DISTINCT [ON ( expression [, ...] )]]
    * | expression [AS output_name] [, ...]
INTO [TEMPORARY | TEMP] [TABLE] new_table
[FROM from_item [, ...]]
[WHERE condition]
[GROUP BY expression [, ...]]
[HAVING condition [, ...]]
[(UNION | INTERSECT | EXCEPT) [ALL] select]
[ORDER BY expression [ASC | DESC | USING operator] [NULLS {FIRST | LAST}] [, ...]]
[LIMIT {count | ALL}]
[OFFSET start]
[FOR {UPDATE | SHARE} [OF table_name [, ...]] [NOWAIT]
[...]]
```

Note: ¹The RECURSIVE keyword is an experimental feature and is not recommended for production use.

Description

SELECT INTO creates a new table and fills it with data computed by a query. The data is not returned to the client, as it is with a normal SELECT. The new table’s columns have the names and data types associated with the output columns of the SELECT.

The RECURSIVE keyword can be enabled by setting the server configuration parameter `gp_recursive_cte_prototype` to true.

Warning: The RECURSIVE keyword is an experimental feature and is not intended for use in a production environment. Experimental features are subject to change without notice in future releases.

Parameters

The majority of parameters for SELECT INTO are the same as SELECT.

TEMPORARY

TEMP

If specified, the table is created as a temporary table.

new_table

The name (optionally schema-qualified) of the table to be created.
Examples
Create a new table `films_recent` consisting of only recent entries from the table `films`:

```
SELECT * INTO films_recent FROM films WHERE date_prod >= '2016-01-01';
```

Compatibility
The SQL standard uses `SELECT INTO` to represent selecting values into scalar variables of a host program, rather than creating a new table. The Greenplum Database usage of `SELECT INTO` to represent table creation is historical. It is best to use `CREATE TABLE AS` for this purpose in new applications.

See Also
`SELECT`, `CREATE TABLE AS`

**SET**
Changes the value of a Greenplum Database configuration parameter.

**Synopsis**

```
SET [SESSION | LOCAL] configuration_parameter {TO | =} value | 'value' | DEFAULT
```

```
SET [SESSION | LOCAL] TIME ZONE {timezone | LOCAL | DEFAULT}
```

**Description**
The `SET` command changes server configuration parameters. Any configuration parameter classified as a `session` parameter can be changed on-the-fly with `SET`. `SET` affects only the value used by the current session.

If `SET` or `SET SESSION` is issued within a transaction that is later aborted, the effects of the `SET` command disappear when the transaction is rolled back. Once the surrounding transaction is committed, the effects will persist until the end of the session, unless overridden by another `SET`.

The effects of `SET LOCAL` last only till the end of the current transaction, whether committed or not. A special case is `SET` followed by `SET LOCAL` within a single transaction: the `SET` LOCAL value will be seen until the end of the transaction, but afterwards (if the transaction is committed) the `SET` value will take effect.

If `SET LOCAL` is used within a function that includes a `SET` option for the same configuration parameter (see `CREATE FUNCTION`), the effects of the `SET LOCAL` command disappear at function exit; the value in effect when the function was called is restored anyway. This allows `SET LOCAL` to be used for dynamic or repeated changes of a parameter within a function, while retaining the convenience of using the `SET` option to save and restore the caller's value. Note that a regular `SET` command overrides any surrounding function's `SET` option; its effects persist unless rolled back.

If you create a cursor with the `DECLARE` command in a transaction, you cannot use the `SET` command in the transaction until you close the cursor with the `CLOSE` command.

See `Server Configuration Parameters` for information about server parameters.

**Parameters**

**SESSION**

Specifies that the command takes effect for the current session. This is the default.
LOCAL

Specifies that the command takes effect for only the current transaction. After COMMIT or ROLLBACK, the session-level setting takes effect again. Note that SET LOCAL will appear to have no effect if it is executed outside of a transaction.

classification parameter

The name of a Greenplum Database configuration parameter. Only parameters classified as session can be changed with SET. See Server Configuration Parameters for details.

value

New value of parameter. Values can be specified as string constants, identifiers, numbers, or comma-separated lists of these. DEFAULT can be used to specify resetting the parameter to its default value. If specifying memory sizing or time units, enclose the value in single quotes.

TIME ZONE

SET TIME ZONE value is an alias for SET timezone TO value. The syntax SET TIME ZONE allows special syntax for the time zone specification. Here are examples of valid values:

'PST8PDT'
'Europe/Rome'
-7 (time zone 7 hours west from UTC)
INTERVAL '-08:00' HOUR TO MINUTE (time zone 8 hours west from UTC).

LOCAL

DEFAULT

Set the time zone to your local time zone (the one that the server's operating system defaults to). See the Time zone section of the PostgreSQL documentation for more information about time zones in Greenplum Database.

Examples

Set the schema search path:

```
SET search_path TO my_schema, public;
```

Increase the segment host memory per query to 200 MB:

```
SET statement_mem TO '200MB';
```

Set the style of date to traditional POSTGRES with "day before month" input convention:

```
SET datestyle TO postgres, dmy;
```

Set the time zone for San Mateo, California (Pacific Time):

```
SET TIME ZONE 'PST8PDT';
```

Set the time zone for Italy:

```
SET TIME ZONE 'Europe/Rome';
```
Compatibility

_SET TIME ZONE_ extends syntax defined in the SQL standard. The standard allows only numeric time zone offsets while Greenplum Database allows more flexible time-zone specifications. All other _SET_ features are Greenplum Database extensions.

See Also

_RESET_, _SHOW_

**SET ROLE**

Sets the current role identifier of the current session.

**Synopsis**

<table>
<thead>
<tr>
<th>Command</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>SET</em> [SESSION</td>
<td>LOCAL] ROLE <em>rolename</em></td>
</tr>
<tr>
<td><em>SET</em> [SESSION</td>
<td>LOCAL] ROLE NONE</td>
</tr>
<tr>
<td><em>RESET</em> ROLE</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

This command sets the current role identifier of the current SQL-session context to be _rolename_. The role name may be written as either an identifier or a string literal. After _SET ROLE_, permissions checking for SQL commands is carried out as though the named role were the one that had logged in originally.

The specified _rolename_ must be a role that the current session user is a member of. If the session user is a superuser, any role can be selected.

The _NONE_ and _RESET_ forms reset the current role identifier to be the current session role identifier. These forms may be executed by any user.

**Parameters**

**SESSION**

Specifies that the command takes effect for the current session. This is the default.

**LOCAL**

Specifies that the command takes effect for only the current transaction. After _COMMIT_ or _ROLLBACK_, the session-level setting takes effect again. Note that _SET LOCAL_ will appear to have no effect if it is executed outside of a transaction.

**rolename**

The name of a role to use for permissions checking in this session.

**NONE**

Reset the current role identifier to be the current session role identifier (that of the role used to log in).

**Notes**

Using this command, it is possible to either add privileges or restrict privileges. If the session user role has the _INHERITS_ attribute, then it automatically has all the privileges of every role that it could _SET ROLE_ to; in this case _SET ROLE_ effectively drops all the privileges assigned directly to the session user and to the other roles it is a member of, leaving only the privileges available to the named role. On the other hand, if the session user role has the _NOINHERITS_ attribute, _SET ROLE_ drops the privileges assigned directly to the session user and instead acquires the privileges available to the named role.
In particular, when a superuser chooses to \texttt{SET ROLE} to a non-superuser role, she loses her superuser privileges.

\texttt{SET ROLE} has effects comparable to \texttt{SET SESSION AUTHORIZATION}, but the privilege checks involved are quite different. Also, \texttt{SET SESSION AUTHORIZATION} determines which roles are allowable for later \texttt{SET ROLE} commands, whereas changing roles with \texttt{SET ROLE} does not change the set of roles allowed to a later \texttt{SET ROLE}.

**Examples**

```
SELECT SESSION_USER, CURRENT_USER;
```

<table>
<thead>
<tr>
<th>session_user</th>
<th>current_user</th>
</tr>
</thead>
<tbody>
<tr>
<td>peter</td>
<td>peter</td>
</tr>
</tbody>
</table>

```
SET ROLE 'paul';
```

```
SELECT SESSION_USER, CURRENT_USER;
```

<table>
<thead>
<tr>
<th>session_user</th>
<th>current_user</th>
</tr>
</thead>
<tbody>
<tr>
<td>peter</td>
<td>paul</td>
</tr>
</tbody>
</table>

**Compatibility**

Greenplum Database allows identifier syntax (\textit{rolename}), while the SQL standard requires the role name to be written as a string literal. SQL does not allow this command during a transaction; Greenplum Database does not make this restriction. The \texttt{SESSION} and \texttt{LOCAL} modifiers are a Greenplum Database extension, as is the \texttt{RESET} syntax.

**See Also**

\texttt{SET SESSION AUTHORIZATION}

**\texttt{SET SESSION AUTHORIZATION}**

Sets the session role identifier and the current role identifier of the current session.

**Synopsis**

```
SET [SESSION | LOCAL] SESSION AUTHORIZATION rolename
```

```
SET [SESSION | LOCAL] SESSION AUTHORIZATION DEFAULT
```

```
RESET SESSION AUTHORIZATION
```

**Description**

This command sets the session role identifier and the current role identifier of the current SQL-session context to be \textit{rolename}. The role name may be written as either an identifier or a string literal. Using this command, it is possible, for example, to temporarily become an unprivileged user and later switch back to being a superuser.

The session role identifier is initially set to be the (possibly authenticated) role name provided by the client. The current role identifier is normally equal to the session user identifier, but may change temporarily in the context of \texttt{setuid} functions and similar mechanisms; it can also be changed by \texttt{SET ROLE}. The current user identifier is relevant for permission checking.

The session user identifier may be changed only if the initial session user (the authenticated user) had the superuser privilege. Otherwise, the command is accepted only if it specifies the authenticated user name.
The `DEFAULT` and `RESET` forms reset the session and current user identifiers to be the originally authenticated user name. These forms may be executed by any user.

**Parameters**

**SESSION**

Specifies that the command takes effect for the current session. This is the default.

**LOCAL**

Specifies that the command takes effect for only the current transaction. After `COMMIT` or `ROLLBACK`, the session-level setting takes effect again. Note that `SET LOCAL` will appear to have no effect if it is executed outside of a transaction.

**rolename**

The name of the role to assume.

**NONE**

Reset the session and current role identifiers to be that of the role used to log in.

**Examples**

```
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          |
```

**Compatibility**

The SQL standard allows some other expressions to appear in place of the literal `rolename`, but these options are not important in practice. Greenplum Database allows identifier syntax (`rolename`), which SQL does not. SQL does not allow this command during a transaction; Greenplum Database does not make this restriction. The `SESSION` and `LOCAL` modifiers are a Greenplum Database extension, as is the `RESET` syntax.

**See Also**

`SET ROLE`

**SET TRANSACTION**

Sets the characteristics of the current transaction.

**Synopsis**

```
SET TRANSACTION [transaction_mode] [READ ONLY | READ WRITE]
SET SESSION CHARACTERISTICS AS TRANSACTION transaction_mode
[READ ONLY | READ WRITE]
```
Description

The `SET TRANSACTION` command sets the characteristics of the current transaction. It has no effect on any subsequent transactions.

The available transaction characteristics are the transaction isolation level and the transaction access mode (read/write or read-only).

The isolation level of a transaction determines what data the transaction can see when other transactions are running concurrently.

- **READ COMMITTED** — A statement can only see rows committed before it began. This is the default.
- **SERIALIZABLE** — All statements of the current transaction can only see rows committed before the first query or data-modification statement was executed in this transaction.

The SQL standard defines two additional levels, **READ UNCOMMITTED** and **REPEATABLE READ**. In Greenplum Database **READ UNCOMMITTED** is treated as **READ COMMITTED**. **REPEATABLE READ** is not supported; use **SERIALIZABLE** if **REPEATABLE READ** behavior is required.

The transaction isolation level cannot be changed after the first query or data-modification statement (SELECT, INSERT, DELETE, UPDATE, FETCH, or COPY) of a transaction has been executed.

The transaction access mode determines whether the transaction is read/write or read-only. Read/write is the default. When a transaction is read-only, the following SQL commands are disallowed: INSERT, UPDATE, DELETE, and COPY FROM if the table they would write to is not a temporary table; all CREATE, ALTER, and DROP commands; GRANT, REVOKE, TRUNCATE; and EXPLAIN ANALYZE and EXECUTE if the command they would execute is among those listed. This is a high-level notion of read-only that does not prevent all writes to disk.

Parameters

**SESSION CHARACTERISTICS**

Sets the default transaction characteristics for subsequent transactions of a session.

- **SERIALIZABLE**
- **READ COMMITTED**
- **READ UNCOMMITTED**

The SQL standard defines four transaction isolation levels: **READ COMMITTED**, **READ UNCOMMITTED**, **SERIALIZABLE**, and **REPEATABLE READ**. The default behavior is that a statement can only see rows committed before it began (**READ COMMITTED**). In Greenplum Database **READ UNCOMMITTED** is treated the same as **READ COMMITTED**. **REPEATABLE READ** is not supported; use **SERIALIZABLE** instead. **SERIALIZABLE** is the strictest transaction isolation. This level emulates serial transaction execution, as if transactions had been executed one after another, serially, rather than concurrently. Applications using this level must be prepared to retry transactions due to serialization failures.

- **READ WRITE**
- **READ ONLY**

Determines whether the transaction is read/write or read-only. Read/write is the default. When a transaction is read-only, the following SQL commands are disallowed: INSERT, UPDATE, DELETE, and COPY FROM if the table they would write to is not a temporary table; all CREATE, ALTER, and DROP commands; GRANT, REVOKE, TRUNCATE; and EXPLAIN ANALYZE and EXECUTE if the command they would execute is among those listed.
Notes

If SET TRANSACTION is executed without a prior START TRANSACTION or BEGIN, it will appear to have no effect.

It is possible to dispense with SET TRANSACTION by instead specifying the desired transaction_modes in BEGIN or START TRANSACTION.

The session default transaction modes can also be set by setting the configuration parameters default_transaction_isolation and default_transaction_read_only.

Examples

Set the transaction isolation level for the current transaction:

```
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Compatibility

Both commands are defined in the SQL standard. SERIALIZABLE is the default transaction isolation level in the standard. In Greenplum Database the default is READ COMMITTED. Because of lack of predicate locking, the SERIALIZABLE level is not truly serializable. Essentially, a predicate-locking system prevents phantom reads by restricting what is written, whereas a multi-version concurrency control model (MVCC) as used in Greenplum Database prevents them by restricting what is read.

In the SQL standard, there is one other transaction characteristic that can be set with these commands: the size of the diagnostics area. This concept is specific to embedded SQL, and therefore is not implemented in the Greenplum Database server.

The SQL standard requires commas between successive transaction_modes, but for historical reasons Greenplum Database allows the commas to be omitted.

See Also

BEGIN, LOCK

SHOW

Shows the value of a system configuration parameter.

Synopsis

```
SHOW configuration_parameter
SHOW ALL
```

Description

SHOW displays the current settings of Greenplum Database system configuration parameters. You can set these parameters with the SET statement, or by editing the postgresql.conf configuration file of the Greenplum Database master. Note that some parameters viewable by SHOW are read-only — their values can be viewed but not set. See the Greenplum Database Reference Guide for details.

Parameters

```
configuration_parameter
ALL
```

The name of a system configuration parameter.
Shows the current value of all configuration parameters.

## Examples

Show the current setting of the parameter `search_path`:

```sql
SHOW search_path;
```

Show the current setting of all parameters:

```sql
SHOW ALL;
```

## Compatibility

`SHOW` is a Greenplum Database extension.

## See Also

`SET, RESET`

### START TRANSACTION

Starts a transaction block.

## Synopsis

```
START TRANSACTION [SERIALIZABLE | READ COMMITTED | READ UNCOMMITTED]
[READ WRITE | READ ONLY]
```

## Description

`START TRANSACTION` begins a new transaction block. If the isolation level or read/write mode is specified, the new transaction has those characteristics, as if `SET TRANSACTION` was executed. This is the same as the `BEGIN` command.

### Parameters

- **SERIALIZABLE**
- **READ COMMITTED**
- **READ UNCOMMITTED**

The SQL standard defines four transaction isolation levels: `READ COMMITTED`, `READ UNCOMMITTED`, `SERIALIZABLE`, and `REPEATABLE READ`. The default behavior is that a statement can only see rows committed before it began (`READ COMMITTED`). In Greenplum Database, `READ UNCOMMITTED` is treated the same as `READ COMMITTED`. `REPEATABLE READ` is not supported; use `SERIALIZABLE` if this behavior is required. `SERIALIZABLE`, wherein all statements of the current transaction can only see rows committed before the first statement was executed in the transaction, is the strictest transaction isolation. This level emulates serial transaction execution, as if transactions had been executed one after another, serially, rather than concurrently. Applications using this level must be prepared to retry transactions due to serialization failures.

- **READ WRITE**
- **READ ONLY**

Determines whether the transaction is read/write or read-only. Read/write is the default. When a transaction is read-only, the following SQL commands are disallowed: `INSERT, UPDATE, DELETE, and COPY FROM` if the table they would write to is not a temporary table;
all CREATE, ALTER, and DROP commands; GRANT, REVOKE, TRUNCATE; and EXPLAIN ANALYZE and EXECUTE if the command they would execute is among those listed.

Examples

To begin a transaction block:

```
START TRANSACTION;
```

Compatibility

In the standard, it is not necessary to issue `START TRANSACTION` to start a transaction block: any SQL command implicitly begins a block. Greenplum Database behavior can be seen as implicitly issuing a `COMMIT` after each command that does not follow `START TRANSACTION` (or `BEGIN`), and it is therefore often called 'autocommit'. Other relational database systems may offer an autocommit feature as a convenience.

The SQL standard requires commas between successive `transaction_modes`, but for historical reasons Greenplum Database allows the commas to be omitted.

See also the compatibility section of `SET TRANSACTION`.

See Also

`BEGIN`, `SET TRANSACTION`

TRUNCATE

Empties a table of all rows.

Synopsis

```
TRUNCATE [TABLE] name [, ...] [CASCADE | RESTRICT]
```

Description

`TRUNCATE` quickly removes all rows from a table or set of tables. It has the same effect as an unqualified `DELETE` on each table, but since it does not actually scan the tables it is faster. This is most useful on large tables.

You must have the TRUNCATE privilege on the table to truncate table rows.

Parameters

- `name`
  - The name (optionally schema-qualified) of a table to be truncated.

- `CASCADE`
  - Since this key word applies to foreign key references (which are not supported in Greenplum Database) it has no effect.

- `RESTRICT`
  - Since this key word applies to foreign key references (which are not supported in Greenplum Database) it has no effect.

Notes

`TRUNCATE` will not run any user-defined `ON DELETE` triggers that might exist for the tables.
**TRUNCATE** will not truncate any tables that inherit from the named table. Only the named table is truncated, not its child tables.

**TRUNCATE** will not truncate any sub-tables of a partitioned table. If you specify a sub-table of a partitioned table, **TRUNCATE** will not remove rows from the sub-table and its child tables.

**Examples**

Empty the table **films**:

```
TRUNCATE films;
```

**Compatibility**

There is no **TRUNCATE** command in the SQL standard.

**See Also**

DELETE, DROP TABLE

### UPDATE

Updates rows of a table.

**Synopsis**

```
UPDATE [ONLY] table [[AS] alias]
SET {column = {expression | DEFAULT} |
(column [, ...]) = ({expression | DEFAULT} [, ...])} [, ...]
[FROM fromlist]
[WHERE condition | WHERE CURRENT OF cursor_name ]
```

**Description**

**UPDATE** changes the values of the specified columns in all rows that satisfy the condition. Only the columns to be modified need be mentioned in the **SET** clause; columns not explicitly modified retain their previous values.

By default, **UPDATE** will update rows in the specified table and all its subtables. If you wish to only update the specific table mentioned, you must use the **ONLY** clause.

There are two ways to modify a table using information contained in other tables in the database: using sub-selects, or specifying additional tables in the **FROM** clause. Which technique is more appropriate depends on the specific circumstances.

If the **WHERE CURRENT OF** clause is specified, the row that is updated is the one most recently fetched from the specified cursor.

You must have the **UPDATE** privilege on the table to update it, as well as the **SELECT** privilege to any table whose values are read in the expressions or condition.

**Outputs**

On successful completion, an **UPDATE** command returns a command tag of the form:

```
UPDATE count
```

where **count** is the number of rows updated. If **count** is 0, no rows matched the condition (this is not considered an error).
Parameters

**ONLY**
If specified, update rows from the named table only. When not specified, any tables inheriting from the named table are also processed.

**table**
The name (optionally schema-qualified) of an existing table.

**alias**
A substitute name for the target table. When an alias is provided, it completely hides the actual name of the table. For example, given `UPDATE foo AS f`, the remainder of the `UPDATE` statement must refer to this table as `f` not `foo`.

**column**
The name of a column in table. The column name can be qualified with a subfield name or array subscript, if needed. Do not include the table's name in the specification of a target column.

**expression**
An expression to assign to the column. The expression may use the old values of this and other columns in the table.

**DEFAULT**
Set the column to its default value (which will be NULL if no specific default expression has been assigned to it).

**fromlist**
A list of table expressions, allowing columns from other tables to appear in the `WHERE` condition and the update expressions. This is similar to the list of tables that can be specified in the `FROM` clause of a `SELECT` statement. Note that the target table must not appear in the `fromlist`, unless you intend a self-join (in which case it must appear with an `alias` in the `fromlist`).

**condition**
An expression that returns a value of type boolean. Only rows for which this expression returns true will be updated.

**cursor_name**
The name of the cursor to use in a `WHERE CURRENT OF` condition. The row to be updated is the one most recently fetched from the cursor. The cursor must be a simple (non-join, non-aggregate) query on the `UPDATE` command target table. See `DECLARE` for more information about creating cursors.

WHERE CURRENT OF cannot be specified together with a Boolean condition.

The `UPDATE...WHERE CURRENT OF` statement can only be executed on the server, for example in an interactive `psql` session or a script. Language extensions such as PL/pgSQL do not have support for updatable cursors.

See `DECLARE` for more information about creating cursors.

**output_expression**
An expression to be computed and returned by the `UPDATE` command after each row is updated. The expression may use any column names of the table or table(s) listed in `FROM`. Write `*` to return all columns.

**output_name**
A name to use for a returned column.
Notes

**SET** is not allowed on the Greenplum distribution key columns of a table.

When a **FROM** clause is present, what essentially happens is that the target table is joined to the tables mentioned in the from list, and each output row of the join represents an update operation for the target table. When using **FROM** you should ensure that the join produces at most one output row for each row to be modified. In other words, a target row should not join to more than one row from the other table(s). If it does, then only one of the join rows will be used to update the target row, but which one will be used is not readily predictable.

Because of this indeterminacy, referencing other tables only within sub-selects is safer, though often harder to read and slower than using a join.

Executing **UPDATE** and **DELETE** commands directly on a specific partition (child table) of a partitioned table is not supported. Instead, execute these commands on the root partitioned table, the table created with the **CREATE TABLE** command.

Examples

Change the word **Drama** to **Dramatic** in the column **kind** of the table **films**:

```sql
UPDATE films SET kind = 'Dramatic' WHERE kind = 'Drama';
```

Adjust temperature entries and reset precipitation to its default value in one row of the table **weather**:

```sql
UPDATE weather SET temp_lo = temp_lo+1, temp_hi = temp_hi+15, prcp = DEFAULT
WHERE city = 'San Francisco' AND date = '2016-07-03';
```

Use the alternative column-list syntax to do the same update:

```sql
UPDATE weather SET (temp_lo, temp_hi, prcp) = (temp_lo+1, temp_hi+15, DEFAULT)
WHERE city = 'San Francisco' AND date = '2016-07-03';
```

Increment the sales count of the salesperson who manages the account for Acme Corporation, using the **FROM** clause syntax (assuming both tables being joined are distributed in Greenplum Database on the **id** column):

```sql
UPDATE employees SET sales_count = sales_count + 1 FROM accounts
WHERE accounts.name = 'Acme Corporation'
AND employees.id = accounts.id;
```

Perform the same operation, using a sub-select in the **WHERE** clause:

```sql
UPDATE employees SET sales_count = sales_count + 1 WHERE id =
(SELECT id FROM accounts WHERE name = 'Acme Corporation');
```

Attempt to insert a new stock item along with the quantity of stock. If the item already exists, instead update the stock count of the existing item. To do this without failing the entire transaction, use savepoints.

```sql
BEGIN;
-- other operations
SAVEPOINT sp1;
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;

Compatibility
This command conforms to the SQL standard, except that the FROM clause is a Greenplum Database extension.

According to the standard, the column-list syntax should allow a list of columns to be assigned from a single row-valued expression, such as a sub-select:

```sql
UPDATE accounts SET (contact_last_name, contact_first_name) =
(SELECT last_name, first_name FROM salesmen
WHERE salesmen.id = accounts.sales_id);
```

This is not currently implemented — the source must be a list of independent expressions.

Some other database systems offer a FROM option in which the target table is supposed to be listed again within FROM. That is not how Greenplum Database interprets FROM. Be careful when porting applications that use this extension.

See Also
DECLARE, DELETE, SELECT, INSERT

VACUUM
Garbage-collects and optionally analyzes a database.

Synopsis

```sql
VACUUM [FULL] [FREEZE] [VERBOSE] [table]
VACUUM [FULL] [FREEZE] [VERBOSE] ANALYZE [table [(column [, ... )]]
```

Description

VACUUM reclaims storage occupied by deleted tuples. In normal Greenplum Database operation, tuples that are deleted or obsoleted by an update are not physically removed from their table; they remain present on disk until a VACUUM is done. Therefore it is necessary to do VACUUM periodically, especially on frequently-updated tables.

With no parameter, VACUUM processes every table in the current database. With a parameter, VACUUM processes only that table.

VACUUM ANALYZE performs a VACUUM and then an ANALYZE for each selected table. This is a handy combination form for routine maintenance scripts. See ANALYZE for more details about its processing.

VACUUM (without FULL) marks deleted and obsoleted data in tables and indexes for future reuse and reclaims space for re-use only if the space is at the end of the table and an exclusive table lock can be easily obtained. Unused space at the start or middle of a table remains as is. With heap tables, this form of the command can operate in parallel with normal reading and writing of the table, as an exclusive lock is not obtained.

With append-optimized tables, VACUUM compacts a table by first vacuuming the indexes, then compacting each segment file in turn, and finally vacuuming auxiliary relations and updating statistics. On each segment, visible rows are copied from the current segment file to a new segment file, and then the current segment file is scheduled to be dropped and the new segment file is made available. Plain VACUUM of
an append-optimized table allows scans, inserts, deletes, and updates of the table while a segment file is compacted. However, an Access Exclusive lock is taken briefly to drop the current segment file and activate the new segment file.

**VACUUM FULL** does more extensive processing, including moving of tuples across blocks to try to compact the table to the minimum number of disk blocks. This form is much slower and requires an Access Exclusive lock on each table while it is being processed. The Access Exclusive lock guarantees that the holder is the only transaction accessing the table in any way.

**Important:** For information on the use of `VACUUM`, `VACUUM FULL`, and `VACUUM ANALYZE`, see [Notes](#).

**Outputs**

When `VERBOSE` is specified, `VACUUM` emits progress messages to indicate which table is currently being processed. Various statistics about the tables are printed as well.

**Parameters**

**FULL**

Selects a full vacuum, which may reclaim more space, but takes much longer and exclusively locks the table.

**FREEZE**

Specifying `FREEZE` is equivalent to performing `VACUUM` with the `vacuum_freeze_min_age` server configuration parameter set to zero. See [Server Configuration Parameters](#) for information about `vacuum_freeze_min_age`.

**VERBOSE**

Prints a detailed vacuum activity report for each table.

**ANALYZE**

Updates statistics used by the planner to determine the most efficient way to execute a query.

**table**

The name (optionally schema-qualified) of a specific table to vacuum. Defaults to all tables in the current database.

**column**

The name of a specific column to analyze. Defaults to all columns.

**Notes**

`VACUUM` cannot be executed inside a transaction block.

Vacuum active databases frequently (at least nightly), in order to remove expired rows. After adding or deleting a large number of rows, running the `VACUUM ANALYZE` command for the affected table might be useful. This updates the system catalogs with the results of all recent changes, and allows the Greenplum Database query optimizer to make better choices in planning queries.

**Important:** Regular PostgreSQL has a separate optional server process called the *autovacuum daemon*, whose purpose is to automate the execution of `VACUUM` and `ANALYZE` commands. This feature is currently disabled in Greenplum Database.

`VACUUM` causes a substantial increase in I/O traffic, which can cause poor performance for other active sessions. Therefore, it is advisable to vacuum the database at low usage times.

For heap tables, expired rows are held in what is called the *free space map*. The free space map must be sized large enough to cover the dead rows of all heap tables in your database. If not sized large enough, space occupied by dead rows that overflow the free space map cannot be reclaimed by a regular `VACUUM` command.
VACUUM commands skip external tables.

VACUUM FULL reclaims all expired row space, however it requires an exclusive lock on each table being processed, is a very expensive operation, and might take a long time to complete on large, distributed Greenplum Database tables. Perform VACUUM FULL operations during database maintenance periods.

As an alternative to VACUUM FULL, you can re-create the table with a CREATE TABLE AS statement and drop the old table.

Size the free space map appropriately. You configure the free space map using the following server configuration parameters:

- max_fsm_pages
- max_fsm_relations

For append-optimized tables, VACUUM requires enough available disk space to accommodate the new segment file during the VACUUM process. If the ratio of hidden rows to total rows in a segment file is less than a threshold value (10, by default), the segment file is not compacted. The threshold value can be configured with the gp_appendonly_compaction_threshold server configuration parameter. VACUUM FULL ignores the threshold and rewrites the segment file regardless of the ratio. VACUUM can be disabled for append-optimized tables using the gp_appendonly_compaction server configuration parameter. See Server Configuration Parameters for information about the server configuration parameters.

If a concurrent serializable transaction is detected when an append-optimized table is being vacuumed, the current and subsequent segment files are not compacted. If a segment file has been compacted but a concurrent serializable transaction is detected in the transaction that drops the original segment file, the drop is skipped. This could leave one or two segment files in an "awaiting drop" state after the vacuum has completed.

For more information about concurrency control in Greenplum Database, see "Routine System Maintenance Tasks" in Greenplum Database Administrator Guide.

Examples

Vacuum all tables in the current database:

```sql
VACUUM;
```

Vacuum a specific table only:

```sql
VACUUM mytable;
```

Vacuum all tables in the current database and collect statistics for the query optimizer:

```sql
VACUUM ANALYZE;
```

Compatibility

There is no VACUUM statement in the SQL standard.

See Also

ANALYZE

VALUES

Computes a set of rows.
Synopsis

VALUES ( expression [, ...] ) [, ...]
    [ORDER BY sort_expression [ASC | DESC | USING operator] [, ...]]
    [LIMIT {count | ALL}] [OFFSET start]

Description

VALUES computes a row value or set of row values specified by value expressions. It is most commonly used to generate a "constant table" within a larger command, but it can be used on its own.

When more than one row is specified, all the rows must have the same number of elements. The data types of the resulting table's columns are determined by combining the explicit or inferred types of the expressions appearing in that column, using the same rules as for UNION.

Within larger commands, VALUES is syntactically allowed anywhere that SELECT is. Because it is treated like a SELECT by the grammar, it is possible to use the ORDERBY, LIMIT, and OFFSET clauses with a VALUES command.

Parameters

expression
A constant or expression to compute and insert at the indicated place in the resulting table (set of rows). In a VALUES list appearing at the top level of an INSERT, an expression can be replaced by DEFAULT to indicate that the destination column's default value should be inserted. DEFAULT cannot be used when VALUES appears in other contexts.

sort_expression
An expression or integer constant indicating how to sort the result rows. This expression may refer to the columns of the VALUES result as column1, column2, etc. For more details, see "The ORDER BY Clause" in the parameters for SELECT.

operator
A sorting operator. For more details, see "The ORDER BY Clause" in the parameters for SELECT.

LIMIT count
OFFSET start
The maximum number of rows to return. For more details, see "The LIMIT Clause" in the parameters for SELECT.

Notes

VALUES lists with very large numbers of rows should be avoided, as you may encounter out-of-memory failures or poor performance. VALUES appearing within INSERT is a special case (because the desired column types are known from the INSERT's target table, and need not be inferred by scanning the VALUES list), so it can handle larger lists than are practical in other contexts.

Examples

A bare VALUES command:

VALUES (1, 'one'), (2, 'two'), (3, 'three');

This will return a table of two columns and three rows. It is effectively equivalent to:

```
SELECT 1 AS column1, 'one' AS column2
UNION ALL
SELECT 2, 'two'
UNION ALL
```
More usually, `VALUES` is used within a larger SQL command. The most common use is in `INSERT`:

```sql
SELECT 3, 'three';
```

In the context of `INSERT`, entries of a `VALUES` list can be `DEFAULT` to indicate that the column default should be used here instead of specifying a value:

```sql
INSERT INTO films (code, title, did, date_prod, kind) VALUES ('T_601', 'Yojimbo', 106, '1961-06-16', 'Drama');
```

VALUES can also be used where a sub-SELECT might be written, for example in a `FROM` clause:

```sql
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;
```

Note that an `AS` clause is required when `VALUES` is used in a `FROM` clause, just as is true for `SELECT`. It is not required that the `AS` clause specify names for all the columns, but it is good practice to do so. The default column names for `VALUES` are `column1`, `column2`, etc. in Greenplum Database, but these names might be different in other database systems.

When `VALUES` is used in `INSERT`, the values are all automatically coerced to the data type of the corresponding destination column. When it is used in other contexts, it may be necessary to specify the correct data type. If the entries are all quoted literal constants, coercing the first is sufficient to determine the assumed type for all:

```sql
SELECT * FROM machines WHERE ip_address IN (VALUES('192.168.0.1'::inet), ('192.168.0.10'), ('192.0.2.43'));
```

Note: For simple `IN` tests, it is better to rely on the list-of-scalars form of `IN` than to write a `VALUES` query as shown above. The list of scalars method requires less writing and is often more efficient.

**Compatibility**

`VALUES` conforms to the SQL standard, except that `LIMIT` and `OFFSET` are Greenplum Database extensions.

**See Also**

`INSERT`, `SELECT`
The following table lists the features described in the 2008 SQL standard. Features that are supported in Greenplum Database are marked as YES in the 'Supported' column, features that are not implemented are marked as NO.

For information about Greenplum features and SQL compliance, see the Greenplum Database Administrator Guide.

Table 94: SQL 2008 Optional Feature Compliance Details

<table>
<thead>
<tr>
<th>ID</th>
<th>Feature</th>
<th>Supported</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>B011</td>
<td>Embedded Ada</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B012</td>
<td>Embedded C</td>
<td>NO</td>
<td>Due to issues with PostgreSQL ecpg</td>
</tr>
<tr>
<td>B013</td>
<td>Embedded COBOL</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B014</td>
<td>Embedded Fortran</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B015</td>
<td>Embedded MUMPS</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B016</td>
<td>Embedded Pascal</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B017</td>
<td>Embedded PL/I</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B021</td>
<td>Direct SQL</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>B031</td>
<td>Basic dynamic SQL</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B032</td>
<td>Extended dynamic SQL</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B033</td>
<td>Untyped SQL-invoked function arguments</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B034</td>
<td>Dynamic specification of cursor attributes</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B035</td>
<td>Non-extended descriptor names</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B041</td>
<td>Extensions to embedded SQL exception declarations</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B051</td>
<td>Enhanced execution rights</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B111</td>
<td>Module language Ada</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B112</td>
<td>Module language C</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B113</td>
<td>Module language COBOL</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B114</td>
<td>Module language Fortran</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B115</td>
<td>Module language MUMPS</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>B116</td>
<td>Module language Pascal</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B117</td>
<td>Module language PL/I</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B121</td>
<td>Routine language Ada</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B122</td>
<td>Routine language C</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B123</td>
<td>Routine language COBOL</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B124</td>
<td>Routine language Fortran</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B125</td>
<td>Routine language MUMPS</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B126</td>
<td>Routine language Pascal</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B127</td>
<td>Routine language PL/I</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>B128</td>
<td>Routine language SQL</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>E011</td>
<td>Numeric data types</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E011-01</td>
<td>INTEGER and SMALLINT data types</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E011-02</td>
<td>DOUBLE PRECISION and FLOAT data types</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E011-03</td>
<td>DECIMAL and NUMERIC data types</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E011-04</td>
<td>Arithmetic operators</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E011-05</td>
<td>Numeric comparison</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E011-06</td>
<td>Implicit casting among the numeric data types</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E021</td>
<td>Character data types</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E021-01</td>
<td>CHARACTER data type</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E021-02</td>
<td>CHARACTER VARYING data type</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E021-03</td>
<td>Character literals</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E021-04</td>
<td>CHARACTER_LENGTH function</td>
<td>YES</td>
<td>Trims trailing spaces from CHARACTER values before counting</td>
</tr>
<tr>
<td>E021-05</td>
<td>OCTET_LENGTH function</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E021-06</td>
<td>SUBSTRING function</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E021-07</td>
<td>Character concatenation</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E021-08</td>
<td>UPPER and LOWER functions</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E021-09</td>
<td>TRIM function</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------------------------</td>
<td>-----------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>E021-10</td>
<td>Implicit casting among the character string types</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E021-11</td>
<td>POSITION function</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E021-12</td>
<td>Character comparison</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E031</td>
<td>Identifiers</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E031-01</td>
<td>Delimited identifiers</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E031-02</td>
<td>Lower case identifiers</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E031-03</td>
<td>Trailing underscore</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E051</td>
<td>Basic query specification</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E051-01</td>
<td>SELECT DISTINCT</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E051-02</td>
<td>GROUP BY clause</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E051-03</td>
<td>GROUP BY can contain columns not in SELECT list</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E051-04</td>
<td>SELECT list items can be renamed</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E051-05</td>
<td>HAVING clause</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E051-06</td>
<td>Qualified * in SELECT list</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E051-07</td>
<td>Correlation names in the FROM clause</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E051-08</td>
<td>Rename columns in the FROM clause</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E061</td>
<td>Basic predicates and search conditions</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E061-01</td>
<td>Comparison predicate</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E061-02</td>
<td>BETWEEN predicate</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E061-03</td>
<td>IN predicate with list of values</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E061-04</td>
<td>LIKE predicate</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E061-05</td>
<td>LIKE predicate ESCAPE clause</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E061-06</td>
<td>NULL predicate</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E061-07</td>
<td>Quantified comparison predicate</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E061-08</td>
<td>EXISTS predicate</td>
<td>YES</td>
<td>Not all uses work in Greenplum</td>
</tr>
<tr>
<td>E061-09</td>
<td>Subqueries in comparison predicate</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------------------------</td>
<td>-----------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>E061-11</td>
<td>Subqueries in IN predicate</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E061-12</td>
<td>Subqueries in quantified comparison predicate</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E061-13</td>
<td>Correlated subqueries</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E061-14</td>
<td>Search condition</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E071</td>
<td>Basic query expressions</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E071-01</td>
<td>UNION DISTINCT table operator</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E071-02</td>
<td>UNION ALL table operator</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E071-03</td>
<td>EXCEPT DISTINCT table operator</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E071-05</td>
<td>Columns combined via table operators need not have exactly the same data type</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E071-06</td>
<td>Table operators in subqueries</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E081</td>
<td>Basic Privileges</td>
<td>NO</td>
<td>Partial sub-feature support</td>
</tr>
<tr>
<td>E081-01</td>
<td>SELECT privilege</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E081-02</td>
<td>DELETE privilege</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E081-03</td>
<td>INSERT privilege at the table level</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E081-04</td>
<td>UPDATE privilege at the table level</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E081-05</td>
<td>UPDATE privilege at the column level</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>E081-06</td>
<td>REFERENCES privilege at the table level</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>E081-07</td>
<td>REFERENCES privilege at the column level</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>E081-08</td>
<td>WITH GRANT OPTION</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E081-09</td>
<td>USAGE privilege</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E081-10</td>
<td>EXECUTE privilege</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E091</td>
<td>Set Functions</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E091-01</td>
<td>AVG</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E091-02</td>
<td>COUNT</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E091-03</td>
<td>MAX</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------------------</td>
<td>-----------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>E091-04</td>
<td>MIN</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E091-05</td>
<td>SUM</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E091-06</td>
<td>ALL quantifier</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E091-07</td>
<td>DISTINCT quantifier</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E101</td>
<td>Basic data manipulation</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E101-01</td>
<td>INSERT statement</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E101-03</td>
<td>Searched UPDATE statement</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E101-04</td>
<td>Searched DELETE statement</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E111</td>
<td>Single row SELECT statement</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E121</td>
<td>Basic cursor support</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E121-01</td>
<td>DECLARE CURSOR</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E121-02</td>
<td>ORDER BY columns need not be in select list</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E121-03</td>
<td>Value expressions in ORDER BY clause</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E121-04</td>
<td>OPEN statement</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E121-06</td>
<td>Positioned UPDATE statement</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>E121-07</td>
<td>Positioned DELETE statement</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>E121-08</td>
<td>CLOSE statement</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E121-10</td>
<td>FETCH statement implicit NEXT</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E121-17</td>
<td>WITH HOLD cursors</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E131</td>
<td>Null value support</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E141</td>
<td>Basic integrity constraints</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E141-01</td>
<td>NOT NULL constraints</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E141-02</td>
<td>UNIQUE constraints of NOT NULL columns</td>
<td>YES</td>
<td>Must be the same as or a superset of the Greenplum distribution key</td>
</tr>
<tr>
<td>E141-03</td>
<td>PRIMARY KEY constraints</td>
<td>YES</td>
<td>Must be the same as or a superset of the Greenplum distribution key</td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------------------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E141-04</td>
<td>Basic FOREIGN KEY constraint with the NO ACTION default for both referential delete action and referential update action</td>
<td>NO</td>
<td>Foreign keys can be declared but are not enforced in Greenplum</td>
</tr>
<tr>
<td>E141-06</td>
<td>CHECK constraints</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E141-07</td>
<td>Column defaults</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E141-08</td>
<td>NOT NULL inferred on PRIMARY KEY</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E141-10</td>
<td>Names in a foreign key can be specified in any order</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E151</td>
<td>Transaction support</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E151-01</td>
<td>COMMIT statement</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E151-02</td>
<td>ROLLBACK statement</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E152</td>
<td>Basic SET TRANSACTION statement</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E152-01</td>
<td>ISOLATION LEVEL SERIALIZABLE clause</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E152-02</td>
<td>READ ONLY and READ WRITE clauses</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E153</td>
<td>Updatable queries with subqueries</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>E153</td>
<td>SQL comments using leading double minus</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E171</td>
<td>SQLSTATE support</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>E182</td>
<td>Module language</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F021</td>
<td>Basic information schema</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F021-01</td>
<td>COLUMNS view</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F021-02</td>
<td>TABLES view</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F021-03</td>
<td>VIEWS view</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F021-04</td>
<td>TABLE_CONSTRAINTS view</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F021-05</td>
<td>REFERENTIAL_CONSTRAINTS view</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F021-06</td>
<td>CHECK_CONSTRAINTS view</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------------------------------------------------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>F031</td>
<td>Basic schema manipulation</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F031-01</td>
<td>CREATE TABLE statement to create persistent base tables</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F031-02</td>
<td>CREATE VIEW statement</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F031-03</td>
<td>GRANT statement</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F031-04</td>
<td>ALTER TABLE statement: ADD COLUMN clause</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F031-13</td>
<td>DROP TABLE statement: RESTRICT clause</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F031-16</td>
<td>DROP VIEW statement: RESTRICT clause</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F031-19</td>
<td>REVOKE statement: RESTRICT clause</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F032</td>
<td>CASCADE drop behavior</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F033</td>
<td>ALTER TABLE statement: DROP COLUMN clause</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F034</td>
<td>Extended REVOKE statement</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F034-01</td>
<td>REVOKE statement performed by other than the owner of a schema object</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F034-02</td>
<td>REVOKE statement: GRANT OPTION FOR clause</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F034-03</td>
<td>REVOKE statement to revoke a privilege that the grantee has WITH GRANT OPTION</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F041</td>
<td>Basic joined table</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F041-01</td>
<td>Inner join (but not necessarily the INNER keyword)</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F041-02</td>
<td>INNER keyword</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F041-03</td>
<td>LEFT OUTER JOIN</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F041-04</td>
<td>RIGHT OUTER JOIN</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F041-05</td>
<td>Outer joins can be nested</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------------------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>F041-07</td>
<td>The inner table in a left or right outer join can also be used in an inner join</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F041-08</td>
<td>All comparison operators are supported (rather than just =)</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F051</td>
<td>Basic date and time</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F051-01</td>
<td>DATE data type (including support of DATE literal)</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F051-02</td>
<td>TIME data type (including support of TIME literal) with fractional seconds precision of at least 0</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F051-03</td>
<td>TIMESTAMP data type (including support of TIMESTAMP literal) with fractional seconds precision of at least 0 and 6</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F051-04</td>
<td>Comparison predicate on DATE, TIME, and TIMESTAMP data types</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F051-05</td>
<td>Explicit CAST between datetime types and character string types</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F051-06</td>
<td>CURRENT_DATE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F051-07</td>
<td>LOCALTIME</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F051-08</td>
<td>LOCALTIMESTAMP</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F052</td>
<td>Intervals and datetime arithmetic</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F053</td>
<td>OVERLAPS predicate</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F081</td>
<td>UNION and EXCEPT in views</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F111</td>
<td>Isolation levels other than SERIALIZABLE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F111-01</td>
<td>READ UNCOMMITTED isolation level</td>
<td>NO</td>
<td>Can be declared but is treated as a synonym for READ COMMITTED</td>
</tr>
<tr>
<td>F111-02</td>
<td>READ COMMITTED isolation level</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------------------------------</td>
<td>-----------</td>
<td>------------------</td>
</tr>
<tr>
<td>F111-03</td>
<td><strong>REPEATABLE READ isolation level</strong></td>
<td>NO</td>
<td>Use <strong>SERIALIZABLE</strong></td>
</tr>
<tr>
<td>F121</td>
<td>Basic diagnostics management</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F122</td>
<td>Enhanced diagnostics management</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F123</td>
<td>All diagnostics</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F131-</td>
<td>Grouped operations</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F131-01</td>
<td><strong>WHERE, GROUP BY, and HAVING clauses supported in queries with grouped views</strong></td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F131-02</td>
<td>Multiple tables supported in queries with grouped views</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F131-03</td>
<td>Set functions supported in queries with grouped views</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F131-04</td>
<td><strong>Subqueries with GROUP BY and HAVING clauses and grouped views</strong></td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F131-05</td>
<td><strong>Single row SELECT with GROUP BY and HAVING clauses and grouped views</strong></td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F171</td>
<td>Multiple schemas per user</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F181</td>
<td>Multiple module support</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F191</td>
<td>Referential delete actions</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F200</td>
<td><strong>TRUNCATE TABLE statement</strong></td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F201</td>
<td><strong>CAST function</strong></td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F202</td>
<td><strong>TRUNCATE TABLE: identity column restart option</strong></td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F221</td>
<td>Explicit defaults</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F222</td>
<td><strong>INSERT statement: DEFAULT VALUES clause</strong></td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F231</td>
<td>Privilege tables</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F231-01</td>
<td><strong>TABLE_PRIVILEGES view</strong></td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------------------------------</td>
<td>-----------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>F231-02</td>
<td>COLUMN_PRIVILEGES view</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F231-03</td>
<td>USAGE_PRIVILEGES view</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F251</td>
<td>Domain support</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F261</td>
<td>CASE expression</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F261-01</td>
<td>Simple CASE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F261-02</td>
<td>Searched CASE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F261-03</td>
<td>NULLIF</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F261-04</td>
<td>COALESCE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F262</td>
<td>Extended CASE expression</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F263</td>
<td>Comma-separated predicates in simple CASE expression</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F271</td>
<td>Compound character literals</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F281</td>
<td>LIKE enhancements</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F291</td>
<td>UNIQUE predicate</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F301</td>
<td>CORRESPONDING in query expressions</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F302</td>
<td>INTERSECT table operator</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F302-01</td>
<td>INTERSECT DISTINCT table operator</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F302-02</td>
<td>INTERSECT ALL table operator</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F304</td>
<td>EXCEPT ALL table operator</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F311</td>
<td>Schema definition statement</td>
<td>YES</td>
<td>Partial sub-feature support</td>
</tr>
<tr>
<td>F311-01</td>
<td>CREATE SCHEMA</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F311-02</td>
<td>CREATE TABLE for persistent base tables</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F311-03</td>
<td>CREATE VIEW</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F311-04</td>
<td>CREATE VIEW: WITH CHECK OPTION</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F311-05</td>
<td>GRANT statement</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F312</td>
<td>MERGE statement</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>-------</td>
<td>------------------------------------------------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>F313</td>
<td>Enhanced MERGE statement</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F321</td>
<td>User authorization</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F341</td>
<td>Usage Tables</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F361</td>
<td>Subprogram support</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F381</td>
<td>Extended schema manipulation</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F381-01</td>
<td>ALTER TABLE statement: ALTER COLUMN clause</td>
<td></td>
<td>Some limitations on altering distribution key columns</td>
</tr>
<tr>
<td>F381-02</td>
<td>ALTER TABLE statement: ADD CONSTRAINT clause</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F381-03</td>
<td>ALTER TABLE statement: DROP CONSTRAINT clause</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F382</td>
<td>Alter column data type</td>
<td>YES</td>
<td>Some limitations on altering distribution key columns</td>
</tr>
<tr>
<td>F391</td>
<td>Long identifiers</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F392</td>
<td>Unicode escapes in identifiers</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F393</td>
<td>Unicode escapes in literals</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F394</td>
<td>Optional normal form specification</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F401</td>
<td>Extended joined table</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F401-01</td>
<td>NATURAL JOIN</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F401-02</td>
<td>FULL OUTER JOIN</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F401-04</td>
<td>CROSS JOIN</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F402</td>
<td>Named column joins for LOBs, arrays, and multisets</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F403</td>
<td>Partitioned joined tables</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F411</td>
<td>Time zone specification</td>
<td>YES</td>
<td>Differences regarding literal interpretation</td>
</tr>
<tr>
<td>F421</td>
<td>National character</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F431</td>
<td>Read-only scrollable cursors</td>
<td>YES</td>
<td>Forward scrolling only</td>
</tr>
<tr>
<td>01</td>
<td>FETCH with explicit NEXT</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>----</td>
<td>--------------------------------------</td>
<td>-----------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>02</td>
<td>FETCH FIRST</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>FETCH LAST</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>FETCH PRIOR</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>FETCH ABSOLUTE</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>FETCH RELATIVE</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F441</td>
<td>Extended set function support</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F442</td>
<td>Mixed column references in set functions</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F451</td>
<td>Character set definition</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F461</td>
<td>Named character sets</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F471</td>
<td>Scalar subquery values</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F481</td>
<td>Expanded NULL predicate</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F491</td>
<td>Constraint management</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F501</td>
<td>Features and conformance views</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F501-01</td>
<td>SQL_FEATURES view</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F501-02</td>
<td>SQL_SIZING view</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F501-03</td>
<td>SQL_LANGUAGES view</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F502</td>
<td>Enhanced documentation tables</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F502-01</td>
<td>SQL_SIZING_PROFILES view</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F502-02</td>
<td>SQL_IMPLEMENTATION_INFO view</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F502-03</td>
<td>SQL_PACKAGES view</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F521</td>
<td>Assertions</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F531</td>
<td>Temporary tables</td>
<td>YES</td>
<td>Non-standard form</td>
</tr>
<tr>
<td>F555</td>
<td>Enhanced seconds precision</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F561</td>
<td>Full value expressions</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F571</td>
<td>Truth value tests</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F591</td>
<td>Derived tables</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F611</td>
<td>Indicator data types</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>F641</td>
<td>Row and table constructors</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F651</td>
<td>Catalog name qualifiers</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F661</td>
<td>Simple tables</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F671</td>
<td>Subqueries in <code>CHECK</code></td>
<td>NO</td>
<td>Intentionally omitted</td>
</tr>
<tr>
<td>F672</td>
<td>Retrospective check constraints</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F690</td>
<td>Collation support</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F692</td>
<td>Enhanced collation support</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F693</td>
<td>SQL-session and client module collations</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F695</td>
<td>Translation support</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F696</td>
<td>Additional translation documentation</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F701</td>
<td>Referential update actions</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F711</td>
<td><code>ALTER</code> domain</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F721</td>
<td>Deferrable constraints</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F731</td>
<td><code>INSERT</code> column privileges</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F741</td>
<td>Referential <code>MATCH</code> types</td>
<td>NO</td>
<td>No partial match</td>
</tr>
<tr>
<td>F751</td>
<td><code>View</code> <code>CHECK</code> enhancements</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F761</td>
<td>Session management</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F762</td>
<td><code>CURRENT_CATALOG</code></td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F763</td>
<td><code>CURRENT_SCHEMA</code></td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F771</td>
<td>Connection management</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F781</td>
<td>Self-referencing operations</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F791</td>
<td>Insensitive cursors</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F801</td>
<td>Full set function</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F812</td>
<td>Basic flagging</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F813</td>
<td>Extended flagging</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F831</td>
<td>Full cursor update</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F841</td>
<td><code>LIKE_REGEX</code> predicate</td>
<td>NO</td>
<td>Non-standard syntax for regex</td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------------------</td>
<td>-----------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>F842</td>
<td>OCCURRENCES_REGEX function</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F843</td>
<td>POSITION_REGEX function</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F844</td>
<td>SUBSTRING_REGEX function</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F845</td>
<td>TRANSLATE_REGEX function</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F846</td>
<td>Octet support in regular expression operators</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F847</td>
<td>Nonconstant regular expressions</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F850</td>
<td>Top-level ORDER BY clause in query expression</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>F851</td>
<td>Top-level ORDER BY clause in subqueries</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F852</td>
<td>Top-level ORDER BY clause in views</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F855</td>
<td>Nested ORDER BY clause in query expression</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F856</td>
<td>Nested FETCH FIRST clause in query expression</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F857</td>
<td>Top-level FETCH FIRST clause in query expression</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F858</td>
<td>FETCH FIRST clause in subqueries</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F859</td>
<td>Top-level FETCH FIRST clause in views</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F860</td>
<td>FETCH FIRST ROW count in FETCH FIRST clause</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F861</td>
<td>Top-level RESULT OFFSET clause in query expression</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F862</td>
<td>RESULT OFFSET clause in subqueries</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F863</td>
<td>Nested RESULT OFFSET clause in query expression</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>F864</td>
<td>Top-level <code>RESULT OFFSET</code> clause in views</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>F865</td>
<td><code>OFFSET ROW count</code> in <code>RESULT OFFSET</code> clause</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S011</td>
<td>Distinct data types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S023</td>
<td>Basic structured types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S024</td>
<td>Enhanced structured types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S025</td>
<td>Final structured types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S026</td>
<td>Self-referencing structured types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S027</td>
<td>Create method by specific method name</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S028</td>
<td>Permutable UDT options list</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S041</td>
<td>Basic reference types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S043</td>
<td>Enhanced reference types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S051</td>
<td>Create table of type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S071</td>
<td>SQL paths in function and type name resolution</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>S091</td>
<td>Basic array support</td>
<td>NO</td>
<td>Greenplum has arrays, but is not fully standards compliant</td>
</tr>
<tr>
<td>S091-01</td>
<td>Arrays of built-in data types</td>
<td>NO</td>
<td>Partially compliant</td>
</tr>
<tr>
<td>S091-02</td>
<td>Arrays of distinct types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S091-03</td>
<td>Array expressions</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S092</td>
<td>Arrays of user-defined types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S094</td>
<td>Arrays of reference types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S095</td>
<td>Array constructors by query</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S096</td>
<td>Optional array bounds</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S097</td>
<td>Array element assignment</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>S098</td>
<td>ARRAY_AGG</td>
<td>Partially</td>
<td>Supported: Using array_agg without a window specification; for example</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SELECT array_agg(x) FROM ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SELECT array_agg (x order by y) FROM ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Not supported: Using array_agg as an aggregate derived window function; for example</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SELECT array_agg(x) over (ORDER BY y) FROM ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SELECT array_agg(x) over (PARTITION BY z) FROM ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SELECT array_agg(x) over (ORDER BY z) FROM ...</td>
</tr>
<tr>
<td>S111</td>
<td>ONLY in query expressions</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>S151</td>
<td>Type predicate</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S161</td>
<td>Subtype treatment</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S162</td>
<td>Subtype treatment for references</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S201</td>
<td>SQL-invoked routines on arrays</td>
<td>NO</td>
<td>Functions can be passed Greenplum array types</td>
</tr>
<tr>
<td>S202</td>
<td>SQL-invoked routines on multisets</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S211</td>
<td>User-defined cast functions</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>S231</td>
<td>Structured type locators</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S232</td>
<td>Array locators</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S233</td>
<td>Multiset locators</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S241</td>
<td>Transform functions</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S242</td>
<td>Alter transform statement</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S251</td>
<td>User-defined orderings</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>S261</td>
<td>Specific type method</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S271</td>
<td>Basic multiset support</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S272</td>
<td>Multisets of user-defined types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S274</td>
<td>Multisets of reference types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S275</td>
<td>Advanced multiset support</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S281</td>
<td>Nested collection types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S291</td>
<td>Unique constraint on entire row</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S301</td>
<td>Enhanced UNNEST</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S401</td>
<td>Distinct types based on array types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S402</td>
<td>Distinct types based on distinct types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S403</td>
<td>MAX_CARDINALITY</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>S404</td>
<td>TRIM ARRAY</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T011</td>
<td>Timestamp in Information Schema</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T021</td>
<td>BINARY and VARBINARY data types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T022</td>
<td>Advanced support for BINARY and VARBINARY data types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T023</td>
<td>Compound binary literal</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T024</td>
<td>Spaces in binary literals</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T031</td>
<td>BOOLEAN data type</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T041</td>
<td>Basic LOB data type support</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T042</td>
<td>Extended LOB data type support</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T043</td>
<td>Multiplier T</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T044</td>
<td>Multiplier P</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T051</td>
<td>Row types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T052</td>
<td>MAX and MIN for row types</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T053</td>
<td>Explicit aliases for all-fields reference</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>------</td>
<td>------------------------------------------------------</td>
<td>-----------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>T061</td>
<td>UCS support</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T071</td>
<td>BIGINT data type</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T101</td>
<td>Enhanced nullability determination</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T111</td>
<td>Updatable joins, unions, and columns</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T121</td>
<td>WITH (excluding RECURSIVE) in query expression</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T122</td>
<td>WITH (excluding RECURSIVE) in subquery</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T131</td>
<td>Recursive query</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T132</td>
<td>Recursive query in subquery</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T141</td>
<td>SIMILAR predicate</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T151</td>
<td>DISTINCT predicate</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T152</td>
<td>DISTINCT predicate with negation</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T171</td>
<td>LIKE clause in table definition</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T172</td>
<td>AS subquery clause in table definition</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T173</td>
<td>Extended LIKE clause in table definition</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T174</td>
<td>Identity columns</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T175</td>
<td>Generated columns</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T176</td>
<td>Sequence generator support</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T177</td>
<td>Sequence generator support: simple restart option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T178</td>
<td>Identity columns: simple restart option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T191</td>
<td>Referential action RESTRICT</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T201</td>
<td>Comparable data types for referential constraints</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T211</td>
<td>Basic trigger capability</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------------------------------------</td>
<td>-----------</td>
<td>-------------------</td>
</tr>
<tr>
<td>T211-01</td>
<td>Triggers activated on <code>UPDATE</code>, <code>INSERT</code>, or <code>DELETE</code> of one base table</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T211-02</td>
<td><code>BEFORE</code> triggers</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T211-03</td>
<td><code>AFTER</code> triggers</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T211-04</td>
<td><code>FOR EACH ROW</code> triggers</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T211-05</td>
<td>Ability to specify a search condition that must be true before the trigger is invoked</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T211-06</td>
<td>Support for run-time rules for the interaction of triggers and constraints</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T211-07</td>
<td>TRIGGER privilege</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T211-08</td>
<td>Multiple triggers for the same event are executed in the order in which they were created in the catalog</td>
<td>NO</td>
<td>Intentionally omitted</td>
</tr>
<tr>
<td>T212</td>
<td>Enhanced trigger capability</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T213</td>
<td><code>INSTEAD OF</code> triggers</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T231</td>
<td>Sensitive cursors</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T241</td>
<td><code>START TRANSACTION</code> statement</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T251</td>
<td><code>SET TRANSACTION</code> statement: <code>LOCAL</code> option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T261</td>
<td>Chained transactions</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T271</td>
<td>Savepoints</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T272</td>
<td>Enhanced savepoint management</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T281</td>
<td><code>SELECT</code> privilege with column granularity</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T285</td>
<td>Enhanced derived column names</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T301</td>
<td>Functional dependencies</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T312</td>
<td>OVERLAY function</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T321</td>
<td>Basic SQL-invoked routines</td>
<td>NO</td>
<td>Partial support</td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------------------------------</td>
<td>-----------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>T321-01</td>
<td>User-defined functions with no overloading</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T321-02</td>
<td>User-defined stored procedures with no overloading</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T321-03</td>
<td>Function invocation</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T321-04</td>
<td>CALL statement</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T321-05</td>
<td>RETURN statement</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T321-06</td>
<td>ROUTINES view</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T321-07</td>
<td>PARAMETERS view</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T322</td>
<td>Overloading of SQL-invoked functions and procedures</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T323</td>
<td>Explicit security for external routines</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T324</td>
<td>Explicit security for SQL routines</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T325</td>
<td>Qualified SQL parameter references</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T326</td>
<td>Table functions</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T331</td>
<td>Basic roles</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T332</td>
<td>Extended roles</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T351</td>
<td>Bracketed SQL comments (/*...*/ comments)</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T431</td>
<td>Extended grouping capabilities</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T432</td>
<td>Nested and concatenated GROUPING SETS</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T433</td>
<td>Multiargument GROUPING function</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T434</td>
<td>GROUP BY DISTINCT</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T441</td>
<td>ABS and MOD functions</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T461</td>
<td>Symmetric BETWEEN predicate</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T471</td>
<td>Result sets return value</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T491</td>
<td>LATERAL derived table</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T501</td>
<td>Enhanced EXISTS predicate</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------------------------------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>T511</td>
<td>Transaction counts</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T541</td>
<td>Updatable table references</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T561</td>
<td>Holdable locators</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T571</td>
<td>Array-returning external SQL-invoked functions</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T572</td>
<td>Multiset-returning external SQL-invoked functions</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T581</td>
<td>Regular expression substring function</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T591</td>
<td>UNIQUE constraints of possibly null columns</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T601</td>
<td>Local cursor references</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T611</td>
<td>Elementary OLAP operations</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T612</td>
<td>Advanced OLAP operations</td>
<td>NO</td>
<td>Partially supported</td>
</tr>
<tr>
<td>T613</td>
<td>Sampling</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T614</td>
<td>NTILE function</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T615</td>
<td>LEAD and LAG functions</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T616</td>
<td>Null treatment option for LEAD and LAG functions</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T617</td>
<td>FIRST_VALUE and LAST_VALUE function</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T618</td>
<td>NTH_VALUE</td>
<td>NO</td>
<td>Function exists in Greenplum but not all options are supported</td>
</tr>
<tr>
<td>T621</td>
<td>Enhanced numeric functions</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>T631</td>
<td>N predicate with one list element</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T641</td>
<td>Multiple column assignment</td>
<td>NO</td>
<td>Some syntax variants supported</td>
</tr>
<tr>
<td>T651</td>
<td>SQL-schema statements in SQL routines</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T652</td>
<td>SQL-dynamic statements in SQL routines</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T653</td>
<td>SQL-schema statements in external routines</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>T654</td>
<td>SQL-dynamic statements in external routines</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>T655</td>
<td>Cyclically dependent routines</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M001</td>
<td>Datalinks</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M002</td>
<td>Datalinks via SQL/CLI</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M003</td>
<td>Datalinks via Embedded SQL</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M004</td>
<td>Foreign data support</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M005</td>
<td>Foreign schema support</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M006</td>
<td>GetSQLString routine</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M007</td>
<td>TransmitRequest</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M009</td>
<td>GetOpts and GetStatistics routines</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M010</td>
<td>Foreign data wrapper support</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M011</td>
<td>Datalinks via Ada</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M012</td>
<td>Datalinks via C</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M013</td>
<td>Datalinks via COBOL</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M014</td>
<td>Datalinks via Fortran</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M015</td>
<td>Datalinks via M</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M016</td>
<td>Datalinks via Pascal</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M017</td>
<td>Datalinks via PL/I</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M018</td>
<td>Foreign data wrapper interface routines in Ada</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M019</td>
<td>Foreign data wrapper interface routines in C</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M020</td>
<td>Foreign data wrapper interface routines in COBOL</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M021</td>
<td>Foreign data wrapper interface routines in Fortran</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M022</td>
<td>Foreign data wrapper interface routines in MUMPS</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M023</td>
<td>Foreign data wrapper interface routines in Pascal</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------------------------------------</td>
<td>-----------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>M024</td>
<td>Foreign data wrapper interface routines in PL/I</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M030</td>
<td>SQL-server foreign data support</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>M031</td>
<td>Foreign data wrapper general routines</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X010</td>
<td>XML type</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X011</td>
<td>Arrays of XML type</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X012</td>
<td>Multisets of XML type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X013</td>
<td>Distinct types of XML type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X014</td>
<td>Attributes of XML type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X015</td>
<td>Fields of XML type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X016</td>
<td>Persistent XML values</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X020</td>
<td>XMLConcat</td>
<td>YES</td>
<td>xmlconcat2() supported</td>
</tr>
<tr>
<td>X025</td>
<td>XMLCast</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X030</td>
<td>XMLDocument</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X031</td>
<td>XMLElement</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X032</td>
<td>XMLForest</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X034</td>
<td>XMLAgg</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X035</td>
<td>XMLAgg: ORDER BY option</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X036</td>
<td>XMLComment</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X037</td>
<td>XMLPI</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X038</td>
<td>XMLText</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X040</td>
<td>Basic table mapping</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X041</td>
<td>Basic table mapping: nulls absent</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X042</td>
<td>Basic table mapping: null as nil</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X043</td>
<td>Basic table mapping: table as forest</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X044</td>
<td>Basic table mapping: table as element</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X045</td>
<td>Basic table mapping: with target namespace</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X046</td>
<td>Basic table mapping: data mapping</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>----</td>
<td>---------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>X047</td>
<td>Basic table mapping: metadata mapping</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X048</td>
<td>Basic table mapping: base64 encoding of binary strings</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X049</td>
<td>Basic table mapping: hex encoding of binary strings</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X051</td>
<td>Advanced table mapping: nulls not absent</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X052</td>
<td>Advanced table mapping: null as nil</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X053</td>
<td>Advanced table mapping: table as forest</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X054</td>
<td>Advanced table mapping: table as element</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X055</td>
<td>Advanced table mapping: target namespace</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X056</td>
<td>Advanced table mapping: data mapping</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X057</td>
<td>Advanced table mapping: metadata mapping</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X058</td>
<td>Advanced table mapping: base64 encoding of binary strings</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X059</td>
<td>Advanced table mapping: hex encoding of binary strings</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X060</td>
<td>XMLParse: Character string input and CONTENT option</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X061</td>
<td>XMLParse: Character string input and DOCUMENT option</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X065</td>
<td>XMLParse: BLOB input and CONTENT option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X066</td>
<td>XMLParse: BLOB input and DOCUMENT option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X068</td>
<td>XMLSerialize: BOM</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X069</td>
<td>XMLSerialize: INDENT</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>X070</td>
<td>XMLSerialize: Character string serialization and CONTENT option</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X071</td>
<td>XMLSerialize: Character string serialization and DOCUMENT option</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X072</td>
<td>XMLSerialize: Character string serialization</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X073</td>
<td>XMLSerialize: BLOB serialization and CONTENT option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X074</td>
<td>XMLSerialize: BLOB serialization and DOCUMENT option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X075</td>
<td>XMLSerialize: BLOB serialization</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X076</td>
<td>XMLSerialize: VERSION</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X077</td>
<td>XMLSerialize: explicit ENCODING option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X078</td>
<td>XMLSerialize: explicit XML declaration</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X080</td>
<td>Namespaces in XML publishing</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X081</td>
<td>Query-level XML namespace declarations</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X082</td>
<td>XML namespace declarations in DML</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X083</td>
<td>XML namespace declarations in DDL</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X084</td>
<td>XML namespace declarations in compound statements</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X085</td>
<td>Predefined namespace prefixes</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X086</td>
<td>XML namespace declarations in XMLTable</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X090</td>
<td>XML document predicate</td>
<td>NO</td>
<td>xml_is_well_formed_document() supported</td>
</tr>
<tr>
<td>X091</td>
<td>XML content predicate</td>
<td>NO</td>
<td>xml_is_well_formed_content() supported</td>
</tr>
<tr>
<td>X096</td>
<td>XMLExists</td>
<td>NO</td>
<td>xmlexists() supported</td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>X100</td>
<td>Host language support for XML: CONTENT option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X101</td>
<td>Host language support for XML: DOCUMENT option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X110</td>
<td>Host language support for XML: VARCHAR mapping</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X111</td>
<td>Host language support for XML: CLOB mapping</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X112</td>
<td>Host language support for XML: BLOB mapping</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X113</td>
<td>Host language support for XML: STRIP WHITESPACE option</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X114</td>
<td>Host language support for XML: PRESERVE WHITESPACE option</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X120</td>
<td>XML parameters in SQL routines</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X121</td>
<td>XML parameters in external routines</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>X131</td>
<td>Query-level XMLBINARY clause</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X132</td>
<td>XMLBINARY clause in DML</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X133</td>
<td>XMLBINARY clause in DDL</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X134</td>
<td>XMLBINARY clause in compound statements</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X135</td>
<td>XMLBINARY clause in subqueries</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X141</td>
<td>IS VALID predicate: data-driven case</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X142</td>
<td>IS VALID predicate: ACCORDING TO clause</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X143</td>
<td>IS VALID predicate: ELEMENT clause</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X144</td>
<td>IS VALID predicate: schema location</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------------------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>X145</td>
<td>IS VALID predicate outside check constraints</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X151</td>
<td>IS VALID predicate with DOCUMENT option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X152</td>
<td>IS VALID predicate with CONTENT option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X153</td>
<td>IS VALID predicate with SEQUENCE option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X155</td>
<td>IS VALID predicate: NAMESPACE without ELEMENT clause</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X157</td>
<td>IS VALID predicate: NO NAMESPACE with ELEMENT clause</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X160</td>
<td>Basic Information Schema for registered XML Schemas</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X161</td>
<td>Advanced Information Schema for registered XML Schemas</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X170</td>
<td>XML null handling options</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X171</td>
<td>NIL ON NO CONTENT option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X181</td>
<td>XML( DOCUMENT (UNTYPED)) type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X182</td>
<td>XML( DOCUMENT (ANY)) type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X190</td>
<td>XML( SEQUENCE) type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X191</td>
<td>XML( DOCUMENT (XMLSCHEMA )) type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X192</td>
<td>XML( CONTENT (XMLSCHEMA)) type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X200</td>
<td>XMLQuery</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X201</td>
<td>XMLQuery: RETURNING CONTENT</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X202</td>
<td>XMLQuery: RETURNING SEQUENCE</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X203</td>
<td>XMLQuery: passing a context item</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>-----</td>
<td>--------------------------------------------------------------------------</td>
<td>-----------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>X204</td>
<td>XMLQuery: initializing an XQuery variable</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X205</td>
<td>XMLQuery: EMPTY ON EMPTY option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X206</td>
<td>XMLQuery: NULL ON EMPTY option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X211</td>
<td>XML 1.1 support</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X221</td>
<td>XML passing mechanism BY VALUE</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X222</td>
<td>XML passing mechanism BY REF</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X231</td>
<td>XML( CONTENT (UNTYPED)) type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X232</td>
<td>XML( CONTENT (ANY)) type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X241</td>
<td>RETURNING CONTENT in XML publishing</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X242</td>
<td>RETURNING SEQUENCE in XML publishing</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X251</td>
<td>Persistent XML values of XML( DOCUMENT (UNTYPED)) type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X252</td>
<td>Persistent XML values of XML( DOCUMENT (ANY)) type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X253</td>
<td>Persistent XML values of XML( CONTENT (UNTYPED)) type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X254</td>
<td>Persistent XML values of XML( CONTENT (ANY)) type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X255</td>
<td>Persistent XML values of XML( SEQUENCE) type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X256</td>
<td>Persistent XML values of XML( DOCUMENT (XMLSCHEMA)) type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X257</td>
<td>Persistent XML values of XML( CONTENT (XMLSCHEMA) ) type</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X260</td>
<td>XML type: ELEMENT clause</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Feature</td>
<td>Supported</td>
<td>Comments</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------------------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>X261</td>
<td>XML type: NAMESPACE without ELEMENT clause</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X263</td>
<td>XML type: NO NAMESPACE with ELEMENT clause</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X264</td>
<td>XML type: schema location</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X271</td>
<td>XMLValidate: data-driven case</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X272</td>
<td>XMLValidate: ACCORDING TO clause</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X273</td>
<td>XMLValidate: ELEMENT clause</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X274</td>
<td>XMLValidate: schema location</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X281</td>
<td>XMLValidate: with DOCUMENT option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X282</td>
<td>XMLValidate with CONTENT option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X283</td>
<td>XMLValidate with SEQUENCE option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X284</td>
<td>XMLValidate NAMESPACE without ELEMENT clause</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X286</td>
<td>XMLValidate: NO NAMESPACE with ELEMENT clause</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X300</td>
<td>XMLTable</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X301</td>
<td>XMLTable: derived column list option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X302</td>
<td>XMLTable: ordinality column option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X303</td>
<td>XMLTable: column default option</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X304</td>
<td>XMLTable: passing a context item</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X305</td>
<td>XMLTable: initializing an XQuery variable</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>X400</td>
<td>Name and identifier mapping</td>
<td>NO</td>
<td></td>
</tr>
</tbody>
</table>
Greenplum Environment Variables

This reference lists and describes the environment variables to set for Greenplum Database. Set these in your user’s startup shell profile (such as ~/.bashrc or ~/.bash_profile), or in /etc/profile if you want to set them for all users.

- Required Environment Variables
- Optional Environment Variables

### Required Environment Variables

**Note:** GPHOME, PATH and LD_LIBRARY_PATH can be set by sourcing the greenplum_path.sh file from your Greenplum Database installation directory

**GPHOME**

This is the installed location of your Greenplum Database software. For example:

```
GPHOME=/usr/local/greenplum-db-<version>
export GPHOME
```

**PATH**

Your PATH environment variable should point to the location of the Greenplum Database bin directory. For example:

```
PATH=$GPHOME/bin:$PATH
export PATH
```

**LD_LIBRARY_PATH**

The LD_LIBRARY_PATH environment variable should point to the location of the Greenplum Database/PostgreSQL library files. For example:

```
LD_LIBRARY_PATH=$GPHOME/lib
export LD_LIBRARY_PATH
```

**MASTER_DATA_DIRECTORY**

This should point to the directory created by the gpinitsystem utility in the master data directory location. For example:

```
MASTER_DATA_DIRECTORY=/data/master/gpseg-1
export MASTER_DATA_DIRECTORY
```

### Optional Environment Variables

The following are standard PostgreSQL environment variables, which are also recognized in Greenplum Database. You may want to add the connection-related environment variables to your profile for convenience, so you do not have to type so many options on the command line for client connections. Note that these environment variables should be set on the Greenplum Database master host only.
**PGAPPNAME**

The name of the application that is usually set by an application when it connects to the server. This name is displayed in the activity view and in log entries. The `PGAPPNAME` environmental variable behaves the same as the `application_name` connection parameter. The default value for `application_name` is `psql`. The name cannot be longer than 63 characters.

**PGDATABASE**

The name of the default database to use when connecting.

**PGHOST**

The Greenplum Database master host name.

**PGHOSTADDR**

The numeric IP address of the master host. This can be set instead of or in addition to `PGHOST` to avoid DNS lookup overhead.

**PGPASSWORD**

The password used if the server demands password authentication. Use of this environment variable is not recommended for security reasons (some operating systems allow non-root users to see process environment variables via `ps`). Instead consider using the `~/.pgpass` file.

**PGPASSFILE**

The name of the password file to use for lookups. If not set, it defaults to `~/.pgpass`. See the topic about The Password File in the PostgreSQL documentation for more information.

**PGOPTIONS**

Sets additional configuration parameters for the Greenplum Database master server.

**PGPORT**

The port number of the Greenplum Database server on the master host. The default port is 5432.

**PGUSER**

The Greenplum Database user name used to connect.

**PGDATESTYLE**

Sets the default style of date/time representation for a session. (Equivalent to `SET datestyle TO...`)

**PGTZ**

Sets the default time zone for a session. (Equivalent to `SET timezone TO...`)

**PGCLIENTENCODING**

Sets the default client character set encoding for a session. (Equivalent to `SET client_encoding TO...`
System Catalog Reference

This reference describes the Greenplum Database system catalog tables and views. System tables prefixed with `gp_` relate to the parallel features of Greenplum Database. Tables prefixed with `pg_` are either standard PostgreSQL system catalog tables supported in Greenplum Database, or are related to features Greenplum that provides to enhance PostgreSQL for data warehousing workloads. Note that the global system catalog for Greenplum Database resides on the master instance.

**Warning**: Changes to Pivotal Greenplum Database system catalog tables or views are not supported. If a catalog table or view is changed by the customer, the Pivotal Greenplum Database cluster is not supported. The cluster must be reinitialized and restored by the customer.

- System Tables
- System Views
- System Catalogs Definitions

### System Tables

- `gp_configuration` (Deprecated. See `gp_segment_configuration`.)
- `gp_configuration_history`
- `gp_db_interfaces` (Deprecated. Not used.)
- `gp_distribution_policy`
- `gp_fastsequence`
- `gp_fault_strategy`
- `gp_global_sequence`
- `gp_id`
- `gp_interfaces` (Deprecated. Not used.)
- `gp_master_mirroring` (Deprecated. See `pg_stat_replication`.)
- `gp_persistent_database_node`
- `gp_persistent_filespace_node`
- `gp_persistent_relation_node`
- `gp_persistent_tablespace_node`
- `gp_relation_node`
- `gp_segment_configuration`
- `gp_version_at_initdb`
- `gexpand.status`
- `gexpand.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_depend`
• `pg_description`
• `pg_exttable`
• `pg_filespace`
• `pg_filespace_entry`
• `pg_foreign_data_wrapper` (not supported)
• `pg_foreign_server` (not supported)
• `pg_foreign_table` (not supported)
• `pg_index`
• `pg_inherits`
• `pg_language`
• `pg_largeobject`
• `pg_listener`
• `pg_namespace`
• `pg_opclass`
• `pg_operator`
• `pg_partition`
• `pg_partition_rule`
• `pg_pltemplate`
• `pg_proc`
• `pg_resgroup`
• `pg_resgroupcapability`
• `pg_resourcetype`
• `pg_resqueue`
• `pg_resqueuemachine`
• `pg_rewrite`
• `pg_shdepend`
• `pg_shdescription`
• `pg_stat_last_operation`
• `pg_stat_last_shoperation`
• `pg_statistic`
• `pg_tablespace`
• `pg_trigger`
• `pg_type`
• `pg_user_mapping` (not supported)
• `pg_window`

**System Views**

Greenplum Database provides the following system views not available in PostgreSQL.

• `gp_distributed_log`
• `gp_distributed_xacts`
• `gp_pgdatabase`
• `gp_resgroup_config`
• `gp_resgroup_status`
• `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` (Deprecated. Use `gp_toolkit.gp_resqueue_status`.)
• `pg_stat_activity`
• `pg_stat_replication`
• `pg_stat_resqueues`
• `session_level_memory_consumption` (See "Viewing Session Memory Usage Information" in the Greenplum Database Administrator Guide.)

For more information about the standard system views supported in PostgreSQL and Greenplum Database, see the following sections of the PostgreSQL documentation:
• System Views
• Statistics Collector Views
• The Information Schema

**System Catalogs Definitions**

System catalog table and view definitions in alphabetical order.

### gp_configuration_history

The `gp_configuration_history` table contains information about system changes related to fault detection and recovery operations. The `fts_probe` process logs data to this table, as do certain related management utilities such as `gpcheck`, `gprecoverseg`, and `gpinitsystem`. For example, when you add a new segment and mirror segment to the system, records for these events are logged to `gp_configuration_history`.

The event descriptions stored in this table may be helpful for troubleshooting serious system issues in collaboration with Pivotal Support technicians.

This table is populated only on the master. This table is defined in the `pg_global` tablespace, meaning it is globally shared across all databases in the system.

**Table 95: pg_catalog.gp_configuration_history**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>timestamp with time zone</td>
<td></td>
<td>Timestamp for the event recorded.</td>
</tr>
<tr>
<td>dbid</td>
<td>smallint</td>
<td><code>gp_segment_configuration.dbid</code></td>
<td>System-assigned ID. The unique identifier of a segment (or master) instance.</td>
</tr>
<tr>
<td>desc</td>
<td>text</td>
<td></td>
<td>Text description of the event.</td>
</tr>
</tbody>
</table>

For information about `gpcheck`, `gprecoverseg`, and `gpinitsystem`, see the Greenplum Database Utility Guide.
gp_distributed_log

The gp_distributed_log view contains status information about distributed transactions and their associated local transactions. A distributed transaction is a transaction that involves modifying data on the segment instances. Greenplum's distributed transaction manager ensures that the segments stay in synch. This view allows you to see the status of distributed transactions.

Table 96: pg_catalog.gp_distributed_log

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_id</td>
<td>smallint</td>
<td>gp_segment_configuration.content</td>
<td>The content id if the segment. The master is always -1 (no content).</td>
</tr>
<tr>
<td>dbid</td>
<td>small_int</td>
<td>gp_segment_configuration.dbid</td>
<td>The unique id of the segment instance.</td>
</tr>
<tr>
<td>distributed_xid</td>
<td>xid</td>
<td></td>
<td>The global transaction id.</td>
</tr>
<tr>
<td>distributed_id</td>
<td>text</td>
<td></td>
<td>A system assigned ID for a distributed transaction.</td>
</tr>
<tr>
<td>status</td>
<td>text</td>
<td></td>
<td>The status of the distributed transaction (Committed or Aborted).</td>
</tr>
<tr>
<td>local_transaction</td>
<td>xid</td>
<td></td>
<td>The local transaction ID.</td>
</tr>
</tbody>
</table>

gp_distributed_xacts

The gp_distributed_xacts view contains information about Greenplum Database distributed transactions. A distributed transaction is a transaction that involves modifying data on the segment instances. Greenplum's distributed transaction manager ensures that the segments stay in synch. This view allows you to see the currently active sessions and their associated distributed transactions.

Table 97: pg_catalog.gp_distributed_xacts

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>distributed_xid</td>
<td>xid</td>
<td></td>
<td>The transaction ID used by the distributed transaction across the Greenplum Database array.</td>
</tr>
<tr>
<td>distributed_id</td>
<td>text</td>
<td></td>
<td>The distributed transaction identifier. It has 2 parts — a unique timestamp and the distributed transaction number.</td>
</tr>
<tr>
<td>state</td>
<td>text</td>
<td></td>
<td>The current state of this session with regards to distributed transactions.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>gp_session_id</td>
<td>int</td>
<td></td>
<td>The ID number of the Greenplum Database session associated with this transaction.</td>
</tr>
<tr>
<td>xmin_distributed_ snapshot</td>
<td>xid</td>
<td></td>
<td>The minimum distributed transaction number found among all open transactions when this transaction was started. It is used for MVCC distributed snapshot purposes.</td>
</tr>
</tbody>
</table>

### gp_distribution_policy

The `gp_distribution_policy` table contains information about Greenplum Database tables and their policy for distributing table data across the segments. This table is populated only on the master. This table is not globally shared, meaning each database has its own copy of this table.

Table 98: `pg_catalog.gp_distribution_policy`

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>localoid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The table object identifier (OID).</td>
</tr>
<tr>
<td>attrnums</td>
<td>smallint[]</td>
<td>pg_attribute.attnum</td>
<td>The column number(s) of the distribution column(s).</td>
</tr>
</tbody>
</table>

### gpexpand.expansion_progress

The `gpexpand.expansion_progress` view contains information about the status of a system expansion operation. The view provides calculations of the estimated rate of table redistribution and estimated time to completion.

Status for specific tables involved in the expansion is stored in `gpexpand.status_detail`.

Table 99: `gpexpand.expansion_progress`

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>text</td>
<td></td>
<td>Name for the data field provided. Includes: Bytes Left, Bytes Done, Estimated Expansion Rate, Estimated Time to Completion, Tables Expanded, Tables Left</td>
</tr>
</tbody>
</table>
The `gpexpand.status` table contains information about the status of a system expansion operation. Status for specific tables involved in the expansion is stored in `gpexpand.status_detail`.

In a normal expansion operation it is not necessary to modify the data stored in this table.

Table 100: `gpexpand.status`  
<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>text</td>
<td></td>
<td>The value for the progress data. For example: Estimated Expansion Rate - 9.75667095996092 MB/s</td>
</tr>
</tbody>
</table>

The `gpexpand.status_detail` table contains information about the status of tables involved in a system expansion operation. You can query this table to determine the status of tables being expanded, or to view the start and end time for completed tables.

This table also stores related information about the table such as the oid, disk size, and normal distribution policy and key. Overall status information for the expansion is stored in `gpexpand.status`.

In a normal expansion operation it is not necessary to modify the data stored in this table.

Table 101: `gpexpand.status_detail`  
<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbname</td>
<td>text</td>
<td></td>
<td>Name of the database to which the table belongs.</td>
</tr>
<tr>
<td>fq_name</td>
<td>text</td>
<td></td>
<td>Fully qualified name of the table.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>schema_oid</td>
<td>oid</td>
<td></td>
<td>OID for the schema of the database to which the table belongs.</td>
</tr>
<tr>
<td>table_oid</td>
<td>oid</td>
<td></td>
<td>OID of the table.</td>
</tr>
<tr>
<td>distribution_policy</td>
<td>smallint()</td>
<td></td>
<td>Array of column IDs for the distribution key of the table.</td>
</tr>
<tr>
<td>distribution_policy_names</td>
<td>text</td>
<td></td>
<td>Column names for the hash distribution key.</td>
</tr>
<tr>
<td>distribution_policy_coloIds</td>
<td>text</td>
<td></td>
<td>Column IDs for the distribution keys of the table.</td>
</tr>
<tr>
<td>storage_options</td>
<td>text</td>
<td></td>
<td>Not enabled in this release. Do not update this field.</td>
</tr>
<tr>
<td>rank</td>
<td>int</td>
<td></td>
<td>Rank determines the order in which tables are expanded. The expansion utility will sort on rank and expand the lowest-ranking tables first.</td>
</tr>
<tr>
<td>status</td>
<td>text</td>
<td></td>
<td>Status of expansion for this table. Valid values are: NOT_STARTED IN PROGRESS FINISHED NO LONGER EXISTS</td>
</tr>
<tr>
<td>last updated</td>
<td>timestamp with timezone</td>
<td></td>
<td>Timestamp of the last change in status for this table.</td>
</tr>
<tr>
<td>expansion started</td>
<td>timestamp with timezone</td>
<td></td>
<td>Timestamp for the start of the expansion of this table. This field is only populated after a table is successfully expanded.</td>
</tr>
<tr>
<td>expansion finished</td>
<td>timestamp with timezone</td>
<td></td>
<td>Timestamp for the completion of expansion of this table.</td>
</tr>
</tbody>
</table>
**source bytes**

The size of disk space associated with the source table. Due to table bloat in heap tables and differing numbers of segments after expansion, it is not expected that the final number of bytes will equal the source number. This information is tracked to help provide progress measurement to aid in duration estimation for the end-to-end expansion operation.

---

**gp_fastsequence**

The `gp_fastsequence` table contains information about append-optimized and column-oriented tables. The `last_sequence` value indicates maximum row number currently used by the table.

**Table 102: pg_catalog.gp_fastsequence**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>Object id of the <code>pg_aoseg.pg_aocsseg_*</code> table used to track append-optimized file segments.</td>
</tr>
<tr>
<td>objmod</td>
<td>bigint</td>
<td></td>
<td>Object modifier.</td>
</tr>
<tr>
<td>last_sequence</td>
<td>bigint</td>
<td></td>
<td>The last sequence number used by the object.</td>
</tr>
</tbody>
</table>

**gp_fault_strategy**

The `gp_fault_strategy` table specifies the fault action.

**Table 103: pg_catalog.gp_fault_strategy**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fault_strategy</td>
<td>char</td>
<td></td>
<td>The mirror failover action to take when a segment failure occurs:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>n</code> = nothing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>f</code> = file-based failover</td>
</tr>
</tbody>
</table>
gp_global_sequence

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>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence_num</td>
<td>bigint</td>
<td></td>
<td>Log sequence number position in the transaction log</td>
</tr>
</tbody>
</table>

gp_id

The `gp_id` system catalog table identifies the Greenplum Database system name and number of segments for the system. It also has local values for the particular database instance (segment or master) on which the table resides. This table is defined in the `pg_global` tablespace, meaning it is globally shared across all databases in the system.

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpname</td>
<td>name</td>
<td></td>
<td>The name of this Greenplum Database system.</td>
</tr>
<tr>
<td>numsegments</td>
<td>integer</td>
<td></td>
<td>The number of segments in the Greenplum Database system.</td>
</tr>
<tr>
<td>dbid</td>
<td>integer</td>
<td></td>
<td>The unique identifier of this segment (or master) instance.</td>
</tr>
<tr>
<td>content</td>
<td>integer</td>
<td></td>
<td>The ID for the portion of data on this segment instance. A primary and its mirror will have the same content ID. For a segment the value is from 0-N-1, where N is the number of segments in Greenplum Database. For the master, the value is -1.</td>
</tr>
</tbody>
</table>

gp_persistent_database_node

The `gp_persistent_database_node` table keeps track of the status of file system objects in relation to the transaction status of database objects. This information is used to make sure the state of the system catalogs and the file system files persisted to disk are synchronized. This information is used by the primary to mirror file replication process.
Table 106: pg_catalog.gp_persistent_database_node

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_oid</td>
<td>oid</td>
<td>pg_tablespace.oid</td>
<td>Table space object id.</td>
</tr>
<tr>
<td>database_oid</td>
<td>oid</td>
<td>pg_database.oid</td>
<td>Database object id.</td>
</tr>
<tr>
<td>persistent_state</td>
<td>smallint</td>
<td></td>
<td>0 - free</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - create pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 - created</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 - drop pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 - aborting create</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5 - &quot;Just in Time&quot; create pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6 - bulk load create pending</td>
</tr>
<tr>
<td>mirror_existence_state</td>
<td>smallint</td>
<td></td>
<td>0 - none</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - not mirrored</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 - mirror create pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 - mirrorcreated</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 - mirror down before create</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5 - mirror down during create</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6 - mirror drop pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7 - only mirror drop remains</td>
</tr>
<tr>
<td>parent_xid</td>
<td>integer</td>
<td></td>
<td>Global transaction id.</td>
</tr>
<tr>
<td>persistent_serial_num</td>
<td>bigint</td>
<td></td>
<td>Log sequence number position in the transaction log for a file block.</td>
</tr>
<tr>
<td>previous_free_tid</td>
<td>tid</td>
<td></td>
<td>Used by Greenplum Database to internally manage persistent representations of file system objects.</td>
</tr>
</tbody>
</table>

**gp_persistent_filespace_node**

The `gp_persistent_filespace_node` table keeps track of the status of file system objects in relation to the transaction status of filespace objects. This information is used to make sure the state of the system catalogs and the file system files persisted to disk are synchronized. This information is used by the primary to mirror file replication process.
### Table 107: pg_catalog gp_persistent_filespace_node

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filespace_oid</td>
<td>oid</td>
<td>pg_filespace.oid</td>
<td>object id of the filesystem</td>
</tr>
<tr>
<td>db_id_1</td>
<td>smallint</td>
<td></td>
<td>primary segment id</td>
</tr>
<tr>
<td>location_1</td>
<td>text</td>
<td></td>
<td>primary filesystem location</td>
</tr>
<tr>
<td>db_id_2</td>
<td>smallint</td>
<td></td>
<td>mirror segment id</td>
</tr>
<tr>
<td>location_2</td>
<td>text</td>
<td></td>
<td>mirror filesystem location</td>
</tr>
<tr>
<td>persistent_state</td>
<td>smallint</td>
<td></td>
<td>0 - free 1 - create pending 2 - created 3 - drop pending 4 - aborting create</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5 - &quot;Just in Time&quot; create pending 6 - bulk load create pending</td>
</tr>
<tr>
<td>mirror_existence_state</td>
<td>smallint</td>
<td></td>
<td>0 - none 1 - not mirrored 2 - mirror create pending 3 - mirrorcreated 4 - mirror down before create 5 - mirror down during create 6 - mirror drop pending 7 - only mirror drop remains</td>
</tr>
<tr>
<td>parent_xid</td>
<td>integer</td>
<td></td>
<td>Global transaction id</td>
</tr>
<tr>
<td>persistent_serial_num</td>
<td>bigint</td>
<td></td>
<td>Log sequence number position in the transaction log for a file block.</td>
</tr>
<tr>
<td>previous_free_tid</td>
<td>tid</td>
<td></td>
<td>Used by Greenplum Database to internally manage persistent representations of file system objects.</td>
</tr>
</tbody>
</table>
gp_persistent_relation_node

The gp_persistent_relation_node table keeps track of the status of file system objects in relation to the transaction status of relation objects (tables, view, indexes, and so on). This information is used to make sure the state of the system catalogs and the file system files persisted to disk are synchronized. This information is used by the primary to mirror file replication process.

Table 108: pg_catalog.gp_persistent_relation_node

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_oid</td>
<td>oid</td>
<td>pg_tablespace.oid</td>
<td>Tablespace object id</td>
</tr>
<tr>
<td>database_oid</td>
<td>oid</td>
<td>pg_database.oid</td>
<td>Database object id</td>
</tr>
<tr>
<td>relfilenode_oid</td>
<td>oid</td>
<td>pg_class.relfilenode</td>
<td>The object id of the relation file node.</td>
</tr>
<tr>
<td>segment_file_num</td>
<td>integer</td>
<td></td>
<td>For append-optimized tables, the append-optimized segment file number.</td>
</tr>
<tr>
<td>relation_storage_manager</td>
<td>smallint</td>
<td></td>
<td>Whether the relation is heap storage or append-optimized storage.</td>
</tr>
<tr>
<td>persistent_state</td>
<td>smallint</td>
<td></td>
<td>0 - free</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - create pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 - created</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 - drop pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 - aborting create</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5 - &quot;Just in Time&quot; create pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6 - bulk load create pending</td>
</tr>
<tr>
<td>mirror_existence_state</td>
<td>smallint</td>
<td></td>
<td>0 - none</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - not mirrored</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 - mirror create pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 - mirrorcreated</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 - mirror down before create</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5 - mirror down during create</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6 - mirror drop pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7 - only mirror drop remains</td>
</tr>
<tr>
<td>parent_xid</td>
<td>integer</td>
<td></td>
<td>Global transaction id.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>persistent_serial_num</td>
<td>bigint</td>
<td></td>
<td>Log sequence number position in the transaction log for a file block.</td>
</tr>
<tr>
<td>previous_free_tid</td>
<td>tid</td>
<td></td>
<td>Used by Greenplum Database to internally manage persistent representations of file system objects.</td>
</tr>
</tbody>
</table>

**gp_persistent_tablespace_node**

The `gp_persistent_tablespace_node` table keeps track of the status of file system objects in relation to the transaction status of tablespace objects. This information is used to make sure the state of the system catalogs and the file system files persisted to disk are synchronized. This information is used by the primary to mirror file replication process.

Table 109: `pg_catalog.gp_persistent_tablespace_node`

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file space oid</td>
<td>oid</td>
<td>pg_filespace.oid</td>
<td>Filespace object id</td>
</tr>
<tr>
<td>table space oid</td>
<td>oid</td>
<td>pg_tablespace.oid</td>
<td>Tablespace object id</td>
</tr>
<tr>
<td>persistent state</td>
<td>smallint</td>
<td></td>
<td>0 - free</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - create pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 - created</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 - drop pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 - aborting create</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5 - &quot;Just In Time&quot; create pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6 - bulk load create pending</td>
</tr>
<tr>
<td>mirror existence state</td>
<td>smallint</td>
<td></td>
<td>0 - none</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - not mirrored</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 - mirror create pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 - mirror created</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 - mirror down before create</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5 - mirror down during create</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6 - mirror drop pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7 - only mirror drop remains</td>
</tr>
<tr>
<td>parent xid</td>
<td>integer</td>
<td></td>
<td>Global transaction id</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>persistent_serial_num</td>
<td>bigint</td>
<td></td>
<td>Log sequence number position in the transaction log for a file block.</td>
</tr>
<tr>
<td>previous_free_tid</td>
<td>tid</td>
<td></td>
<td>Used by Greenplum Database to internally manage persistent representations of file system objects.</td>
</tr>
</tbody>
</table>

**gp_pgdatabase**

The `gp_pgdatabase` view shows status information about the Greenplum segment instances and whether they are acting as the mirror or the primary. This view is used internally by the Greenplum fault detection and recovery utilities to determine failed segments.

**Table 110: pg_catalog.gp_pgdatabase**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbid</td>
<td>smallint</td>
<td><code>gp_segment_configuration.dbid</code></td>
<td>System-assigned ID. The unique identifier of a segment (or master) instance.</td>
</tr>
<tr>
<td>isprimary</td>
<td>boolean</td>
<td><code>gp_segment_configuration.role</code></td>
<td>Whether or not this instance is active. Is it currently acting as the primary segment (as opposed to the mirror).</td>
</tr>
<tr>
<td>content</td>
<td>smallint</td>
<td><code>gp_segment_configuration.content</code></td>
<td>The ID for the portion of data on an instance. A primary segment instance and its mirror will have the same content ID. For a segment the value is from 0-(N-1), where (N) is the number of segments in Greenplum Database. For the master, the value is -1.</td>
</tr>
<tr>
<td>definedprimary</td>
<td>boolean</td>
<td><code>gp_segment_configuration.preferred_role</code></td>
<td>Whether or not this instance was defined as the primary (as opposed to the mirror) at the time the system was initialized.</td>
</tr>
</tbody>
</table>
**gp_relation_node**

The `gp_relation_node` table contains information about the file system objects for a relation (table, view, index, and so on).

### Table 111: `pg_catalog.gp_relation_node`

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>relfilenode_oid</td>
<td>oid</td>
<td>pg_class.relfilenode</td>
<td>The object id of the relation file node.</td>
</tr>
<tr>
<td>segment_file_num</td>
<td>integer</td>
<td></td>
<td>For append-optimized tables, the append-optimized segment file number.</td>
</tr>
<tr>
<td>persistent_tid</td>
<td>tid</td>
<td></td>
<td>Used by Greenplum Database to internally manage persistent representations of file system objects.</td>
</tr>
<tr>
<td>persistent_serial_num</td>
<td>bigint</td>
<td></td>
<td>Log sequence number position in the transaction log for a file block.</td>
</tr>
</tbody>
</table>

**gp_resgroup_config**

The `gp_toolkit.gp_resgroup_config` view allows administrators to see the current CPU, memory, and concurrency limits for a resource group. The view also displays proposed limit settings. A proposed limit will differ from a current limit when the limit has been altered, but the new value could not be immediately applied.

### Table 112: `gp_toolkit.gp_resgroup_config`

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupid</td>
<td>oid</td>
<td>pg_resgroup.oid</td>
<td>The ID of the resource group.</td>
</tr>
<tr>
<td>groupname</td>
<td>name</td>
<td>pg_resgroup.rsgname</td>
<td>The name of the resource group.</td>
</tr>
<tr>
<td>concurrency</td>
<td>text</td>
<td>pg_resgroupcapability.value for pg_resgroupcapability. reslimittype = 1</td>
<td>The concurrency (CONCURRENCY) value specified for the resource group.</td>
</tr>
<tr>
<td>proposed_concurrency</td>
<td>text</td>
<td>pg_resgroupcapability.proposed for pg_resgroupcapability. reslimittype = 1</td>
<td>The pending concurrency value for the resource group.</td>
</tr>
<tr>
<td>cpu_rate_limit</td>
<td>text</td>
<td>pg_resgroupcapability.value for pg_resgroupcapability. reslimittype = 2</td>
<td>The CPU limit (CPU_RATE_LIMIT) value specified for the resource group.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------</td>
<td>------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>proposed_cpu_rate_limit</td>
<td>text</td>
<td>pg_resgroupcapability. proposed for pg_</td>
<td>The pending CPU limit value for the resource group.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>resgroupcapability. reslimittype = 2</td>
<td></td>
</tr>
<tr>
<td>memory_limit</td>
<td>text</td>
<td>pg_resgroupcapability. value for pg_</td>
<td>The memory limit (MEMORY_LIMIT) value specified for the resource group.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>resgroupcapability. reslimittype = 3</td>
<td></td>
</tr>
<tr>
<td>proposed_memory_limit</td>
<td>text</td>
<td>pg_resgroupcapability. proposed for pg_</td>
<td>The pending memory limit value for the resource group.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>resgroupcapability. reslimittype = 3</td>
<td></td>
</tr>
<tr>
<td>memory_shared_quota</td>
<td>text</td>
<td>pg_resgroupcapability. value for pg_</td>
<td>The shared memory quota (MEMORY_SHARED_QUOTA) value specified for the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>resgroupcapability. reslimittype = 4</td>
<td>resource group.</td>
</tr>
<tr>
<td>proposed_memory_shared_quota</td>
<td>text</td>
<td>pg_resgroupcapability. proposed for pg_</td>
<td>The pending shared memory quota value for the resource group.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>resgroupcapability. reslimittype = 4</td>
<td></td>
</tr>
<tr>
<td>memory_spill_ratio</td>
<td>text</td>
<td>pg_resgroupcapability. value for pg_</td>
<td>The memory spill ratio (MEMORY_SPILL_RATIO) value specified for the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>resgroupcapability. reslimittype = 5</td>
<td>resource group.</td>
</tr>
<tr>
<td>proposed_memory_spill_ratio</td>
<td>text</td>
<td>pg_resgroupcapability. proposed for pg_</td>
<td>The pending memory spill ratio value for the resource group.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>resgroupcapability. reslimittype&gt; = 5</td>
<td></td>
</tr>
</tbody>
</table>

### gp_resgroup_status

The `gp_toolkit.gp_resgroup_status` view allows administrators to see status and activity for a resource group. It shows how many queries are waiting to run and how many queries are currently active in the system for each resource group. The view also displays current memory and CPU usage for the resource group.

#### Table 113: gp_toolkit.gp_resgroup_status

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsgname</td>
<td>name</td>
<td>pg_resgroup.rsgname</td>
<td>The name of the resource group.</td>
</tr>
<tr>
<td>groupid</td>
<td>oid</td>
<td>pg_resgroup.oid</td>
<td>The ID of the resource group.</td>
</tr>
<tr>
<td>num_running</td>
<td>integer</td>
<td></td>
<td>The number of transactions currently executing in the resource group.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>num_queueing</td>
<td>integer</td>
<td></td>
<td>The number of currently queued transactions for the resource group.</td>
</tr>
<tr>
<td>num_queued</td>
<td>integer</td>
<td></td>
<td>The total number of queued transactions for the resource group since the Greenplum Database cluster was last started, excluding the num_queueing.</td>
</tr>
<tr>
<td>num_executed</td>
<td>integer</td>
<td></td>
<td>The total number of executed transactions in the resource group since the Greenplum Database cluster was last started, excluding the num_running.</td>
</tr>
<tr>
<td>total_queue_duration</td>
<td>interval</td>
<td></td>
<td>The total time any transaction was queued since the Greenplum Database cluster was last started.</td>
</tr>
<tr>
<td>cpu_usage</td>
<td>json</td>
<td></td>
<td>The real-time CPU usage of the resource group on each Greenplum Database segment's host.</td>
</tr>
<tr>
<td>memory_usage</td>
<td>json</td>
<td></td>
<td>The real-time memory usage of the resource group on each Greenplum Database segment's host.</td>
</tr>
</tbody>
</table>

**gp_resqueue_status**

The `gp_toolkit.gp_resqueue_status` view allows administrators to see status and activity for a resource queue. It shows how many queries are waiting to run and how many queries are currently active in the system from a particular resource queue.

**Table 114: gp_toolkit.gp_resqueue_status**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queueid</td>
<td>oid</td>
<td>gp_toolkit.gp_resqueue_queueid</td>
<td>The ID of the resource queue.</td>
</tr>
<tr>
<td>rsqname</td>
<td>name</td>
<td>gp_toolkit.gp_resqueue_rsqname</td>
<td>The name of the resource queue.</td>
</tr>
</tbody>
</table>
### rsqcountlimit
- **Type:** real
- **References:** `gp_toolkit.gp_resqueue_rsqcountlimit`
- **Description:** The active query threshold of the resource queue. A value of -1 means no limit.

### rsqcountvalue
- **Type:** real
- **References:** `gp_toolkit.gp_resqueue_rsqcountvalue`
- **Description:** The number of active query slots currently being used in the resource queue.

### rsqcostlimit
- **Type:** real
- **References:** `gp_toolkit.gp_resqueue_rsqcostlimit`
- **Description:** The query cost threshold of the resource queue. A value of -1 means no limit.

### rsqcostvalue
- **Type:** real
- **References:** `gp_toolkit.gp_resqueue_rsqcostvalue`
- **Description:** The total cost of all statements currently in the resource queue.

### rsqmemorylimit
- **Type:** real
- **References:** `gp_toolkit.gp_resqueue_rsqmemorylimit`
- **Description:** The memory limit for the resource queue.

### rsqmemoryvalue
- **Type:** real
- **References:** `gp_toolkit.gp_resqueue_rsqmemoryvalue`
- **Description:** The total memory used by all statements currently in the resource queue.

### rsqwaiters
- **Type:** integer
- **References:** `gp_toolkit.gp_resqueue_rsqwaiter`
- **Description:** The number of statements currently waiting in the resource queue.

### rsqholders
- **Type:** integer
- **References:** `gp_toolkit.gp_resqueue_rsqholders`
- **Description:** The number of statements currently running on the system from this resource queue.

### gp_segment_configuration

The `gp_segment_configuration` table contains information about mirroring and segment configuration.

**Table 115: pg_catalog.gp_segment_configuration**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbid</td>
<td>smallint</td>
<td></td>
<td>The unique identifier of a segment (or master) instance.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>content</td>
<td>smallint</td>
<td></td>
<td>The content identifier for a segment instance. A primary segment instance and its corresponding mirror will always have the same content identifier. For a segment the value is from 0-N-1, where N is the number of primary segments in the system. For the master, the value is always -1.</td>
</tr>
<tr>
<td>role</td>
<td>char</td>
<td></td>
<td>The role that a segment is currently running as. Values are p (primary) or m (mirror).</td>
</tr>
<tr>
<td>preferred_role</td>
<td>char</td>
<td></td>
<td>The role that a segment was originally assigned at initialization time. Values are p (primary) or m (mirror).</td>
</tr>
<tr>
<td>mode</td>
<td>char</td>
<td></td>
<td>The synchronization status of a segment with its mirror copy. Values are s (synchronized), c (change logging), or r (resyncing).</td>
</tr>
<tr>
<td>status</td>
<td>char</td>
<td></td>
<td>The fault status of a segment. Values are u (up) or d (down).</td>
</tr>
<tr>
<td>port</td>
<td>integer</td>
<td></td>
<td>The TCP port the database server listener process is using.</td>
</tr>
<tr>
<td>hostname</td>
<td>text</td>
<td></td>
<td>The hostname of a segment host.</td>
</tr>
<tr>
<td>address</td>
<td>text</td>
<td></td>
<td>The hostname used to access a particular segment on a segment host. This value may be the same as hostname in systems upgraded from 3.x or on systems that do not have per-interface hostnames configured.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------</td>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>replication_port</td>
<td>integer</td>
<td></td>
<td>The TCP port the file block replication process is using to keep primary and mirror segments synchronized.</td>
</tr>
</tbody>
</table>

**gp_transaction_log**

The `gp_transaction_log` view contains status information about transactions local to a particular segment. This view allows you to see the status of local transactions.

**Table 116: pg_catalog.gp_transaction_log**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_id</td>
<td>smallint</td>
<td>gp_segment_configuration.content</td>
<td>The content id if the segment. The master is always -1 (no content).</td>
</tr>
<tr>
<td>dbid</td>
<td>smallint</td>
<td>gp_segment_configuration.dbid</td>
<td>The unique id of the segment instance.</td>
</tr>
<tr>
<td>transaction</td>
<td>xid</td>
<td></td>
<td>The local transaction ID.</td>
</tr>
<tr>
<td>status</td>
<td>text</td>
<td></td>
<td>The status of the local transaction (Commited or Aborted).</td>
</tr>
</tbody>
</table>

**gp_version_at_initdb**

The `gp_version_at_initdb` table is populated on the master and each segment in the Greenplum Database system. It identifies the version of Greenplum Database used when the system was first initialized. This table is defined in the `pg_global` tablespace, meaning it is globally shared across all databases in the system.

**Table 117: pg_catalog.gp_version**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaversion</td>
<td>integer</td>
<td></td>
<td>Schema version number.</td>
</tr>
<tr>
<td>productversion</td>
<td>text</td>
<td></td>
<td>Product version number.</td>
</tr>
</tbody>
</table>

**pg_aggregate**

The `pg_aggregate` table stores information about aggregate functions. An aggregate function is a function that operates on a set of values (typically one column from each row that matches a query condition) and returns a single value computed from all these values. Typical aggregate functions are `sum`, `count`, and `max`. Each entry in `pg_aggregate` is an extension of an entry in `pg_proc`. The `pg_proc` entry carries the aggregate's name, input and output data types, and other information that is similar to ordinary functions.
Table 118: pg_catalog.pg_aggregate

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggfnoid</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Aggregate function OID</td>
</tr>
<tr>
<td>aggtransfn</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Transition function OID</td>
</tr>
<tr>
<td>aggpelimfn</td>
<td>regproc</td>
<td></td>
<td>Preliminary function OID (zero if none)</td>
</tr>
<tr>
<td>aggfinalfn</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Final function OID (zero if none)</td>
</tr>
<tr>
<td>agginitval</td>
<td>text</td>
<td></td>
<td>The initial value of the transition state. This is a text field containing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the initial value in its external string representation. If this field is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NULL, the transition state value starts out NULL</td>
</tr>
<tr>
<td>agginvtransfn</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>The OID in pg_proc of the inverse function of aggtransfn</td>
</tr>
<tr>
<td>agginvpelimfn</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>The OID in pg_proc of the inverse function of aggpelimfn</td>
</tr>
<tr>
<td>aggordered</td>
<td>Boolean</td>
<td></td>
<td>If true, the aggregate is defined as ORDERED.</td>
</tr>
<tr>
<td>aggsortop</td>
<td>oid</td>
<td>pg_operator.oid</td>
<td>Associated sort operator OID (zero if none)</td>
</tr>
<tr>
<td>aggtranstype</td>
<td>oid</td>
<td>pg_type.oid</td>
<td>Data type of the aggregate function’s internal transition (state) data</td>
</tr>
</tbody>
</table>

pg_am

The pg_am table stores information about index access methods. There is one row for each index access method supported by the system.

Table 119: pg_catalog.pg_am

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amname</td>
<td>name</td>
<td></td>
<td>Name of the access method</td>
</tr>
<tr>
<td>amstrategies</td>
<td>int2</td>
<td></td>
<td>Number of operator strategies for this access method</td>
</tr>
<tr>
<td>amsupport</td>
<td>int2</td>
<td></td>
<td>Number of support routines for this access method</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>------------</td>
<td>----------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>amorderstrategy</td>
<td>int2</td>
<td></td>
<td>Zero if the index offers no sort order, otherwise the strategy number of the strategy operator that describes the sort order</td>
</tr>
<tr>
<td>amcanunique</td>
<td>boolean</td>
<td></td>
<td>Does the access method support unique indexes?</td>
</tr>
<tr>
<td>amcanmulticol</td>
<td>boolean</td>
<td></td>
<td>Does the access method support multicolumn indexes?</td>
</tr>
<tr>
<td>amoptionalkey</td>
<td>boolean</td>
<td></td>
<td>Does the access method support a scan without any constraint for the first index column?</td>
</tr>
<tr>
<td>amindexnulls</td>
<td>boolean</td>
<td></td>
<td>Does the access method support null index entries?</td>
</tr>
<tr>
<td>amstorage</td>
<td>boolean</td>
<td></td>
<td>Can index storage data type differ from column data type?</td>
</tr>
<tr>
<td>amclusterable</td>
<td>boolean</td>
<td></td>
<td>Can an index of this type be clustered on?</td>
</tr>
<tr>
<td>aminsert</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>&quot;Insert this tuple&quot; function</td>
</tr>
<tr>
<td>ambeginscan</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>&quot;Start new scan&quot; function</td>
</tr>
<tr>
<td>amgettupple</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>&quot;Next valid tuple&quot; function</td>
</tr>
<tr>
<td>amgetmulti</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>&quot;Fetch multiple tuples&quot; function</td>
</tr>
<tr>
<td>amrescan</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>&quot;Restart this scan&quot; function</td>
</tr>
<tr>
<td>amendscan</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>&quot;End this scan&quot; function</td>
</tr>
<tr>
<td>ammarkpos</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>&quot;Mark current scan position&quot; function</td>
</tr>
<tr>
<td>amrestrpos</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>&quot;Restore marked scan position&quot; function</td>
</tr>
<tr>
<td>ambuild</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>&quot;Build new index&quot; function</td>
</tr>
<tr>
<td>ambulkdelete</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Bulk-delete function</td>
</tr>
<tr>
<td>amvacuumcleanup</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Post-VACUUM cleanup function</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>amcostestimate</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Function to estimate cost of an index scan</td>
</tr>
<tr>
<td>amoptions</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Function to parse and validate reloptions for an index</td>
</tr>
</tbody>
</table>

### pg_amop

The `pg_amop` table stores information about operators associated with index access method operator classes. There is one row for each operator that is a member of an operator class.

**Table 120: pg_catalog.pg_amop**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amopclaid</td>
<td>oid</td>
<td>pg_opclass.oid</td>
<td>The index operator class this entry is for</td>
</tr>
<tr>
<td>amopsubtype</td>
<td>oid</td>
<td>pg_type.oid</td>
<td>Subtype to distinguish multiple entries for one strategy; zero for default</td>
</tr>
<tr>
<td>amopstrategy</td>
<td>int2</td>
<td></td>
<td>Operator strategy number</td>
</tr>
<tr>
<td>amopreqcheck</td>
<td>boolean</td>
<td></td>
<td>Index hit must be rechecked</td>
</tr>
<tr>
<td>amopopr</td>
<td>oid</td>
<td>pg_operator.oid</td>
<td>OID of the operator</td>
</tr>
</tbody>
</table>

### pg_amproc

The `pg_amproc` table stores information about support procedures associated with index access method operator classes. There is one row for each support procedure belonging to an operator class.

**Table 121: pg_catalog.pg_amproc**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amopclaid</td>
<td>oid</td>
<td>pg_opclass.oid</td>
<td>The index operator class this entry is for</td>
</tr>
<tr>
<td>amprocssubtype</td>
<td>oid</td>
<td>pg_type.oid</td>
<td>Subtype, if cross-type routine, else zero</td>
</tr>
<tr>
<td>amprocnun</td>
<td>int2</td>
<td></td>
<td>Support procedure number</td>
</tr>
<tr>
<td>amproc</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>OID of the procedure</td>
</tr>
</tbody>
</table>

### pg_appendonly

The `pg_appendonly` table contains information about the storage options and other characteristics of append-optimized tables.
### Table 122: pg_catalog.pg_appendonly

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>relid</td>
<td>oid</td>
<td></td>
<td>The table object identifier (OID) of the compressed table.</td>
</tr>
<tr>
<td>blocksize</td>
<td>integer</td>
<td></td>
<td>Block size used for compression of append-optimized tables. Valid values are 8K - 2M. Default is 32K.</td>
</tr>
<tr>
<td>safefswritesize</td>
<td>integer</td>
<td></td>
<td>Minimum size for safe write operations to append-optimized tables in a non-mature file system. Commonly set to a multiple of the extent size of the file system; for example, Linux ext3 is 4096 bytes, so a value of 32768 is commonly used.</td>
</tr>
<tr>
<td>compresslevel</td>
<td>smallint</td>
<td></td>
<td>The compression level, with compression ratio increasing from 1 to 9. When quicklz is specified for compressstype, valid values are 1 or 3. With zlib specified, valid values are 1-9.</td>
</tr>
<tr>
<td>majorversion</td>
<td>smallint</td>
<td></td>
<td>The major version number of the pg_appendonly table.</td>
</tr>
<tr>
<td>minorversion</td>
<td>smallint</td>
<td></td>
<td>The minor version number of the pg_appendonly table.</td>
</tr>
<tr>
<td>checksum</td>
<td>boolean</td>
<td></td>
<td>A checksum value that is stored to compare the state of a block of data at compression time and at scan time to ensure data integrity.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>compressype</td>
<td>text</td>
<td></td>
<td>Type of compression used to compress append-optimized tables. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• zlib (gzip compression)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• quicklz(^1)</td>
</tr>
<tr>
<td>columnstore</td>
<td>boolean</td>
<td></td>
<td>1 for column-oriented storage, 0 for row-oriented storage.</td>
</tr>
<tr>
<td>segrelid</td>
<td>oid</td>
<td></td>
<td>Table on-disk segment file id.</td>
</tr>
<tr>
<td>segidxid</td>
<td>oid</td>
<td></td>
<td>Index on-disk segment file id.</td>
</tr>
<tr>
<td>blkdirrelid</td>
<td>oid</td>
<td></td>
<td>Block used for on-disk column-oriented table file.</td>
</tr>
<tr>
<td>blkdiridxid</td>
<td>oid</td>
<td></td>
<td>Block used for on-disk column-oriented index file.</td>
</tr>
<tr>
<td>visimaprelid</td>
<td>oid</td>
<td></td>
<td>Visibility map for the table.</td>
</tr>
<tr>
<td>visimapidxid</td>
<td>oid</td>
<td></td>
<td>B-tree index on the visibility map.</td>
</tr>
</tbody>
</table>

\(^1\)QuickLZ compression is available only in the commercial release of Pivotal Greenplum Database.

**pg_attrdef**

The `pg_attrdef` table stores column default values. The main information about columns is stored in `pg_attribute`. Only columns that explicitly specify a default value (when the table is created or the column is added) will have an entry here.

**Table 123: pg_catalog.pg_attrdef**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adrelid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The table this column belongs to</td>
</tr>
<tr>
<td>adnum</td>
<td>int2</td>
<td>pg_attribute.attnum</td>
<td>The number of the column</td>
</tr>
<tr>
<td>adbin</td>
<td>text</td>
<td></td>
<td>The internal representation of the column default value</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>adsrc</td>
<td>text</td>
<td></td>
<td>A human-readable representation of the default value. This field is historical, and is best not used.</td>
</tr>
</tbody>
</table>

**pg_attribute**

The `pg_attribute` table stores information about table columns. There will be exactly one `pg_attribute` row for every column in every table in the database. (There will also be attribute entries for indexes, and all objects that have `pg_class` entries.) The term attribute is equivalent to column.

**Table 124: pg_catalog.pg_attribute**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrelid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The table this column belongs to</td>
</tr>
<tr>
<td>attname</td>
<td>name</td>
<td></td>
<td>The column name</td>
</tr>
<tr>
<td>atttypid</td>
<td>oid</td>
<td>pg_type.oid</td>
<td>The data type of this column</td>
</tr>
<tr>
<td>attstat</td>
<td>int4</td>
<td></td>
<td>Controls the level of detail of statistics accumulated for this column by ANALYZE. A zero value indicates that no statistics should be collected. A negative value says to use the system default statistics target. The exact meaning of positive values is data type-dependent. For scalar data types, it is both the target number of &quot;most common values&quot; to collect, and the target number of histogram bins to create.</td>
</tr>
<tr>
<td>attlen</td>
<td>int2</td>
<td></td>
<td>A copy of pg_type.typlen of this column’s type.</td>
</tr>
<tr>
<td>attnum</td>
<td>int2</td>
<td></td>
<td>The number of the column. Ordinary columns are numbered from 1 up. System columns, such as oid, have (arbitrary) negative numbers.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>attndims</td>
<td>int4</td>
<td></td>
<td>Number of dimensions, if the column is an array type; otherwise 0. (Presently, the number of dimensions of an array is not enforced, so any nonzero value effectively means it is an array)</td>
</tr>
<tr>
<td>attcacheoff</td>
<td>int4</td>
<td></td>
<td>Always (-1) in storage, but when loaded into a row descriptor in memory this may be updated to cache the offset of the attribute within the row</td>
</tr>
<tr>
<td>atttypmod</td>
<td>int4</td>
<td></td>
<td>Records type-specific data supplied at table creation time (for example, the maximum length of a varchar column). It is passed to type-specific input functions and length coercion functions. The value will generally be (-1) for types that do not need it.</td>
</tr>
<tr>
<td>attbyval</td>
<td>boolean</td>
<td></td>
<td>A copy of pg_type. typbyval of this column's type</td>
</tr>
<tr>
<td>attstorage</td>
<td>char</td>
<td></td>
<td>Normally a copy of pg_type.typstorage of this column's type. For TOAST-able data types, this can be altered after column creation to control storage policy.</td>
</tr>
<tr>
<td>attalign</td>
<td>char</td>
<td></td>
<td>A copy of pg_type. typalign of this column's type</td>
</tr>
<tr>
<td>attnotnull</td>
<td>boolean</td>
<td></td>
<td>This represents a not-null constraint. It is possible to change this column to enable or disable the constraint.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>atthasdef</td>
<td>boolean</td>
<td></td>
<td>This column has a default value, in which case there will be a corresponding entry in the pg_attrdef catalog that actually defines the value</td>
</tr>
<tr>
<td>attisdropped</td>
<td>boolean</td>
<td></td>
<td>This column has been dropped and is no longer valid. A dropped column is still physically present in the table, but is ignored by the parser and so cannot be accessed via SQL</td>
</tr>
<tr>
<td>attislocal</td>
<td>boolean</td>
<td></td>
<td>This column is defined locally in the relation. Note that a column may be locally defined and inherited simultaneously</td>
</tr>
<tr>
<td>attinhcount</td>
<td>int4</td>
<td></td>
<td>The number of direct ancestors this column has. A column with a nonzero number of ancestors cannot be dropped nor renamed</td>
</tr>
</tbody>
</table>

### pg_attribute_encoding

The pg_attribute_encoding system catalog table contains column storage information.

**Table 125: pg_catalog.pg_attribute_encoding**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>modifiers</th>
<th>storage</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrelid</td>
<td>oid</td>
<td>not null</td>
<td>plain</td>
<td>Foreign key to pg_attribute.attrelid</td>
</tr>
<tr>
<td>attnum</td>
<td>smallint</td>
<td>not null</td>
<td>plain</td>
<td>Foreign key to pg_attribute.attnum</td>
</tr>
<tr>
<td>attoptions</td>
<td>text []</td>
<td></td>
<td>extended</td>
<td>The options</td>
</tr>
</tbody>
</table>

### pg_auth_members

The pg_auth_members system catalog table shows the membership relations between roles. Any non-circular set of relationships is allowed. Because roles are system-wide, pg_auth_members is shared across all databases of a Greenplum Database system.
Table 126: pg_catalog.pg_auth_members

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>roleid</td>
<td>oid</td>
<td>pg_authid.oid</td>
<td>ID of the parent-level (group) role</td>
</tr>
<tr>
<td>member</td>
<td>oid</td>
<td>pg_authid.oid</td>
<td>ID of a member role</td>
</tr>
<tr>
<td>grantor</td>
<td>oid</td>
<td>pg_authid.oid</td>
<td>ID of the role that granted this membership</td>
</tr>
<tr>
<td>admin_option</td>
<td>boolean</td>
<td></td>
<td>True if role member may grant membership to others</td>
</tr>
</tbody>
</table>

pg_authid

The pg_authid table contains information about database authorization identifiers (roles). A role subsumes the concepts of users and groups. A user is a role with the rolcanlogin flag set. Any role (with or without rolcanlogin) may have other roles as members. See pg_authid_members.

Since this catalog contains passwords, it must not be publicly readable. pg_roles is a publicly readable view on pg_authid that blanks out the password field.

Because user identities are system-wide, pg_authid is shared across all databases in a Greenplum Database system: there is only one copy of pg_authid per system, not one per database.

Table 127: pg_catalog.pg_authid

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rolname</td>
<td>name</td>
<td></td>
<td>Role name</td>
</tr>
<tr>
<td>rolsuper</td>
<td>boolean</td>
<td></td>
<td>Role has superuser privileges</td>
</tr>
<tr>
<td>rolinherit</td>
<td>boolean</td>
<td></td>
<td>Role automatically inherits privileges of roles it is a member of</td>
</tr>
<tr>
<td>rolcreaterole</td>
<td>boolean</td>
<td></td>
<td>Role may create more roles</td>
</tr>
<tr>
<td>rolcreatedb</td>
<td>boolean</td>
<td></td>
<td>Role may create databases</td>
</tr>
<tr>
<td>rolcatupdate</td>
<td>boolean</td>
<td></td>
<td>Role may update system catalogs directly. (Even a superuser may not do this unless this column is true)</td>
</tr>
<tr>
<td>rolcanlogin</td>
<td>boolean</td>
<td></td>
<td>Role may log in. That is, this role can be given as the initial session authorization identifier</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rolconnlimit</td>
<td>int4</td>
<td></td>
<td>For roles that can log in, this sets maximum number of concurrent connections this role can make. -1 means no limit</td>
</tr>
<tr>
<td>rolpassword</td>
<td>text</td>
<td></td>
<td>Password (possibly encrypted); NULL if none</td>
</tr>
<tr>
<td>rolvaliduntil</td>
<td>timestampz</td>
<td></td>
<td>Password expiry time (only used for password authentication); NULL if no expiration</td>
</tr>
<tr>
<td>rolconfig</td>
<td>text[]</td>
<td></td>
<td>Session defaults for server configuration parameters</td>
</tr>
<tr>
<td>rolresqueue</td>
<td>oid</td>
<td></td>
<td>Object ID of the associated resource queue ID in pg_resqueue</td>
</tr>
<tr>
<td>rolcreaterextgpfd</td>
<td>boolean</td>
<td></td>
<td>Privilege to create read external tables with the gpfdist or gpfdists protocol</td>
</tr>
<tr>
<td>rolcreaterexhttp</td>
<td>boolean</td>
<td></td>
<td>Privilege to create read external tables with the http protocol</td>
</tr>
<tr>
<td>rolcreatetextgpdf</td>
<td>boolean</td>
<td></td>
<td>Privilege to create write external tables with the gpfdist or gpfdists protocol</td>
</tr>
<tr>
<td>rolcreaterexhdfs</td>
<td>boolean</td>
<td></td>
<td>Privilege to create read external tables with the gphdfs protocol</td>
</tr>
<tr>
<td>rolcreaterexhdfs</td>
<td>boolean</td>
<td></td>
<td>Privilege to create write external tables with the gphdfs protocol</td>
</tr>
<tr>
<td>rolresgroup</td>
<td>oid</td>
<td></td>
<td>Object ID of the associated resource group ID in pg_resgroup</td>
</tr>
</tbody>
</table>

**pg_available_extension_versions**

The `pg_available_extension_versions` view lists the specific extension versions that are available for installation. The `pg_extension` system catalog table shows the extensions currently installed. The view is read only.
Table 128: pg_catalog.pg_available_extension_versions

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
<td>Extension name.</td>
</tr>
<tr>
<td>version</td>
<td>text</td>
<td>Version name.</td>
</tr>
<tr>
<td>installed</td>
<td>boolean</td>
<td>True if this version of this extension is currently installed, False otherwise.</td>
</tr>
<tr>
<td>superuser</td>
<td>boolean</td>
<td>True if only superusers are allowed to install the extension, False otherwise.</td>
</tr>
<tr>
<td>relocatable</td>
<td>boolean</td>
<td>True if extension can be relocated to another schema, False otherwise.</td>
</tr>
<tr>
<td>schema</td>
<td>name</td>
<td>Name of the schema that the extension must be installed into, or NULL if partially or fully relocatable.</td>
</tr>
<tr>
<td>requires</td>
<td>name[]</td>
<td>Names of prerequisite extensions, or NULL if none</td>
</tr>
<tr>
<td>comment</td>
<td>text</td>
<td>Comment string from the extension control file.</td>
</tr>
</tbody>
</table>

pg_available_extensions

The pg_available_extensions view lists the extensions that are available for installation. The pg_extension system catalog table shows the extensions currently installed.

The view is read only.

Table 129: pg_catalog.pg_available_extensions

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
<td>Extension name.</td>
</tr>
<tr>
<td>default_version</td>
<td>text</td>
<td>Name of default version, or NULL if none is specified.</td>
</tr>
<tr>
<td>installed_version</td>
<td>text</td>
<td>Currently installed version of the extension, or NULL if not installed.</td>
</tr>
<tr>
<td>comment</td>
<td>text</td>
<td>Comment string from the extension control file.</td>
</tr>
</tbody>
</table>

pg_cast

The pg_cast table stores data type conversion paths, both built-in paths and those defined with CREATE CAST. The cast functions listed in pg_cast must always take the cast source type as their first argument type, and return the cast destination type as their result type. A cast function can have up to three arguments. The second argument, if present, must be type integer; it receives the type modifier associated with the destination type, or -1 if there is none. The third argument, if present, must be type boolean; it receives true if the cast is an explicit cast, false otherwise.
It is legitimate to create a `pg_cast` entry in which the source and target types are the same, if the associated function takes more than one argument. Such entries represent 'length coercion functions' that coerce values of the type to be legal for a particular type modifier value. Note however that at present there is no support for associating non-default type modifiers with user-created data types, and so this facility is only of use for the small number of built-in types that have type modifier syntax built into the grammar.

When a `pg_cast` entry has different source and target types and a function that takes more than one argument, it represents converting from one type to another and applying a length coercion in a single step. When no such entry is available, coercion to a type that uses a type modifier involves two steps, one to convert between data types and a second to apply the modifier.

Table 130: `pg_catalog.pg_cast`

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>castsource</td>
<td>oid</td>
<td>pg_type.oid</td>
<td>OID of the source data type.</td>
</tr>
<tr>
<td>casttarget</td>
<td>oid</td>
<td>pg_type.oid</td>
<td>OID of the target data type.</td>
</tr>
<tr>
<td>castfunc</td>
<td>oid</td>
<td>pg_proc.oid</td>
<td>The OID of the function to use to perform this cast. Zero is stored if the data types are binary compatible (that is, no run-time operation is needed to perform the cast).</td>
</tr>
<tr>
<td>castcontext</td>
<td>char</td>
<td></td>
<td>Indicates what contexts the cast may be invoked in. e means only as an explicit cast (using <code>CAST</code> or <code>::</code> syntax). a means implicitly in assignment to a target column, as well as explicitly. i means implicitly in expressions, as well as the other cases.</td>
</tr>
</tbody>
</table>

**pg_class**

The system catalog table `pg_class` catalogs tables and most everything else that has columns or is otherwise similar to a table (also known as `relations`). This includes indexes (see also `pg_index`), sequences, views, composite types, and TOAST tables. Not all columns are meaningful for all relation types.

Table 131: `pg_catalog.pg_class`

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>relname</td>
<td>name</td>
<td></td>
<td>Name of the table, index, view, etc.</td>
</tr>
<tr>
<td>relnamespace</td>
<td>oid</td>
<td>pg_namespace.oid</td>
<td>The OID of the namespace (schema) that contains this relation</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>reltype</td>
<td>oid</td>
<td>pg_type.oid</td>
<td>The OID of the data type that corresponds to this table's row type, if any</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(zero for indexes, which have no pg_type entry)</td>
</tr>
<tr>
<td>relowner</td>
<td>oid</td>
<td>pg_authid.oid</td>
<td>Owner of the relation</td>
</tr>
<tr>
<td>relam</td>
<td>oid</td>
<td>pg_am.oid</td>
<td>If this is an index, the access method used (B-tree, Bitmap, hash, etc.)</td>
</tr>
<tr>
<td>relfilenode</td>
<td>oid</td>
<td></td>
<td>Name of the on-disk file of this relation; 0 if none.</td>
</tr>
<tr>
<td>reltablespace</td>
<td>oid</td>
<td>pg_tablespace.oid</td>
<td>The tablespace in which this relation is stored. If zero, the database's</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>default tablespace is implied. (Not meaningful if the relation has no on-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>disk file.)</td>
</tr>
<tr>
<td>relpages</td>
<td>int4</td>
<td></td>
<td>Size of the on-disk representation of this table in pages (of 32K each).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This is only an estimate used by the planner. It is updated by VACUUM,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ANALYZE, and a few DDL commands.</td>
</tr>
<tr>
<td>reltuples</td>
<td>float4</td>
<td></td>
<td>Number of rows in the table. This is only an estimate used by the planner.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>It is updated by VACUUM, ANALYZE, and a few DDL commands.</td>
</tr>
<tr>
<td>reltoastrelid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>OID of the TOAST table associated with this table, 0 if none. The TOAST</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>table stores large attributes &quot;out of line&quot; in a secondary table.</td>
</tr>
<tr>
<td>reltoastidxid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>For a TOAST table, the OID of its index. 0 if not a TOAST table.</td>
</tr>
<tr>
<td>relasegidxid</td>
<td>oid</td>
<td></td>
<td>Deprecated in Greenplum Database 3. 4.</td>
</tr>
<tr>
<td>relasegrelid</td>
<td>oid</td>
<td></td>
<td>Deprecated in Greenplum Database 3. 4.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>relhasindex</td>
<td>boolean</td>
<td></td>
<td>True if this is a table and it has (or recently had) any indexes. This is set by CREATE INDEX, but not cleared immediately by DROP INDEX. VACUUM will clear if it finds the table has no indexes.</td>
</tr>
<tr>
<td>relisshared</td>
<td>boolean</td>
<td></td>
<td>True if this table is shared across all databases in the system. Only certain system catalog tables are shared.</td>
</tr>
<tr>
<td>relkind</td>
<td>char</td>
<td></td>
<td>The type of object: ( r = ) heap or append-optimized table, ( i = ) index, ( S = ) sequence, ( v = ) view, ( c = ) composite type, ( t = ) TOAST value, ( o = ) internal append-optimized segment files and EOFs, ( c = ) composite type, ( u = ) uncataloged temporary heap table</td>
</tr>
<tr>
<td>relstorage</td>
<td>char</td>
<td></td>
<td>The storage mode of a table: ( a = ) append-optimized, ( c = ) column-oriented, ( h = ) heap, ( v = ) virtual, ( x = ) external table.</td>
</tr>
<tr>
<td>relnatts</td>
<td>int2</td>
<td></td>
<td>Number of user columns in the relation (system columns not counted). There must be this many corresponding entries in pg_attribute.</td>
</tr>
<tr>
<td>relchecks</td>
<td>int2</td>
<td></td>
<td>Number of check constraints on the table.</td>
</tr>
<tr>
<td>reltriggers</td>
<td>int2</td>
<td></td>
<td>Number of triggers on the table.</td>
</tr>
<tr>
<td>relukeys</td>
<td>int2</td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td>reloffkeys</td>
<td>int2</td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td>relrefs</td>
<td>int2</td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>relhasoids</td>
<td>boolean</td>
<td></td>
<td>True if an OID is generated for each row of the relation.</td>
</tr>
<tr>
<td>relhaspkey</td>
<td>boolean</td>
<td></td>
<td>True if the table has (or once had) a primary key.</td>
</tr>
<tr>
<td>relhasrules</td>
<td>boolean</td>
<td></td>
<td>True if table has rules.</td>
</tr>
<tr>
<td>relhassubclass</td>
<td>boolean</td>
<td></td>
<td>True if table has (or once had) any inheritance children.</td>
</tr>
<tr>
<td>relfrozenxid</td>
<td>xid</td>
<td></td>
<td>All transaction IDs before this one have been replaced with a permanent (frozen) transaction ID in this table. This is used to track whether the table needs to be vacuumed in order to prevent transaction ID wraparound or to allow pg_clog to be shrunk. Zero (InvalidTransactionId) if the relation is not a table.</td>
</tr>
<tr>
<td>relacl</td>
<td>aclitem[]</td>
<td></td>
<td>Access privileges assigned by GRANT and REVOKE.</td>
</tr>
<tr>
<td>reloptions</td>
<td>text[]</td>
<td></td>
<td>Access-method-specific options, as “keyword=value” strings.</td>
</tr>
</tbody>
</table>

**pg_compression**

The `pg_compression` system catalog table describes the compression methods available.

**Table 132: pg_catalog.pg_compression**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>modifiers</th>
<th>storage</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compname</td>
<td>name</td>
<td>not null</td>
<td>plain</td>
<td>Name of the compression</td>
</tr>
<tr>
<td>compconstructor</td>
<td>regproc</td>
<td>not null</td>
<td>plain</td>
<td>Name of compression constructor</td>
</tr>
<tr>
<td>compdestructor</td>
<td>regproc</td>
<td>not null</td>
<td>plain</td>
<td>Name of compression destructor</td>
</tr>
</tbody>
</table>
### compcompressor

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>modifiers</th>
<th>storage</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compcompressor</td>
<td>regproc</td>
<td>not null</td>
<td>plain</td>
<td>Name of the compressor</td>
</tr>
</tbody>
</table>

### compdecompressor

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>modifiers</th>
<th>storage</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compdecompressor</td>
<td>regproc</td>
<td>not null</td>
<td>plain</td>
<td>Name of the decompressor</td>
</tr>
</tbody>
</table>

### compvalidator

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>modifiers</th>
<th>storage</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compvalidator</td>
<td>regproc</td>
<td>not null</td>
<td>plain</td>
<td>Name of the compression validator</td>
</tr>
</tbody>
</table>

### compowner

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>modifiers</th>
<th>storage</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compowner</td>
<td>oid</td>
<td>not null</td>
<td>plain</td>
<td>oid from pg_authid</td>
</tr>
</tbody>
</table>

**pg_constraint**

The `pg_constraint` system catalog table stores check, primary key, unique, and foreign key constraints on tables. Column constraints are not treated specially. Every column constraint is equivalent to some table constraint. Not-null constraints are represented in the `pg_attribute` catalog table. Check constraints on domains are stored here, too.

**Table 133: pg_catalog.pg_constraint**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conname</td>
<td>name</td>
<td></td>
<td>Constraint name (not necessarily unique)</td>
</tr>
<tr>
<td>connamespace</td>
<td>oid</td>
<td>pg_namespace.oid</td>
<td>The OID of the namespace (schema) that contains this constraint.</td>
</tr>
<tr>
<td>contype</td>
<td>char</td>
<td></td>
<td>c = check constraint, f = foreign key constraint, p = primary key constraint, u = unique constraint.</td>
</tr>
<tr>
<td>condeferrable</td>
<td>boolean</td>
<td></td>
<td>Is the constraint deferrable?</td>
</tr>
<tr>
<td>condeferred</td>
<td>boolean</td>
<td></td>
<td>Is the constraint deferred by default?</td>
</tr>
<tr>
<td>conrelid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The table this constraint is on; 0 if not a table constraint.</td>
</tr>
<tr>
<td>contypid</td>
<td>oid</td>
<td>pg_type.oid</td>
<td>The domain this constraint is on; 0 if not a domain constraint.</td>
</tr>
<tr>
<td>confrelid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>If a foreign key, the referenced table; else 0.</td>
</tr>
<tr>
<td>confupdtype</td>
<td>char</td>
<td></td>
<td>Foreign key update action code.</td>
</tr>
<tr>
<td>confdeltype</td>
<td>char</td>
<td></td>
<td>Foreign key deletion action code.</td>
</tr>
<tr>
<td>confmatchtype</td>
<td>char</td>
<td></td>
<td>Foreign key match type.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>conkey</td>
<td>int2[]</td>
<td>pg_attribute.attnum</td>
<td>If a table constraint, list of columns which the constraint constrains.</td>
</tr>
<tr>
<td>confkey</td>
<td>int2[]</td>
<td>pg_attribute.attnum</td>
<td>If a foreign key, list of the referenced columns.</td>
</tr>
<tr>
<td>conbin</td>
<td>text</td>
<td></td>
<td>If a check constraint, an internal representation of the expression.</td>
</tr>
<tr>
<td>consrc</td>
<td>text</td>
<td></td>
<td>If a check constraint, a human-readable representation of the expression.</td>
</tr>
</tbody>
</table>

The `pg_conversion` system catalog table describes the available encoding conversion procedures as defined by `CREATE CONVERSION`.

**Table 134: pg_catalog.pg_conversion**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conname</td>
<td>name</td>
<td></td>
<td>Conversion name (unique within a namespace).</td>
</tr>
<tr>
<td>connamespace</td>
<td>oid</td>
<td>pg_namespace.oid</td>
<td>The OID of the namespace (schema) that contains this conversion.</td>
</tr>
<tr>
<td>conowner</td>
<td>oid</td>
<td>pg_authid.oid</td>
<td>Owner of the conversion.</td>
</tr>
<tr>
<td>conforencoding</td>
<td>int4</td>
<td></td>
<td>Source encoding ID.</td>
</tr>
<tr>
<td>contoencoding</td>
<td>int4</td>
<td></td>
<td>Destination encoding ID.</td>
</tr>
<tr>
<td>conproc</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Conversion procedure.</td>
</tr>
<tr>
<td>condefault</td>
<td>boolean</td>
<td></td>
<td>True if this is the default conversion.</td>
</tr>
</tbody>
</table>
**pg_database**

The `pg_database` system catalog table stores information about the available databases. Databases are created with the `CREATE DATABASE` SQL command. Unlike most system catalogs, `pg_database` is shared across all databases in the system. There is only one copy of `pg_database` per system, not one per database.

**Table 135: pg_catalog.pg_database**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>datname</code></td>
<td><code>name</code></td>
<td></td>
<td>Database name.</td>
</tr>
<tr>
<td><code>datdba</code></td>
<td><code>oid</code></td>
<td><code>pg_authid.oid</code></td>
<td>Owner of the database, usually the user who created it.</td>
</tr>
<tr>
<td><code>encoding</code></td>
<td><code>int4</code></td>
<td></td>
<td>Character encoding for this database. <code>pg_encoding_to_char()</code> can translate this number to the encoding name.</td>
</tr>
<tr>
<td><code>datistemplate</code></td>
<td><code>boolean</code></td>
<td></td>
<td>If true then this database can be used in the <code>TEMPLATE</code> clause of <code>CREATE DATABASE</code> to create a new database as a clone of this one.</td>
</tr>
<tr>
<td><code>datallowconn</code></td>
<td><code>boolean</code></td>
<td></td>
<td>If false then no one can connect to this database. This is used to protect the <code>template0</code> database from being altered.</td>
</tr>
<tr>
<td><code>datconnlimit</code></td>
<td><code>int4</code></td>
<td></td>
<td>Sets the maximum number of concurrent connections that can be made to this database. <code>-1</code> means no limit.</td>
</tr>
<tr>
<td><code>datlastsysoid</code></td>
<td><code>oid</code></td>
<td></td>
<td>Last system OID in the database.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>datfrozenxid</td>
<td>xid</td>
<td></td>
<td>All transaction IDs before this one have been replaced with a permanent (frozen) transaction ID in this database. This is used to track whether the database needs to be vacuumed in order to prevent transaction ID wraparound or to allow pg_clog to be shrunk. It is the minimum of the per-table pg_class. relfrozenxid values.</td>
</tr>
<tr>
<td>dattablespace</td>
<td>oid</td>
<td>pg_tablespace.oid</td>
<td>The default tablespace for the database. Within this database, all tables for which pg_class. reltablespace is zero will be stored in this tablespace. All non-shared system catalogs will also be there.</td>
</tr>
<tr>
<td>datconfig</td>
<td>text[]</td>
<td></td>
<td>Session defaults for user-settable server configuration parameters.</td>
</tr>
<tr>
<td>datacl</td>
<td>aclitem[]</td>
<td></td>
<td>Database access privileges as given by GRANT and REVOKE.</td>
</tr>
</tbody>
</table>

**pg_depend**

The pg_depend system catalog table records the dependency relationships between database objects. This information allows DROP commands to find which other objects must be dropped by DROP CASCADE or prevent dropping in the DROP RESTRICT case. See also pg_shdepend, which performs a similar function for dependencies involving objects that are shared across a Greenplum system.

In all cases, a pg_depend entry indicates that the referenced object may not be dropped without also dropping the dependent object. However, there are several subflavors identified by deptype:

- **DEPENDENCY_NORMAL (n)** — A normal relationship between separately-created objects. The dependent object may be dropped without affecting the referenced object. The referenced object may only be dropped by specifying CASCADE, in which case the dependent object is dropped, too. Example: a table column has a normal dependency on its data type.
- **DEPENDENCY_AUTO (a)** — The dependent object can be dropped separately from the referenced object, and should be automatically dropped (regardless of RESTRICT or CASCADE mode) if the referenced object is dropped. Example: a named constraint on a table is made autodependent on the table, so that it will go away if the table is dropped.
- **DEPENDENCY_INTERNAL (i)** — The dependent object was created as part of creation of the referenced object, and is really just a part of its internal implementation. A DROP of the dependent object will be disallowed outright (we'll tell the user to issue a DROP against the referenced object, instead).
A DROP of the referenced object will be propagated through to drop the dependent object whether CASCADE is specified or not.

- **DEPENDENCY_PIN (p)** — There is no dependent object; this type of entry is a signal that the system itself depends on the referenced object, and so that object must never be deleted. Entries of this type are created only by system initialization. The columns for the dependent object contain zeroes.

Table 136: pg_catalog.pg_depend

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>classid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The OID of the system catalog the dependent object is in.</td>
</tr>
<tr>
<td>objid</td>
<td>oid</td>
<td>any OID column</td>
<td>The OID of the specific dependent object.</td>
</tr>
<tr>
<td>objsubid</td>
<td>int4</td>
<td></td>
<td>For a table column, this is the column number. For all other object types, this column is zero.</td>
</tr>
<tr>
<td>refclassid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The OID of the system catalog the referenced object is in.</td>
</tr>
<tr>
<td>refobjid</td>
<td>oid</td>
<td>any OID column</td>
<td>The OID of the specific referenced object.</td>
</tr>
<tr>
<td>refobjsubid</td>
<td>int4</td>
<td></td>
<td>For a table column, this is the referenced column number. For all other object types, this column is zero.</td>
</tr>
<tr>
<td>deptype</td>
<td>char</td>
<td></td>
<td>A code defining the specific semantics of this dependency relationship.</td>
</tr>
</tbody>
</table>

**pg_description**

The pg_description system catalog table stores optional descriptions (comments) for each database object. Descriptions can be manipulated with the `COMMENT` command and viewed with `psql`'s `\d` meta-commands. Descriptions of many built-in system objects are provided in the initial contents of pg_description. See also pg_shdescription, which performs a similar function for descriptions involving objects that are shared across a Greenplum system.

Table 137: pg_catalog.pg_description

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objoid</td>
<td>oid</td>
<td>any OID column</td>
<td>The OID of the object this description pertains to.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>classoid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The OID of the system catalog this object appears in</td>
</tr>
<tr>
<td>objsubid</td>
<td>int4</td>
<td></td>
<td>For a comment on a table column, this is the column number. For all other object types, this column is zero.</td>
</tr>
<tr>
<td>description</td>
<td>text</td>
<td></td>
<td>Arbitrary text that serves as the description of this object.</td>
</tr>
</tbody>
</table>

**pg_enum**

The `pg_enum` table contains entries matching enum types to their associated values and labels. The internal representation of a given enum value is actually the OID of its associated row in `pg_enum`. The OIDs for a particular enum type are guaranteed to be ordered in the way the type should sort, but there is no guarantee about the ordering of OIDs of unrelated enum types.

Table 138: `pg_catalog.pg_enum`

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>References</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enumtypid</td>
<td>oid</td>
<td>pgtype.oid</td>
<td>The OID of the <code>pg_type</code> entry owning this enum value</td>
</tr>
<tr>
<td>enumlabel</td>
<td>name</td>
<td></td>
<td>The textual label for this enum value</td>
</tr>
</tbody>
</table>

**pg_extension**

The system catalog table `pg_extension` stores information about installed extensions.

Table 139: `pg_catalog.pg_extension`

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extname</td>
<td>name</td>
<td></td>
<td>Name of the extension.</td>
</tr>
<tr>
<td>extowner</td>
<td>oid</td>
<td>pg_authid.oid</td>
<td>Owner of the extension</td>
</tr>
<tr>
<td>extnamespace</td>
<td>oid</td>
<td>pg_namespace.oid</td>
<td>Schema containing the extension exported objects.</td>
</tr>
<tr>
<td>extrelocatable</td>
<td>boolean</td>
<td></td>
<td>True if the extension can be relocated to another schema.</td>
</tr>
<tr>
<td>extversion</td>
<td>text</td>
<td></td>
<td>Version name for the extension.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>extconfig</td>
<td>oid[]</td>
<td>pg_class.oid</td>
<td>Array of regclass OIDs for the extension configuration tables, or NULL if none.</td>
</tr>
<tr>
<td>extcondition</td>
<td>text[]</td>
<td></td>
<td>Array of WHERE-clause filter conditions for the extension configuration tables, or NULL if none.</td>
</tr>
</tbody>
</table>

Unlike most catalogs with a namespace column, extnamespace does not imply that the extension belongs to that schema. Extension names are never schema-qualified. The extnamespace schema indicates the schema that contains most or all of the extension objects. If extrelocatable is true, then this schema must contain all schema-qualifiable objects that belong to the extension.

**pg_exttable**

The pg_exttable system catalog table is used to track external tables and web tables created by the CREATE EXTERNAL TABLE command.

Table 140: pg_catalog.pg_exttable

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reloid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The OID of this external table.</td>
</tr>
<tr>
<td>urilocation</td>
<td>text[]</td>
<td></td>
<td>The URI location(s) of the external table files.</td>
</tr>
<tr>
<td>execlocation</td>
<td>text[]</td>
<td></td>
<td>The ON segment locations defined for the external table.</td>
</tr>
<tr>
<td>fmttype</td>
<td>char</td>
<td></td>
<td>Format of the external table files: t for text, or c for csv.</td>
</tr>
<tr>
<td>fmtopts</td>
<td>text</td>
<td></td>
<td>Formatting options of the external table files, such as the field delimiter, null string, escape character, etc.</td>
</tr>
<tr>
<td>options</td>
<td>text[]</td>
<td></td>
<td>The options defined for the external table.</td>
</tr>
<tr>
<td>command</td>
<td>text</td>
<td></td>
<td>The OS command to execute when the external table is accessed.</td>
</tr>
<tr>
<td>rejectlimit</td>
<td>integer</td>
<td></td>
<td>The per segment reject limit for rows with errors, after which the load will fail.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rejectlimittype</td>
<td>char</td>
<td></td>
<td>Type of reject limit threshold: r for number of rows.</td>
</tr>
<tr>
<td>fmterrtbl</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The object id of the error table where format errors will be logged.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>NOTE:</strong> This column is no longer used and will be removed in a future release.</td>
</tr>
<tr>
<td>encoding</td>
<td>text</td>
<td></td>
<td>The client encoding.</td>
</tr>
<tr>
<td>writable</td>
<td>boolean</td>
<td></td>
<td>0 for readable external tables, 1 for writable external tables.</td>
</tr>
</tbody>
</table>

**pg_filespace**

The `pg_filespace` table contains information about the filesystems created in a Greenplum Database system. Every system contains a default filesystem, `pg_system`, which is a collection of all the data directory locations created at system initialization time.

A tablespace requires a file system location to store its database files. In Greenplum Database, the master and each segment (primary and mirror) needs its own distinct storage location. This collection of file system locations for all components in a Greenplum system is referred to as a filespace.

**Table 141: pg_catalog.pg_filespace**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsname</td>
<td>name</td>
<td></td>
<td>The name of the filespace.</td>
</tr>
<tr>
<td>fsowner</td>
<td>oid</td>
<td>pg_roles.oid</td>
<td>The object id of the role that created the filespace.</td>
</tr>
</tbody>
</table>

**pg_filespace_entry**

A tablespace requires a file system location to store its database files. In Greenplum Database, the master and each segment (primary and mirror) needs its own distinct storage location. This collection of file system locations for all components in a Greenplum system is referred to as a filespace. The `pg_filespace_entry` table contains information about the collection of file system locations across a Greenplum Database system that comprise a Greenplum Database filespace.

**Table 142: pg_catalog.pg_filespace_entry**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsefsoid</td>
<td>OID</td>
<td>pg_filespace.oid</td>
<td>Object id of the filespace.</td>
</tr>
<tr>
<td>fsedbid</td>
<td>integer</td>
<td>gp_segment_ configuration.dbid</td>
<td>Segment id.</td>
</tr>
</tbody>
</table>
### pg_index

The `pg_index` system catalog table contains part of the information about indexes. The rest is mostly in `pg_class`.

**Table 143: pg_catalog.pg_index**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>indexrelid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The OID of the pg_class entry for this index.</td>
</tr>
<tr>
<td>indrelid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The OID of the pg_class entry for the table this index is for.</td>
</tr>
<tr>
<td>indnatts</td>
<td>int2</td>
<td></td>
<td>The number of columns in the index (duplicates pg_class.relnatts).</td>
</tr>
<tr>
<td>indisunique</td>
<td>boolean</td>
<td></td>
<td>If true, this is a unique index.</td>
</tr>
<tr>
<td>indisprimary</td>
<td>boolean</td>
<td></td>
<td>If true, this index represents the primary key of the table. (indisunique should always be true when this is true.)</td>
</tr>
<tr>
<td>indisclustered</td>
<td>boolean</td>
<td></td>
<td>If true, the table was last clustered on this index via the CLUSTER command.</td>
</tr>
<tr>
<td>indisvalid</td>
<td>boolean</td>
<td></td>
<td>If true, the index is currently valid for queries. False means the index is possibly incomplete: it must still be modified by INSERT/UPDATE operations, but it cannot safely be used for queries.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------</td>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>indkey</td>
<td>int2vector</td>
<td>pg_attribute.attnum</td>
<td>This is an array of indnatts values that indicate which table columns this index indexes. For example a value of 1 3 would mean that the first and the third table columns make up the index key. A zero in this array indicates that the corresponding index attribute is an expression over the table columns, rather than a simple column reference.</td>
</tr>
<tr>
<td>indclass</td>
<td>oidvector</td>
<td>pg_opclass.oid</td>
<td>For each column in the index key this contains the OID of the operator class to use.</td>
</tr>
<tr>
<td>indexprs</td>
<td>text</td>
<td></td>
<td>Expression trees (in nodeToString() representation) for index attributes that are not simple column references. This is a list with one element for each zero entry in indkey. NULL if all index attributes are simple references.</td>
</tr>
<tr>
<td>indpred</td>
<td>text</td>
<td></td>
<td>Expression tree (in nodeToString() representation) for partial index predicate. NULL if not a partial index.</td>
</tr>
</tbody>
</table>

**pg_inherits**

The `pg_inherits` system catalog table records information about table inheritance hierarchies. There is one entry for each direct child table in the database. (Indirect inheritance can be determined by following chains of entries.) In Greenplum Database, inheritance relationships are created by both the `INHERITS` clause (standalone inheritance) and the `PARTITION BY` clause (partitioned child table inheritance) of `CREATE TABLE`.

**Table 144: pg_catalog.pg_inherits**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inhrelid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The OID of the child table.</td>
</tr>
</tbody>
</table>
Greenplum Database Reference Guide

Release Notes

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inhparent</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The OID of the parent table.</td>
</tr>
<tr>
<td>inhseqno</td>
<td>int4</td>
<td></td>
<td>If there is more than one direct parent for a child table (multiple inheritance), this number tells the order in which the inherited columns are to be arranged. The count starts at 1.</td>
</tr>
</tbody>
</table>

**pg_language**

The `pg_language` system catalog table registers languages in which you can write functions or stored procedures. It is populated by `CREATE LANGUAGE`.

Table 145: `pg_catalog.pg_language`

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lanname</td>
<td>name</td>
<td></td>
<td>Name of the language.</td>
</tr>
<tr>
<td>lanowner</td>
<td>oid</td>
<td>pg_authid.oid</td>
<td>Owner of the language.</td>
</tr>
<tr>
<td>lanispl</td>
<td>boolean</td>
<td></td>
<td>This is false for internal languages (such as SQL) and true for user-defined languages. Currently, <code>pg_dump</code> still uses this to determine which languages need to be dumped, but this may be replaced by a different mechanism in the future.</td>
</tr>
<tr>
<td>lanpltrusted</td>
<td>boolean</td>
<td></td>
<td>True if this is a trusted language, which means that it is believed not to grant access to anything outside the normal SQL execution environment. Only superusers may create functions in untrusted languages.</td>
</tr>
<tr>
<td>lanplcallfoid</td>
<td>oid</td>
<td>pg_proc.oid</td>
<td>For noninternal languages this references the language handler, which is a special function that is responsible for executing all functions that are written in the particular language.</td>
</tr>
</tbody>
</table>
### `laninline`

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>laninline</td>
<td>oid</td>
<td>pg_proc.oid</td>
<td>This references a function that is responsible for executing inline anonymous code blocks (see the DO command). Zero if anonymous blocks are not supported.</td>
</tr>
</tbody>
</table>

### `lanvalidator`

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lanvalidator</td>
<td>oid</td>
<td>pg_proc.oid</td>
<td>This references a language validator function that is responsible for checking the syntax and validity of new functions when they are created. Zero if no validator is provided.</td>
</tr>
</tbody>
</table>

### `lanacl`

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lanacl</td>
<td>aclitem[]</td>
<td></td>
<td>Access privileges for the language.</td>
</tr>
</tbody>
</table>

### `pg_largeobject`

The `pg_largeobject` system catalog table holds the data making up 'large objects'. A large object is identified by an OID assigned when it is created. Each large object is broken into segments or 'pages' small enough to be conveniently stored as rows in `pg_largeobject`. The amount of data per page is defined to be `LOBLKSIZE` (which is currently `BLCKSZ/4`, or typically 8K).

Each row of `pg_largeobject` holds data for one page of a large object, beginning at byte offset (`pageno * LOBLKSIZE`) within the object. The implementation allows sparse storage: pages may be missing, and may be shorter than `LOBLKSIZE` bytes even if they are not the last page of the object. Missing regions within a large object read as zeroes.

**Table 146: pg_catalog.pg_largeobject**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>loid</td>
<td>oid</td>
<td></td>
<td>Identifier of the large object that includes this page.</td>
</tr>
<tr>
<td>pageno</td>
<td>int4</td>
<td></td>
<td>Page number of this page within its large object (counting from zero).</td>
</tr>
<tr>
<td>data</td>
<td>bytea</td>
<td></td>
<td>Actual data stored in the large object. This will never be more than <code>LOBLKSIZE</code> bytes and may be less.</td>
</tr>
</tbody>
</table>

### `pg_listener`

The `pg_listener` system catalog table supports the LISTEN and NOTIFY commands. A listener creates an entry in `pg_listener` for each notification name it is listening for. A notifier scans and updates each
matching entry to show that a notification has occurred. The notifier also sends a signal (using the PID recorded in the table) to awaken the listener from sleep.

This table is not currently used in Greenplum Database.

**Table 147: pg_catalog.pg_listener**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>relname</td>
<td>name</td>
<td></td>
<td>Notify condition name. (The name need not match any actual relation in the database.</td>
</tr>
<tr>
<td>listenerpid</td>
<td>int4</td>
<td></td>
<td>PID of the server process that created this entry.</td>
</tr>
<tr>
<td>notification</td>
<td>int4</td>
<td></td>
<td>Zero if no event is pending for this listener. If an event is pending, the PID of the server process that sent the notification.</td>
</tr>
</tbody>
</table>

**pg_locks**

The `pg_locks` view provides access to information about the locks held by open transactions within Greenplum Database.

`pg_locks` contains one row per active lockable object, requested lock mode, and relevant transaction. Thus, the same lockable object may appear many times if multiple transactions are holding or waiting for locks on it. An object with no current locks on it will not appear in the view at all.

There are several distinct types of lockable objects: whole relations (such as tables), individual pages of relations, individual tuples of relations, transaction IDs (both virtual and permanent IDs), and general database objects. Also, the right to extend a relation is represented as a separate lockable object.

**Table 148: pg_catalog.pg_locks**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>locktype</td>
<td>text</td>
<td></td>
<td>Type of the lockable object: relation, extend, page, tuple, transactionid, object, userlock, resource queue, or advisory</td>
</tr>
<tr>
<td>database</td>
<td>oid</td>
<td>pg_database.oid</td>
<td>OID of the database in which the object exists, zero if the object is a shared object, or NULL if the object is a transaction ID</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>relation</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>OID of the relation, or NULL if the object is not a relation or part of a relation</td>
</tr>
<tr>
<td>page</td>
<td>integer</td>
<td></td>
<td>Page number within the relation, or NULL if the object is not a relation page</td>
</tr>
<tr>
<td>tuple</td>
<td>smallint</td>
<td></td>
<td>Tuple number within the page, or NULL if the object is not a tuple</td>
</tr>
<tr>
<td>virtualxid</td>
<td>text</td>
<td></td>
<td>Virtual ID of a transaction, or NULL if the object is not a virtual transaction ID</td>
</tr>
<tr>
<td>transactionid</td>
<td>xid</td>
<td></td>
<td>ID of a transaction, or NULL if the object is not a transaction ID</td>
</tr>
<tr>
<td>classid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>OID of the system catalog containing the object, or NULL if the object is not a general database object</td>
</tr>
<tr>
<td>objid</td>
<td>oid</td>
<td>any OID column</td>
<td>OID of the object within its system catalog, or NULL if the object is not a general database object</td>
</tr>
<tr>
<td>objsubid</td>
<td>smallint</td>
<td></td>
<td>For a table column, this is the column number (the classid and objid refer to the table itself). For all other object types, this column is zero. NULL if the object is not a general database object</td>
</tr>
<tr>
<td>virtualtransaction</td>
<td>text</td>
<td></td>
<td>Virtual ID of the transaction that is holding or awaiting this lock</td>
</tr>
<tr>
<td>pid</td>
<td>integer</td>
<td></td>
<td>Process ID of the server process holding or awaiting this lock. NULL if the lock is held by a prepared transaction</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mode</td>
<td>text</td>
<td></td>
<td>Name of the lock mode held or desired by this process</td>
</tr>
<tr>
<td>granted</td>
<td>boolean</td>
<td></td>
<td>True if lock is held, false if lock is awaited.</td>
</tr>
<tr>
<td>mppsessionid</td>
<td>integer</td>
<td></td>
<td>The id of the client session associated with this lock.</td>
</tr>
<tr>
<td>mppiswriter</td>
<td>boolean</td>
<td></td>
<td>Specifies whether the lock is held by a writer process.</td>
</tr>
<tr>
<td>gp_segment_id</td>
<td>integer</td>
<td></td>
<td>The Greenplum segment id (dbid) where the lock is held.</td>
</tr>
</tbody>
</table>

**pg_max_external_files**

The `pg_max_external_files` view shows the maximum number of external table files allowed per segment host when using the external table file protocol.

**Table 149: pg_catalog.pg_max_external_files**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostname</td>
<td>name</td>
<td></td>
<td>The host name used to access a particular segment instance on a segment host.</td>
</tr>
<tr>
<td>maxfiles</td>
<td>bigint</td>
<td></td>
<td>Number of primary segment instances on the host.</td>
</tr>
</tbody>
</table>

**pg_namespace**

The `pg_namespace` system catalog table stores namespaces. A namespace is the structure underlying SQL schemas: each namespace can have a separate collection of relations, types, etc. without name conflicts.

**Table 150: pg_catalog.pg_namespace**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nspname</td>
<td>name</td>
<td></td>
<td>Name of the namespace</td>
</tr>
<tr>
<td>nspowner</td>
<td>oid</td>
<td>pg_authid.oid</td>
<td>Owner of the namespace</td>
</tr>
<tr>
<td>nspacl</td>
<td>aclitem[]</td>
<td></td>
<td>Access privileges as given by GRANT and REVOKE.</td>
</tr>
</tbody>
</table>
**pg_opclass**

The `pg_opclass` system catalog table defines index access method operator classes. Each operator class defines semantics for index columns of a particular data type and a particular index access method. Note that there can be multiple operator classes for a given data type/access method combination, thus supporting multiple behaviors. The majority of the information defining an operator class is actually not in its `pg_opclass` row, but in the associated rows in `pg_amop` and `pg_amproc`. Those rows are considered to be part of the operator class definition — this is not unlike the way that a relation is defined by a single `pg_class` row plus associated rows in `pg_attribute` and other tables.

### Table 151: pg_catalog.pg_opclass

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcamid</td>
<td>oid</td>
<td>pg_am.oid</td>
<td>Index access method operator class is for.</td>
</tr>
<tr>
<td>opcname</td>
<td>name</td>
<td></td>
<td>Name of this operator class</td>
</tr>
<tr>
<td>opcnamespace</td>
<td>oid</td>
<td>pg_namespace.oid</td>
<td>Namespace of this operator class</td>
</tr>
<tr>
<td>opcowner</td>
<td>oid</td>
<td>pg_authid.oid</td>
<td>Owner of the operator class</td>
</tr>
<tr>
<td>opcintype</td>
<td>oid</td>
<td>pg_type.oid</td>
<td>Data type that the operator class indexes.</td>
</tr>
<tr>
<td>opcdefault</td>
<td>boolean</td>
<td></td>
<td>True if this operator class is the default for the data type opcintype.</td>
</tr>
<tr>
<td>opckeytype</td>
<td>oid</td>
<td>pg_type.oid</td>
<td>Type of data stored in index, or zero if same as opcintype.</td>
</tr>
</tbody>
</table>

**pg_operator**

The `pg_operator` system catalog table stores information about operators, both built-in and those defined by `CREATE OPERATOR`. Unused column contain zeroes. For example, `oprleft` is zero for a prefix operator.

### Table 152: pg_catalog.pg_operator

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oprname</td>
<td>name</td>
<td></td>
<td>Name of the operator.</td>
</tr>
<tr>
<td>oprnamespace</td>
<td>oid</td>
<td>pg_namespace.oid</td>
<td>The OID of the namespace that contains this operator.</td>
</tr>
<tr>
<td>oprowner</td>
<td>oid</td>
<td>pg_authid.oid</td>
<td>Owner of the operator.</td>
</tr>
<tr>
<td>oprkind</td>
<td>char</td>
<td></td>
<td>$b = $infix (both), $l = prefix (left), $r = postfix (right)$</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>-------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>oprcanhash</td>
<td>boolean</td>
<td></td>
<td>This operator supports hash joins.</td>
</tr>
<tr>
<td>oprleft</td>
<td>oid</td>
<td>pg_type.oid</td>
<td>Type of the left operand.</td>
</tr>
<tr>
<td>oprright</td>
<td>oid</td>
<td>pg_type.oid</td>
<td>Type of the right operand.</td>
</tr>
<tr>
<td>oprresult</td>
<td>oid</td>
<td>pg_type.oid</td>
<td>Type of the result.</td>
</tr>
<tr>
<td>oprcom</td>
<td>oid</td>
<td>pg_operator.oid</td>
<td>Commutator of this operator, if any.</td>
</tr>
<tr>
<td>oprnegate</td>
<td></td>
<td>pg_operator.oid</td>
<td>Negator of this operator, if any.</td>
</tr>
<tr>
<td>oprlsorop</td>
<td>oid</td>
<td>pg_operator.oid</td>
<td>If this operator supports merge joins, the operator that sorts the type of the left-hand operand (L&lt; L).</td>
</tr>
<tr>
<td>oprrsorop</td>
<td>oid</td>
<td>pg_operator.oid</td>
<td>If this operator supports merge joins, the operator that sorts the type of the right-hand operand (R&lt; R).</td>
</tr>
<tr>
<td>oprltcmpop</td>
<td>oid</td>
<td>pg_operator.oid</td>
<td>If this operator supports merge joins, the less-than operator that compares the left and right operand types (L&lt; R).</td>
</tr>
<tr>
<td>oprgtcmpop</td>
<td>oid</td>
<td>pg_operator.oid</td>
<td>If this operator supports merge joins, the greater-than operator that compares the left and right operand types (L&gt; R).</td>
</tr>
<tr>
<td>oprcode</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Function that implements this operator.</td>
</tr>
<tr>
<td>oprrest</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Restriction selectivity estimation function for this operator.</td>
</tr>
<tr>
<td>oprjoin</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Join selectivity estimation function for this operator.</td>
</tr>
</tbody>
</table>

**pg_partition**

The *pg_partition* system catalog table is used to track partitioned tables and their inheritance level relationships. Each row of *pg_partition* represents either the level of a partitioned table in the partition hierarchy, or a subpartition template description. The value of the attribute `paristemplate` determines what a particular row represents.
Table 153: pg_catalog.pg_partition

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parrelid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The object identifier of the table.</td>
</tr>
<tr>
<td>parkind</td>
<td>char</td>
<td></td>
<td>The partition type - R for range or L for list.</td>
</tr>
<tr>
<td>parlevel</td>
<td>smallint</td>
<td></td>
<td>The partition level of this row: 0 for the top-level parent table, 1 for the first level under the parent table, 2 for the second level, and so on.</td>
</tr>
<tr>
<td>paristemplate</td>
<td>boolean</td>
<td></td>
<td>Whether or not this row represents a subpartition template definition (true) or an actual partitioning level (false).</td>
</tr>
<tr>
<td>parnatts</td>
<td>smallint</td>
<td></td>
<td>The number of attributes that define this level.</td>
</tr>
<tr>
<td>paratts</td>
<td>smallint()</td>
<td></td>
<td>An array of the attribute numbers (as in pg_attribute.attnum) of the attributes that participate in defining this level.</td>
</tr>
<tr>
<td>parclass</td>
<td>oidvector</td>
<td>pg_opclass.oid</td>
<td>The operator class identifier(s) of the partition columns.</td>
</tr>
</tbody>
</table>

pg_partition_columns

The pg_partition_columns system view is used to show the partition key columns of a partitioned table.

Table 154: pg_catalog.pg_partition_columns

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaname</td>
<td>name</td>
<td></td>
<td>The name of the schema the partitioned table is in.</td>
</tr>
<tr>
<td>tablename</td>
<td>name</td>
<td></td>
<td>The table name of the top-level parent table.</td>
</tr>
<tr>
<td>columnname</td>
<td>name</td>
<td></td>
<td>The name of the partition key column.</td>
</tr>
<tr>
<td>partitionlevel</td>
<td>smallint</td>
<td></td>
<td>The level of this subpartition in the hierarchy.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>position_in_partition_key</td>
<td>integer</td>
<td></td>
<td>For list partitions you can have a composite (multi-column) partition key. This shows the position of the column in a composite key.</td>
</tr>
</tbody>
</table>

### pg_partition_encoding

The `pg_partition_encoding` system catalog table describes the available column compression options for a partition template.

**Table 155: pg_catalog.pg_attribute_encoding**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>modifiers</th>
<th>storage</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parencoid</td>
<td>oid</td>
<td>not null</td>
<td>plain</td>
<td></td>
</tr>
<tr>
<td>parencattnum</td>
<td>smallint</td>
<td>not null</td>
<td>plain</td>
<td></td>
</tr>
<tr>
<td>parencatoptions</td>
<td>text []</td>
<td></td>
<td>extended</td>
<td></td>
</tr>
</tbody>
</table>

### pg_partition_rule

The `pg_partition_rule` system catalog table is used to track partitioned tables, their check constraints, and data containment rules. Each row of `pg_partition_rule` represents either a leaf partition (the bottom level partitions that contain data), or a branch partition (a top or mid-level partition that is used to define the partition hierarchy, but does not contain any data).

**Table 156: pg_catalog.pg_partition_rule**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>paroid</td>
<td>oid</td>
<td>pg_partition.oid</td>
<td>Row identifier of the partitioning level (from <code>pg_partition</code>) to which this partition belongs. In the case of a branch partition, the corresponding table (identified by <code>pg_partition_rule</code>) is an empty container table. In case of a leaf partition, the table contains the rows for that partition containment rule.</td>
</tr>
<tr>
<td>parchildrelid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The table identifier of the partition (child table).</td>
</tr>
<tr>
<td>parparentrule</td>
<td>oid</td>
<td>pg_partition_rule.paroid</td>
<td>The row identifier of the rule associated with the parent table of this partition.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>parname</td>
<td>name</td>
<td></td>
<td>The given name of this partition.</td>
</tr>
<tr>
<td>parisdefault</td>
<td>boolean</td>
<td></td>
<td>Whether or not this partition is a default partition.</td>
</tr>
<tr>
<td>parruleord</td>
<td>smallint</td>
<td></td>
<td>For range partitioned tables, the rank of this partition on this level of the partition hierarchy.</td>
</tr>
<tr>
<td>parrangestartincl</td>
<td>boolean</td>
<td></td>
<td>For range partitioned tables, whether or not the starting value is inclusive.</td>
</tr>
<tr>
<td>parrangeendincl</td>
<td>boolean</td>
<td></td>
<td>For range partitioned tables, whether or not the ending value is inclusive.</td>
</tr>
<tr>
<td>parrangestart</td>
<td>text</td>
<td></td>
<td>For range partitioned tables, the starting value of the range.</td>
</tr>
<tr>
<td>parrangeend</td>
<td>text</td>
<td></td>
<td>For range partitioned tables, the ending value of the range.</td>
</tr>
<tr>
<td>rangeevery</td>
<td>text</td>
<td></td>
<td>For range partitioned tables, the interval value of the EVERY clause.</td>
</tr>
<tr>
<td>parlistvalues</td>
<td>text</td>
<td></td>
<td>For list partitioned tables, the list of values assigned to this partition.</td>
</tr>
<tr>
<td>parreloptions</td>
<td>text</td>
<td></td>
<td>An array describing the storage characteristics of the particular partition.</td>
</tr>
</tbody>
</table>

**pg_partition_templates**

The `pg_partition_templates` system view is used to show the subpartitions that were created using a subpartition template.

**Table 157: pg_catalog.pg_partition_templates**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaname</td>
<td>name</td>
<td></td>
<td>The name of the schema the partitioned table is in.</td>
</tr>
<tr>
<td>tablename</td>
<td>name</td>
<td></td>
<td>The table name of the top-level parent table.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------</td>
<td>------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>partitionname</td>
<td>name</td>
<td></td>
<td>The name of the subpartition (this is the name to use if referring to the partition in an <code>ALTER TABLE</code> command). NULL if the partition was not given a name at create time or generated by an <code>EVERY</code> clause.</td>
</tr>
<tr>
<td>partitiontype</td>
<td>text</td>
<td></td>
<td>The type of subpartition (range or list).</td>
</tr>
<tr>
<td>partitionlevel</td>
<td>smallint</td>
<td></td>
<td>The level of this subpartition in the hierarchy.</td>
</tr>
<tr>
<td>partitionrank</td>
<td>bigint</td>
<td></td>
<td>For range partitions, the rank of the partition compared to other partitions of the same level.</td>
</tr>
<tr>
<td>partitionposition</td>
<td>smallint</td>
<td></td>
<td>The rule order position of this subpartition.</td>
</tr>
<tr>
<td>partitionlistvalues</td>
<td>text</td>
<td></td>
<td>For list partitions, the list value(s) associated with this subpartition.</td>
</tr>
<tr>
<td>partitionrangestart</td>
<td>text</td>
<td></td>
<td>For range partitions, the start value of this subpartition.</td>
</tr>
<tr>
<td>partitionstartinclusive</td>
<td>boolean</td>
<td></td>
<td>T if the start value is included in this subpartition. F if it is excluded.</td>
</tr>
<tr>
<td>partitionrangeend</td>
<td>text</td>
<td></td>
<td>For range partitions, the end value of this subpartition.</td>
</tr>
<tr>
<td>partitionendinclusive</td>
<td>boolean</td>
<td></td>
<td>T if the end value is included in this subpartition. F if it is excluded.</td>
</tr>
<tr>
<td>partitioneveryclause</td>
<td>text</td>
<td></td>
<td>The <code>EVERY</code> clause (interval) of this subpartition.</td>
</tr>
<tr>
<td>partitionisdefault</td>
<td>boolean</td>
<td></td>
<td>T if this is a default subpartition, otherwise F.</td>
</tr>
</tbody>
</table>
### partitionboundary

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>partitionboundary</td>
<td>text</td>
<td></td>
<td>The entire partition specification for this subpartition.</td>
</tr>
</tbody>
</table>

### pg_partitions

The `pg_partitions` system view is used to show the structure of a partitioned table.

**Table 158: pg_catalog.pg_partitions**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaname</td>
<td>name</td>
<td></td>
<td>The name of the schema the partitioned table is in.</td>
</tr>
<tr>
<td>tablename</td>
<td>name</td>
<td></td>
<td>The name of the top-level parent table.</td>
</tr>
<tr>
<td>partitiontablename</td>
<td>name</td>
<td></td>
<td>The relation name of the partitioned table (this is the table name to use if accessing the partition directly).</td>
</tr>
<tr>
<td>partitionname</td>
<td>name</td>
<td></td>
<td>The name of the partition (this is the name to use if referring to the partition in an <code>ALTER TABLE</code> command). <strong>NULL</strong> if the partition was not given a name at create time or generated by an <strong>EVERY</strong> clause.</td>
</tr>
<tr>
<td>parentpartitiontablename</td>
<td>name</td>
<td></td>
<td>The relation name of the parent table one level up from this partition.</td>
</tr>
<tr>
<td>parentpartitionname</td>
<td>name</td>
<td></td>
<td>The given name of the parent table one level up from this partition.</td>
</tr>
<tr>
<td>partitiontype</td>
<td>text</td>
<td></td>
<td>The type of partition (range or list).</td>
</tr>
<tr>
<td>partitionlevel</td>
<td>smallint</td>
<td></td>
<td>The level of this partition in the hierarchy.</td>
</tr>
<tr>
<td>partitionrank</td>
<td>bigint</td>
<td></td>
<td>For range partitions, the rank of the partition compared to other partitions of the same level.</td>
</tr>
<tr>
<td>partitionposition</td>
<td>smallint</td>
<td></td>
<td>The rule order position of this partition.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>partitionlistvalues</td>
<td>text</td>
<td></td>
<td>For list partitions, the list value(s) associated with this partition.</td>
</tr>
<tr>
<td>partitionrangestart</td>
<td>text</td>
<td></td>
<td>For range partitions, the start value of this partition.</td>
</tr>
<tr>
<td>partitionstartinclusive</td>
<td>boolean</td>
<td></td>
<td>T if the start value is included in this partition. F if it is excluded.</td>
</tr>
<tr>
<td>partitionrangeend</td>
<td>text</td>
<td></td>
<td>For range partitions, the end value of this partition.</td>
</tr>
<tr>
<td>partitionendinclusive</td>
<td>boolean</td>
<td></td>
<td>T if the end value is included in this partition. F if it is excluded.</td>
</tr>
<tr>
<td>partitioneveryclause</td>
<td>text</td>
<td></td>
<td>The EVERY clause (interval) of this partition.</td>
</tr>
<tr>
<td>partitionisdefault</td>
<td>boolean</td>
<td></td>
<td>T if this is a default partition, otherwise F.</td>
</tr>
<tr>
<td>partitionboundary</td>
<td>text</td>
<td></td>
<td>The entire partition specification for this partition.</td>
</tr>
</tbody>
</table>

**pg_pltemplate**

The `pg_pltemplate` system catalog table stores template information for procedural languages. A template for a language allows the language to be created in a particular database by a simple `CREATE LANGUAGE` command, with no need to specify implementation details. Unlike most system catalogs, `pg_pltemplate` is shared across all databases of Greenplum system: there is only one copy of `pg_pltemplate` per system, not one per database. This allows the information to be accessible in each database as it is needed.

There are not currently any commands that manipulate procedural language templates; to change the built-in information, a superuser must modify the table using ordinary `INSERT`, `DELETE`, or `UPDATE` commands.

**Table 159: pg_catalog.pg_pltemplate**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmplname</td>
<td>name</td>
<td></td>
<td>Name of the language this template is for</td>
</tr>
<tr>
<td>tmpltrusted</td>
<td>boolean</td>
<td></td>
<td>True if language is considered trusted</td>
</tr>
<tr>
<td>tmplhandler</td>
<td>text</td>
<td></td>
<td>Name of call handler function</td>
</tr>
<tr>
<td>tmplvalidator</td>
<td>text</td>
<td></td>
<td>Name of validator function, or NULL if none</td>
</tr>
</tbody>
</table>
### pg_proc

The `pg_proc` system catalog table stores information about functions (or procedures), both built-in functions and those defined by `CREATE FUNCTION`. The table contains data for aggregate and window functions as well as plain functions. If `proisagg` is true, there should be a matching row in `pg_aggregate`. If `proiswin` is true, there should be a matching row in `pg_window`.

For compiled functions, both built-in and dynamically loaded, `prosrc` contains the function's C-language name (link symbol). For all other currently-known language types, `prosrc` contains the function's source text. `probin` is unused except for dynamically-loaded C functions, for which it gives the name of the shared library file containing the function.

**Table 160: pg_catalog.pg_proc**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proname</td>
<td>name</td>
<td></td>
<td>Name of the function.</td>
</tr>
<tr>
<td>pronamespace</td>
<td>oid</td>
<td>pg_namespace.oid</td>
<td>The OID of the namespace that contains this function.</td>
</tr>
<tr>
<td>proowner</td>
<td>oid</td>
<td>pg_authid.oid</td>
<td>Owner of the function.</td>
</tr>
<tr>
<td>prolang</td>
<td>oid</td>
<td>pg_language.oid</td>
<td>Implementation language or call interface of this function.</td>
</tr>
<tr>
<td>procost</td>
<td>float4</td>
<td></td>
<td>Estimated execution cost (in cpu_operator_cost units); if proretset is true, identifies the cost per row returned.</td>
</tr>
<tr>
<td>prorows</td>
<td>float4</td>
<td></td>
<td>Estimated number of result rows (zero if not proretset)</td>
</tr>
<tr>
<td>provariadic</td>
<td>oid</td>
<td>pg_type.oid</td>
<td>Data type of the variadic array parameter's elements, or zero if the function does not have a variadic parameter.</td>
</tr>
<tr>
<td>proisagg</td>
<td>boolean</td>
<td></td>
<td>Function is an aggregate function.</td>
</tr>
<tr>
<td>prosecdet</td>
<td>boolean</td>
<td></td>
<td>Function is a security definer (for example, a 'setuid' function).</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>proisstrict</td>
<td>boolean</td>
<td></td>
<td>Function returns NULL if any call argument is NULL. In that case the function will not actually be called at all. Functions that are not strict must be prepared to handle NULL inputs.</td>
</tr>
<tr>
<td>proretset</td>
<td>boolean</td>
<td></td>
<td>Function returns a set (multiple values of the specified data type).</td>
</tr>
</tbody>
</table>
| provolatile   | char          |            | Tells whether the function's result depends only on its input arguments, or is affected by outside factors.  
|               |               |            | $i = \text{immutable}$ (always delivers the same result for the same inputs),  
|               |               |            | $s = \text{stable}$ (results (for fixed inputs) do not change within a scan),  
<p>|               |               |            | $v = \text{volatile}$ (results may change at any time or functions with side-effects).                                                   |
| pronargs      | int2          |            | Number of arguments.                                                                                                                       |
| pronargdefaults | int2        |            | Number of arguments that have default values.                                                                                              |
| prorettype    | oid           | pg_type.oid| Data type of the return value.                                                                                                              |
| proiswin      | boolean       |            | Function is neither an aggregate nor a scalar function, but a pure window function.                                                       |
| proargtypes   | oidvector     | pg_type.oid| An array with the data types of the function arguments. This includes only input arguments (including INOUT and VARIADIC arguments), and thus represents the call signature of the function. |</p>
<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proallargtypes</td>
<td>oid[]</td>
<td>pg_type.oid</td>
<td>An array with the data types of the function arguments. This includes all arguments (including OUT and INOUT arguments); however, if all the arguments are IN arguments, this field will be null. Note that subscripting is 1-based, whereas for historical reasons proargtypes is subscripted from 0.</td>
</tr>
<tr>
<td>proargmodes</td>
<td>char[]</td>
<td></td>
<td>An array with the modes of the function arguments: i = IN, o = OUT, b = INOUT, v = VARIADIC. If all the arguments are IN arguments, this field will be null. Note that subscripts correspond to positions of proallargtypes not proargtypes.</td>
</tr>
<tr>
<td>proargnames</td>
<td>text[]</td>
<td></td>
<td>An array with the names of the function arguments. Arguments without a name are set to empty strings in the array. If none of the arguments have a name, this field will be null. Note that subscripts correspond to positions of proallargtypes not proargtypes.</td>
</tr>
<tr>
<td>proargdefaults</td>
<td>text</td>
<td></td>
<td>Expression trees for default argument values. This is a list with proargdefaults elements, corresponding to the last $N$ input arguments (i.e., the last $N$ proargtypes positions). If none of the arguments have defaults, this field is empty.</td>
</tr>
</tbody>
</table>
Greenplum Database Reference Guide

Release Notes

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prosrc</td>
<td>text</td>
<td></td>
<td>This tells the function handler how to invoke the function. It might be the actual source code of the function for interpreted languages, a link symbol, a file name, or just about anything else, depending on the implementation language/call convention.</td>
</tr>
<tr>
<td>probin</td>
<td>bytea</td>
<td></td>
<td>Additional information about how to invoke the function. Again, the interpretation is language-specific.</td>
</tr>
<tr>
<td>proacl</td>
<td>aclitem[]</td>
<td></td>
<td>Access privileges for the function as given by GRANT/REVOKE.</td>
</tr>
</tbody>
</table>

pg_resgroup

The pg_resgroup system catalog table contains information about Greenplum Database resource groups, which are used for managing concurrent statements, CPU, and memory resources. This table, defined in the pg_global tablespace, is globally shared across all databases in the system.

Table 161: pg_catalog.pg_resgroup

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsgname</td>
<td>name</td>
<td></td>
<td>The name of the resource group.</td>
</tr>
<tr>
<td>parent</td>
<td>oid</td>
<td></td>
<td>Unused; reserved for future use.</td>
</tr>
</tbody>
</table>

pg_resgroupcapability

The pg_resgroupcapability system catalog table contains information about the capabilities and limits of defined Greenplum Database resource groups. You can join this table to the pg_resgroup table by resource group object ID.

The pg_resgroupcapability table, defined in the pg_global tablespace, is globally shared across all databases in the system.

Table 162: pg_catalog.pg_resgroupcapability

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resgroupid</td>
<td>oid</td>
<td>pg_resgroup.oid</td>
<td>The object ID of the associated resource group.</td>
</tr>
<tr>
<td><strong>column</strong></td>
<td><strong>type</strong></td>
<td><strong>references</strong></td>
<td><strong>description</strong></td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
</tbody>
</table>
| reslimittype | smallint |                | The resource group limit type:  
| | | | 0 - Unknown  
| | | | 1 - Concurrency  
| | | | 2 - CPU  
| | | | 3 - Memory  
| | | | 4 - Memory shared quota  
| | | | 5 - Memory spill ratio |
| value | opaque type | | The specific value set for the resource limit referenced in this record. This value has the fixed type text, and will be converted to a different data type depending upon the limit referenced. |
| proposed | opaque type | | If you altered a resource limit and the limit cannot be updated immediately, the proposed value for the limit referenced in this record. Otherwise, proposed reflects the currently set value. |

**pg_resourcetype**

The `pg_resourcetype` system catalog table contains information about the extended attributes that can be assigned to Greenplum Database resource queues. Each row details an attribute and inherent qualities such as its default setting, whether it is required, and the value to disable it (when allowed).

This table is populated only on the master. This table is defined in the `pg_global` tablespace, meaning it is globally shared across all databases in the system.

**Table 163: pg_catalog.pg_resourcetype**

<table>
<thead>
<tr>
<th><strong>column</strong></th>
<th><strong>type</strong></th>
<th><strong>references</strong></th>
<th><strong>description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>restypid</td>
<td>smallint</td>
<td></td>
<td>The resource type ID.</td>
</tr>
<tr>
<td>resname</td>
<td>name</td>
<td></td>
<td>The name of the resource type.</td>
</tr>
<tr>
<td>resrequired</td>
<td>boolean</td>
<td></td>
<td>Whether the resource type is required for a valid resource queue.</td>
</tr>
</tbody>
</table>
The `pg_resqueue` system catalog table contains information about Greenplum Database resource queues, which are used for the resource management feature. This table is populated only on the master. This table is defined in the `pg_global` tablespace, meaning it is globally shared across all databases in the system.

Table 164: `pg_catalog.pg_resqueue`

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsqname</td>
<td>name</td>
<td></td>
<td>The name of the resource queue.</td>
</tr>
<tr>
<td>rsqcountlimit</td>
<td>real</td>
<td></td>
<td>The active query threshold of the resource queue.</td>
</tr>
<tr>
<td>rsqcostlimit</td>
<td>real</td>
<td></td>
<td>The query cost threshold of the resource queue.</td>
</tr>
<tr>
<td>rsqovercommit</td>
<td>boolean</td>
<td></td>
<td>Allows queries that exceed the cost threshold to run when the system is idle.</td>
</tr>
<tr>
<td>rsqignorecostlimit</td>
<td>real</td>
<td></td>
<td>The query cost limit of what is considered a 'small query'. Queries with a cost under this limit will not be queued and run immediately.</td>
</tr>
</tbody>
</table>

The `pg_resqueue_attributes` view allows administrators to see the attributes set for a resource queue, such as its active statement limit, query cost limits, and priority.
Table 165: pg_catalog.pg_resqueue_attributes

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsqname</td>
<td>name</td>
<td>pg_resqueue.rsqname</td>
<td>The name of the resource queue.</td>
</tr>
<tr>
<td>resname</td>
<td>text</td>
<td></td>
<td>The name of the resource queue attribute.</td>
</tr>
<tr>
<td>ressetting</td>
<td>text</td>
<td></td>
<td>The current value of a resource queue attribute.</td>
</tr>
<tr>
<td>restypid</td>
<td>integer</td>
<td></td>
<td>System assigned resource type id.</td>
</tr>
</tbody>
</table>

**pg_resqueuecapability**

The `pg_resqueuecapability` system catalog table contains information about the extended attributes, or capabilities, of existing Greenplum Database resource queues. Only resource queues that have been assigned an extended capability, such as a priority setting, are recorded in this table. This table is joined to the `pg_resqueue` table by resource queue object ID, and to the `pg_resourcetype` table by resource type ID (restypid).

This table is populated only on the master. This table is defined in the `pg_global` tablespace, meaning it is globally shared across all databases in the system.

Table 166: pg_catalog.pg_resqueuecapability

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsqueueid</td>
<td>oid</td>
<td>pg_resqueue.oid</td>
<td>The object ID of the associated resource queue.</td>
</tr>
<tr>
<td>restypid</td>
<td>smallint</td>
<td>pg_resourcetype. restypid</td>
<td>The resource type, derived from the <code>pg_resqueuecapability</code> system table.</td>
</tr>
<tr>
<td>ressetting</td>
<td>opaque type</td>
<td></td>
<td>The specific value set for the capability referenced in this record. Depending on the actual resource type, this value may have different data types.</td>
</tr>
</tbody>
</table>

**pg_rewrite**

The `pg_rewrite` system catalog table stores rewrite rules for tables and views. `pg_class.relhasrules` must be true if a table has any rules in this catalog.

Table 167: pg_catalog.pg_rewrite

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rulename</td>
<td>name</td>
<td></td>
<td>Rule name.</td>
</tr>
<tr>
<td>ev_class</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The table this rule is for.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ev_attr</td>
<td>int2</td>
<td></td>
<td>The column this rule is for (currently, always zero to indicate the whole table).</td>
</tr>
<tr>
<td>ev_type</td>
<td>char</td>
<td></td>
<td>Event type that the rule is for: 1 = SELECT, 2 = UPDATE, 3 = INSERT, 4 = DELETE.</td>
</tr>
<tr>
<td>is_instead</td>
<td>boolean</td>
<td></td>
<td>True if the rule is an INSTEAD rule.</td>
</tr>
<tr>
<td>ev_qual</td>
<td>text</td>
<td></td>
<td>Expression tree (in the form of a nodeToString() representation) for the rule's qualifying condition.</td>
</tr>
<tr>
<td>ev_action</td>
<td>text</td>
<td></td>
<td>Query tree (in the form of a nodeToString() representation) for the rule's action.</td>
</tr>
</tbody>
</table>

**pg_roles**

The view `pg_roles` provides access to information about database roles. This is simply a publicly readable view of `pg_authid` that blanks out the password field. This view explicitly exposes the OID column of the underlying table, since that is needed to do joins to other catalogs.

**Table 168: pg_catalog.pg_roles**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rolname</td>
<td>name</td>
<td></td>
<td>Role name</td>
</tr>
<tr>
<td>rolsuper</td>
<td>bool</td>
<td></td>
<td>Role has superuser privileges</td>
</tr>
<tr>
<td>rolinherit</td>
<td>bool</td>
<td></td>
<td>Role automatically inherits privileges of roles it is a member of</td>
</tr>
<tr>
<td>rolcreaterole</td>
<td>bool</td>
<td></td>
<td>Role may create more roles</td>
</tr>
<tr>
<td>rolcreatedb</td>
<td>bool</td>
<td></td>
<td>Role may create databases</td>
</tr>
<tr>
<td>rolcatupdate</td>
<td>bool</td>
<td></td>
<td>Role may update system catalogs directly. (Even a superuser may not do this unless this column is true.)</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rolcanlogin</td>
<td>bool</td>
<td></td>
<td>Role may log in. That is, this role can be given as the initial session authorization identifier</td>
</tr>
<tr>
<td>rolconnlimit</td>
<td>int4</td>
<td></td>
<td>For roles that can log in, this sets maximum number of concurrent connections this role can make. -1 means no limit</td>
</tr>
<tr>
<td>rolpassword</td>
<td>text</td>
<td></td>
<td>Not the password (always reads as ********)</td>
</tr>
<tr>
<td>rolvaliduntil</td>
<td>timestamp</td>
<td></td>
<td>Password expiry time (only used for password authentication); NULL if no expiration</td>
</tr>
<tr>
<td>rolconfig</td>
<td>text[]</td>
<td></td>
<td>Session defaults for run-time configuration variables</td>
</tr>
<tr>
<td>rolresqueue</td>
<td>oid</td>
<td>pg_resqueue.oid</td>
<td>Object ID of the resource queue this role is assigned to.</td>
</tr>
<tr>
<td>oid</td>
<td>oid</td>
<td>pg_authid.oid</td>
<td>Object ID of role</td>
</tr>
<tr>
<td>rolcreaterextgpfd</td>
<td>bool</td>
<td></td>
<td>Role may create readable external tables that use the gpfdist protocol.</td>
</tr>
<tr>
<td>rolcreaterexthttp</td>
<td>bool</td>
<td></td>
<td>Role may create readable external tables that use the http protocol.</td>
</tr>
<tr>
<td>rolcreatemewextgpfd</td>
<td>bool</td>
<td></td>
<td>Role may create writable external tables that use the gpfdist protocol.</td>
</tr>
<tr>
<td>rolcreaterexthdfs</td>
<td>bool</td>
<td></td>
<td>Role may create readable external tables that use the gphdfs protocol.</td>
</tr>
<tr>
<td>rolcreatewexthdfs</td>
<td>bool</td>
<td></td>
<td>Role may create writable external tables that use the gphdfs protocol.</td>
</tr>
<tr>
<td>rolresgroup</td>
<td>oid</td>
<td>pg_resgroup.oid</td>
<td>Object ID of the resource group to which this role is assigned.</td>
</tr>
</tbody>
</table>
**pg_shdepend**

The `pg_shdepend` system catalog table records the dependency relationships between database objects and shared objects, such as roles. This information allows Greenplum Database to ensure that those objects are unreferenced before attempting to delete them. See also `pg_depend`, which performs a similar function for dependencies involving objects within a single database. Unlike most system catalogs, `pg_shdepend` is shared across all databases of Greenplum system: there is only one copy of `pg_shdepend` per system, not one per database.

In all cases, a `pg_shdepend` entry indicates that the referenced object may not be dropped without also dropping the dependent object. However, there are several subflavors identified by `deptype`:

- **SHARED_DEPENDENCY_OWNER (o)** — The referenced object (which must be a role) is the owner of the dependent object.
- **SHARED_DEPENDENCY_ACL (a)** — The referenced object (which must be a role) is mentioned in the ACL (access control list) of the dependent object.
- **SHARED_DEPENDENCY_PIN (p)** — There is no dependent object; this type of entry is a signal that the system itself depends on the referenced object, and so that object must never be deleted. Entries of this type are created only by system initialization. The columns for the dependent object contain zeroes.

**Table 169: pg_catalog.pg_shdepend**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbid</td>
<td>oid</td>
<td>pg_database.oid</td>
<td>The OID of the database the dependent object is in, or zero for a shared object.</td>
</tr>
<tr>
<td>classid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The OID of the system catalog the dependent object is in.</td>
</tr>
<tr>
<td>objid</td>
<td>oid</td>
<td>any OID column</td>
<td>The OID of the specific dependent object.</td>
</tr>
<tr>
<td>objsubid</td>
<td>int4</td>
<td></td>
<td>For a table column, this is the column number. For all other object types, this column is zero.</td>
</tr>
<tr>
<td>refclassid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The OID of the system catalog the referenced object is in (must be a shared catalog).</td>
</tr>
<tr>
<td>refobjid</td>
<td>oid</td>
<td>any OID column</td>
<td>The OID of the specific referenced object.</td>
</tr>
<tr>
<td>refobjsubid</td>
<td>int4</td>
<td></td>
<td>For a table column, this is the referenced column number. For all other object types, this column is zero.</td>
</tr>
</tbody>
</table>
Greenplum Database Reference Guide

**deptype**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deptype</td>
<td>char</td>
<td></td>
<td>A code defining the specific semantics of this dependency relationship.</td>
</tr>
</tbody>
</table>

**pg_shdescription**

The **pg_shdescription** system catalog table stores optional descriptions (comments) for shared database objects. Descriptions can be manipulated with the `COMMENT` command and viewed with `psql`'s \d meta-commands. See also **pg_description**, which performs a similar function for descriptions involving objects within a single database. Unlike most system catalogs, **pg_shdescription** is shared across all databases of a Greenplum system: there is only one copy of **pg_shdescription** per system, not one per database.

**Table 170: pg_catalog.pg_shdescription**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objoid</td>
<td>oid</td>
<td>any OID column</td>
<td>The OID of the object this description pertains to.</td>
</tr>
<tr>
<td>classoid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The OID of the system catalog this object appears in.</td>
</tr>
<tr>
<td>description</td>
<td>text</td>
<td></td>
<td>Arbitrary text that serves as the description of this object.</td>
</tr>
</tbody>
</table>

**pg_stat_activity**

The view **pg_stat_activity** shows one row per server process and details about it associated user session and query. The columns that report data on the current query are available unless the parameter `stats_command_string` has been turned off. Furthermore, these columns are only visible if the user examining the view is a superuser or the same as the user owning the process being reported on.

The maximum length of the query text string stored in the column **current_query** can be controlled with the server configuration parameter `pgstat_track_activity_query_size`.

**Table 171: pg_catalog.pg_stat_activity**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datid</td>
<td>oid</td>
<td>pg_database.oid</td>
<td>Database OID</td>
</tr>
<tr>
<td>datname</td>
<td>name</td>
<td></td>
<td>Database name</td>
</tr>
<tr>
<td>procpid</td>
<td>integer</td>
<td></td>
<td>Process ID of the server process</td>
</tr>
<tr>
<td>sess_id</td>
<td>integer</td>
<td></td>
<td>Session ID</td>
</tr>
<tr>
<td>usesysid</td>
<td>oid</td>
<td>pg_authid.oid</td>
<td>Role OID</td>
</tr>
<tr>
<td>username</td>
<td>name</td>
<td></td>
<td>Role name</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>-----------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>current_query</td>
<td>text</td>
<td></td>
<td>Current query that process is running</td>
</tr>
<tr>
<td>waiting</td>
<td>boolean</td>
<td></td>
<td>True if waiting on a lock, false if not waiting</td>
</tr>
<tr>
<td>query_start</td>
<td>timestamp</td>
<td></td>
<td>Time query began execution</td>
</tr>
<tr>
<td>backend_start</td>
<td>timestamp</td>
<td></td>
<td>Time backend process was started</td>
</tr>
<tr>
<td>client_addr</td>
<td>inet</td>
<td></td>
<td>Client address</td>
</tr>
<tr>
<td>client_port</td>
<td>integer</td>
<td></td>
<td>Client port</td>
</tr>
<tr>
<td>application_name</td>
<td>text</td>
<td></td>
<td>Client application name</td>
</tr>
<tr>
<td>xact_start</td>
<td>timestamp</td>
<td></td>
<td>Transaction start time</td>
</tr>
<tr>
<td>waiting_reason</td>
<td>text</td>
<td></td>
<td>Reason the server process is waiting. The value can be: lock, replication, or resgroup</td>
</tr>
<tr>
<td>rsgid</td>
<td>oid</td>
<td>pg_resgroup.oid</td>
<td>Resource group OID</td>
</tr>
<tr>
<td>rsgname</td>
<td>text</td>
<td>pg_resgroup.rsgname</td>
<td>Resource group name</td>
</tr>
<tr>
<td>rsgqueueduration</td>
<td>interval</td>
<td></td>
<td>For a queued query, the total time the query has been queued.</td>
</tr>
</tbody>
</table>

### pg_stat_last_operation

The `pg_stat_last_operation` table contains metadata tracking information about database objects (tables, views, etc.).

#### Table 172: `pg_catalog.pg_stat_last_operation`

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>classid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>OID of the system catalog containing the object.</td>
</tr>
<tr>
<td>objid</td>
<td>oid</td>
<td>any OID column</td>
<td>OID of the object within its system catalog.</td>
</tr>
<tr>
<td>staactionname</td>
<td>name</td>
<td></td>
<td>The action that was taken on the object.</td>
</tr>
<tr>
<td>stasysid</td>
<td>oid</td>
<td>pg_authid.oid</td>
<td>A foreign key to pg_authid.oid.</td>
</tr>
<tr>
<td>stausename</td>
<td>name</td>
<td></td>
<td>The name of the role that performed the operation on this object.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>stasubtype</td>
<td>text</td>
<td></td>
<td>The type of object operated on or the subclass of operation performed.</td>
</tr>
<tr>
<td>statime</td>
<td>timestamp with timezone</td>
<td></td>
<td>The timestamp of the operation. This is the same timestamp that is written to the Greenplum Database server log files in case you need to look up more detailed information about the operation in the logs.</td>
</tr>
</tbody>
</table>

**pg_stat_last_shoperation**

The `pg_stat_last_shoperation` table contains metadata tracking information about global objects (roles, tablespaces, etc.).

**Table 173: pg_catalog.pg_stat_last_shoperation**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>classid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>OID of the system catalog containing the object.</td>
</tr>
<tr>
<td>objid</td>
<td>oid</td>
<td>any OID column</td>
<td>OID of the object within its system catalog.</td>
</tr>
<tr>
<td>staactionname</td>
<td>name</td>
<td></td>
<td>The action that was taken on the object.</td>
</tr>
<tr>
<td>stasysid</td>
<td>oid</td>
<td></td>
<td>The name of the role that performed the operation on this object.</td>
</tr>
<tr>
<td>stausename</td>
<td>name</td>
<td></td>
<td>The type of object operated on or the subclass of operation performed.</td>
</tr>
<tr>
<td>stasubtype</td>
<td>text</td>
<td></td>
<td>The type of object operated on or the subclass of operation performed.</td>
</tr>
<tr>
<td>statime</td>
<td>timestamp with timezone</td>
<td></td>
<td>The timestamp of the operation. This is the same timestamp that is written to the Greenplum Database server log files in case you need to look up more detailed information about the operation in the logs.</td>
</tr>
</tbody>
</table>
**pg_stat_operations**

The view **pg_stat_operations** shows details about the last operation performed on a database object (such as a table, index, view or database) or a global object (such as a role).

**Table 174: pg_catalog.pg_stat_operations**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>classname</td>
<td>text</td>
<td></td>
<td>The name of the system table in the <code>pg_catalog</code> schema where the record about this object is stored (<code>pg_class=relations</code>, <code>pg_class=tables</code>, <code>pg_class=views</code>, <code>pg_namespace=schemas</code>, <code>pg_authid=roles</code>)</td>
</tr>
<tr>
<td>objname</td>
<td>name</td>
<td></td>
<td>The name of the object.</td>
</tr>
<tr>
<td>objid</td>
<td>oid</td>
<td></td>
<td>The OID of the object.</td>
</tr>
<tr>
<td>schemaname</td>
<td>name</td>
<td></td>
<td>The name of the schema where the object resides.</td>
</tr>
<tr>
<td>usestatus</td>
<td>text</td>
<td></td>
<td>The status of the role who performed the last operation on the object ([CURRENT]=a currently active role in the system, [DROPPED]=a role that no longer exists in the system, [CHANGED]=a role name that exists in the system, but has changed since the last operation was performed)</td>
</tr>
<tr>
<td>username</td>
<td>name</td>
<td></td>
<td>The name of the role that performed the operation on this object.</td>
</tr>
<tr>
<td>actionname</td>
<td>name</td>
<td></td>
<td>The action that was taken on the object.</td>
</tr>
<tr>
<td>subtype</td>
<td>text</td>
<td></td>
<td>The type of object operated on or the subclass of operation performed.</td>
</tr>
</tbody>
</table>
### pg_stat_partition_operations

The `pg_stat_partition_operations` view shows details about the last operation performed on a partitioned table.

#### Table 175: `pg_catalog.pg_stat_partition_operations`

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>classname</td>
<td>text</td>
<td></td>
<td>The name of the system table in the <code>pg_catalog</code> schema where the record about this object is stored (always <code>pg_class</code> for tables and partitions).</td>
</tr>
<tr>
<td>objname</td>
<td>name</td>
<td></td>
<td>The name of the object.</td>
</tr>
<tr>
<td>objid</td>
<td>oid</td>
<td></td>
<td>The OID of the object.</td>
</tr>
<tr>
<td>schemaname</td>
<td>name</td>
<td></td>
<td>The name of the schema where the object resides.</td>
</tr>
<tr>
<td>usestatus</td>
<td>text</td>
<td></td>
<td>The status of the role who performed the last operation on the object (CURRENT=a currently active role in the system, DROPPED=a role that no longer exists in the system, CHANGED=a role name that exists in the system, but its definition has changed since the last operation was performed).</td>
</tr>
<tr>
<td>usename</td>
<td>name</td>
<td></td>
<td>The name of the role that performed the operation on this object.</td>
</tr>
<tr>
<td>actionname</td>
<td>name</td>
<td></td>
<td>The action that was taken on the object.</td>
</tr>
</tbody>
</table>
### pg_stat_replication

The pg_stat_replication view contains metadata of the walsender process that is used for Greenplum Database master mirroring.

**Table 176: pg_catalog.pg_stat_replication**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>procpid</td>
<td>integer</td>
<td></td>
<td>Process ID of WAL sender backend process.</td>
</tr>
<tr>
<td>usesysid</td>
<td>integer</td>
<td></td>
<td>User system ID that runs the WAL sender backend process</td>
</tr>
<tr>
<td>usename</td>
<td>name</td>
<td></td>
<td>User name that runs WAL sender backend process.</td>
</tr>
<tr>
<td>application_name</td>
<td>oid</td>
<td></td>
<td>Client application name.</td>
</tr>
<tr>
<td>client_addr</td>
<td>name</td>
<td></td>
<td>Client IP address.</td>
</tr>
<tr>
<td>client_port</td>
<td>integer</td>
<td></td>
<td>Client port number.</td>
</tr>
<tr>
<td>backend_start</td>
<td>timestamp</td>
<td></td>
<td>Operation start timestamp.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------</td>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>state</td>
<td>text</td>
<td></td>
<td>WAL sender state. The value can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>startup</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>backup</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>catchup</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>streaming</td>
</tr>
<tr>
<td>sent_location</td>
<td>text</td>
<td></td>
<td>WAL sender xlog record sent location.</td>
</tr>
<tr>
<td>write_location</td>
<td>text</td>
<td></td>
<td>WAL receiver xlog record write location.</td>
</tr>
<tr>
<td>flush_location</td>
<td>text</td>
<td></td>
<td>WAL receiver xlog record flush location.</td>
</tr>
<tr>
<td>replay_location</td>
<td>text</td>
<td></td>
<td>Standby xlog record replay location.</td>
</tr>
<tr>
<td>sync_priority</td>
<td>text</td>
<td></td>
<td>Priority. the value is 1.</td>
</tr>
<tr>
<td>sync_state</td>
<td>text</td>
<td></td>
<td>WAL sender synchronization state. The value is \textit{sync}.</td>
</tr>
</tbody>
</table>

**pg\_statistic**

The \texttt{pg\_statistic} system catalog table stores statistical data about the contents of the database. Entries are created by \texttt{ANALYZE} and subsequently used by the query optimizer. There is one entry for each table column that has been analyzed. Note that all the statistical data is inherently approximate, even assuming that it is up-to-date.

\texttt{pg\_statistic} also stores statistical data about the values of index expressions. These are described as if they were actual data columns; in particular, \texttt{starelid} references the index. No entry is made for an ordinary non-expression index column, however, since it would be redundant with the entry for the underlying table column.

Since different kinds of statistics may be appropriate for different kinds of data, \texttt{pg\_statistic} is designed not to assume very much about what sort of statistics it stores. Only extremely general statistics (such as nullness) are given dedicated columns in \texttt{pg\_statistic}. Everything else is stored in slots, which are groups of associated columns whose content is identified by a code number in one of the slot's columns.

\texttt{pg\_statistic} should not be readable by the public, since even statistical information about a table's contents may be considered sensitive (for example: minimum and maximum values of a salary column). \texttt{pg\_stats} is a publicly readable view on \texttt{pg\_statistic} that only exposes information about those tables that are readable by the current user.

**Table 177: pg\_catalog.pg\_statistic**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>starelid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The table or index that the described column belongs to.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>staattnum</td>
<td>int2</td>
<td>pg_attribute.attnum</td>
<td>The number of the described column.</td>
</tr>
<tr>
<td>stanullfrac</td>
<td>float4</td>
<td></td>
<td>The fraction of the column's entries that are null.</td>
</tr>
<tr>
<td>stawidth</td>
<td>int4</td>
<td></td>
<td>The average stored width, in bytes, of nonnull entries.</td>
</tr>
<tr>
<td>stadistinct</td>
<td>float4</td>
<td></td>
<td>The number of distinct nonnull data values in the column. A value greater than zero is the actual number of distinct values. A value less than zero is the negative of a fraction of the number of rows in the table (for example, a column in which values appear about twice on the average could be represented by stadistinct = -0.5). A zero value means the number of distinct values is unknown.</td>
</tr>
<tr>
<td>stakindN</td>
<td>int2</td>
<td></td>
<td>A code number indicating the kind of statistics stored in the Nth slot of the pg_statistic row.</td>
</tr>
<tr>
<td>staopN</td>
<td>oid</td>
<td>pg_operator.oid</td>
<td>An operator used to derive the statistics stored in the Nth slot. For example, a histogram slot would show the &lt; operator that defines the sort order of the data.</td>
</tr>
<tr>
<td>stanumbersN</td>
<td>float4[]</td>
<td></td>
<td>Numerical statistics of the appropriate kind for the Nth slot, or NULL if the slot kind does not involve numerical values.</td>
</tr>
</tbody>
</table>
column | type    | description                                      |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>stavaluesN</td>
<td>anyarray</td>
<td>Column data values of the appropriate kind for the Nth slot, or NULL if the slot kind does not store any data values. Each array's element values are actually of the specific column's data type, so there is no way to define these columns' type more specifically than anyarray.</td>
</tr>
</tbody>
</table>

**pg_stat_resqueues**

The `pg_stat_resqueues` view allows administrators to view metrics about a resource queue's workload over time. To allow statistics to be collected for this view, you must enable the `stats_queue_level` server configuration parameter on the Greenplum Database master instance. Enabling the collection of these metrics does incur a small performance penalty, as each statement submitted through a resource queue must be logged in the system catalog tables.

**Table 178: pg_catalog.pg_stat_resqueues**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queueoid</td>
<td>oid</td>
<td>The OID of the resource queue.</td>
</tr>
<tr>
<td>queuename</td>
<td>name</td>
<td>The name of the resource queue.</td>
</tr>
<tr>
<td>n_queries_exec</td>
<td>bigint</td>
<td>Number of queries submitted for execution from this resource queue.</td>
</tr>
<tr>
<td>n_queries_wait</td>
<td>bigint</td>
<td>Number of queries submitted to this resource queue that had to wait before they could execute.</td>
</tr>
<tr>
<td>elapsed_exec</td>
<td>bigint</td>
<td>Total elapsed execution time for statements submitted through this resource queue.</td>
</tr>
<tr>
<td>elapsed_wait</td>
<td>bigint</td>
<td>Total elapsed time that statements submitted through this resource queue had to wait before they were executed.</td>
</tr>
</tbody>
</table>
**pg_tablespace**

The `pg_tablespace` system catalog table stores information about the available tablespaces. Tables can be placed in particular tablespaces to aid administration of disk layout. Unlike most system catalogs, `pg_tablespace` is shared across all databases of a Greenplum system: there is only one copy of `pg_tablespace` per system, not one per database.

**Table 179: pg_catalog.pg_tablespace**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spcname</td>
<td>name</td>
<td></td>
<td>Tablespace name.</td>
</tr>
<tr>
<td>spcowner</td>
<td>oid</td>
<td>pg_authid.oid</td>
<td>Owner of the tablespace, usually the user who created it.</td>
</tr>
<tr>
<td>spclocation</td>
<td>text[]</td>
<td></td>
<td>Deprecated.</td>
</tr>
<tr>
<td>spcacl</td>
<td>aclitem[]</td>
<td></td>
<td>Tablespace access privileges.</td>
</tr>
<tr>
<td>spcprilocations</td>
<td>text[]</td>
<td></td>
<td>Deprecated.</td>
</tr>
<tr>
<td>spcmrilocations</td>
<td>text[]</td>
<td></td>
<td>Deprecated.</td>
</tr>
<tr>
<td>spcfsoiid</td>
<td>oid</td>
<td>pg_filespace.oid</td>
<td>The object id of the filesystem used by this tablespace. A filesystem defines directory locations on the primary, mirror and master segments.</td>
</tr>
</tbody>
</table>

**pg_trigger**

The `pg_trigger` system catalog table stores triggers on tables.

**Note:** Greenplum Database does not support triggers.

**Table 180: pg_catalog.pg_trigger**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tgrelid</td>
<td>oid</td>
<td><code>pg_class.oid</code></td>
<td>The table this trigger is on. Note that Greenplum Database does not enforce referential integrity.</td>
</tr>
<tr>
<td>tgnname</td>
<td>name</td>
<td></td>
<td>Trigger name (must be unique among triggers of same table).</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tgfoid</td>
<td>oid</td>
<td>pg_proc.oid</td>
<td>The function to be called. Note that Greenplum Database does not enforce referential integrity.</td>
</tr>
<tr>
<td>tgtype</td>
<td>int2</td>
<td></td>
<td>Bit mask identifying trigger conditions.</td>
</tr>
<tr>
<td>tgenabled</td>
<td>boolean</td>
<td></td>
<td>True if trigger is enabled.</td>
</tr>
<tr>
<td>tgisconstraint</td>
<td>boolean</td>
<td></td>
<td>True if trigger implements a referential integrity constraint.</td>
</tr>
<tr>
<td>tgconstrname</td>
<td>name</td>
<td></td>
<td>Referential integrity constraint name.</td>
</tr>
<tr>
<td>tgconstrrelid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>The table referenced by an referential integrity constraint.</td>
</tr>
<tr>
<td>tgdeferrable</td>
<td>boolean</td>
<td></td>
<td>True if deferrable.</td>
</tr>
<tr>
<td>tginittest</td>
<td>boolean</td>
<td></td>
<td>True if initially deferred.</td>
</tr>
<tr>
<td>tgnargs</td>
<td>int2</td>
<td></td>
<td>Number of argument strings passed to trigger function.</td>
</tr>
<tr>
<td>tgnattr</td>
<td>int2vector</td>
<td></td>
<td>Currently not used.</td>
</tr>
<tr>
<td>targs</td>
<td>bytea</td>
<td></td>
<td>Argument strings to pass to trigger, each NULL-terminated.</td>
</tr>
</tbody>
</table>

**pg_type**

The *pg_type* system catalog table stores information about data types. Base types (scalar types) are created with `CREATE TYPE`, and domains with `CREATE DOMAIN`. A composite type is automatically created for each table in the database, to represent the row structure of the table. It is also possible to create composite types with `CREATE TYPE AS`.

**Table 181: pg_catalog.pg_type**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typename</td>
<td>name</td>
<td></td>
<td>Data type name.</td>
</tr>
<tr>
<td>typnamespacename</td>
<td>oid</td>
<td>pg_namespace.oid</td>
<td>The OID of the namespace that contains this type.</td>
</tr>
<tr>
<td>typowner</td>
<td>oid</td>
<td>pg_authid.oid</td>
<td>Owner of the type.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>typlenh</td>
<td>int2</td>
<td></td>
<td>For a fixed-size type, typlenh is the number of bytes in the internal representation of the type. But for a variable-length type, typlenh is negative. -1 indicates a 'varlena' type (one that has a length word), -2 indicates a null-terminated C string.</td>
</tr>
<tr>
<td>typhval</td>
<td>boolean</td>
<td></td>
<td>Determines whether internal routines pass a value of this type by value or by reference.</td>
</tr>
<tr>
<td>typtype</td>
<td>char</td>
<td></td>
<td>b for a base type, c for a composite type, d for a domain, e for an enum type, or p for a pseudo-type.</td>
</tr>
<tr>
<td>typlisdefined</td>
<td>boolean</td>
<td></td>
<td>True if the type is defined, false if this is a placeholder entry for a not-yet-defined type. When false, nothing except the type name, namespace, and OID can be relied on.</td>
</tr>
<tr>
<td>typdelim</td>
<td>char</td>
<td></td>
<td>Character that separates two values of this type when parsing array input. Note that the delimiter is associated with the array element data type, not the array data type.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>typrelid</td>
<td>oid</td>
<td>pg_class.oid</td>
<td>If this is a composite type, then this column points to the pg_class entry that defines the corresponding table. (For a free-standing composite type, the pg_class entry does not really represent a table, but it is needed anyway for the type’s pg_attribute entries to link to.) Zero for non-composite types.</td>
</tr>
<tr>
<td>typelem</td>
<td>oid</td>
<td>pg_type.oid</td>
<td>If not 0, then it identifies another row in pg_type. The current type can then be subscripted like an array yielding values of type typelem. A true array type is variable length (typlen = -1), but some fixed-length (tylen &gt; 0) types also have nonzero typelem, for example name and point. If a fixed-length type has a typelem then its internal representation must be some number of values of the typelem data type with no other data. Variable-length array types have a header defined by the array subroutines.</td>
</tr>
<tr>
<td>typarray</td>
<td>oid</td>
<td>pg_type.oid</td>
<td>If not 0, identifies another row in pg_type, which is the “true” array type having this type as its element. Use pg_type.typarray to locate the array type associated with a specific type.</td>
</tr>
<tr>
<td>typinput</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Input conversion function (text format).</td>
</tr>
<tr>
<td>typoutput</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Output conversion function (text format).</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>typreceive</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Input conversion function (binary format), or 0 if none.</td>
</tr>
<tr>
<td>typesend</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Output conversion function (binary format), or 0 if none.</td>
</tr>
<tr>
<td>typmodin</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Type modifier input function, or 0 if the type does not support modifiers.</td>
</tr>
<tr>
<td>typmodout</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Type modifier output function, or 0 to use the standard format.</td>
</tr>
<tr>
<td>typanalyze</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>Custom ANALYZE function, or 0 to use the standard function.</td>
</tr>
<tr>
<td>typalign</td>
<td>char</td>
<td></td>
<td>The alignment required when storing a value of this type. It applies to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>storage on disk as well as most representations of the value inside Greenplum Database. When multiple values are stored consecutively, such as in the representation of a complete row on disk, padding is inserted before a datum of this type so that it begins on the specified boundary. The alignment reference is the beginning of the first datum in the sequence. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>c = char alignment (no alignment needed).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>s = short alignment (2 bytes on most machines).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>i = int alignment (4 bytes on most machines).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>d = double alignment (8 bytes on many machines, but not all).</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>-------------</td>
<td>----------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>typstorage</td>
<td>char</td>
<td></td>
<td>For varlena types (those with <code>typlen = -1</code>) tells if the type is prepared for toasting and what the default strategy for attributes of this type should be. Possible values are: p: Value must always be stored plain. e: Value can be stored in a secondary relation (if relation has one, see <code>pg_class.reltoastrelid</code>). m: Value can be stored compressed inline. x: Value can be stored compressed inline or stored in secondary storage. Note that m columns can also be moved out to secondary storage, but only as a last resort (e and x columns are moved first).</td>
</tr>
<tr>
<td>typnotnull</td>
<td>boolean</td>
<td></td>
<td>Represents a not-null constraint on a type. Used for domains only.</td>
</tr>
<tr>
<td>typbasetype</td>
<td>oid</td>
<td><code>pg_type.oid</code></td>
<td>Identifies the type that a domain is based on. Zero if this type is not a domain.</td>
</tr>
<tr>
<td>typtypmod</td>
<td>int4</td>
<td></td>
<td>Domains use <code>typtypmod</code> to record the typmod to be applied to their base type (-1 if base type does not use a typmod). -1 if this type is not a domain.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>typndims</td>
<td>int4</td>
<td></td>
<td>The number of array dimensions for a domain that is an array (if typbasetype is an array type; the domain's typelem will match the base type's typelem). Zero for types other than array domains.</td>
</tr>
<tr>
<td>typdefaultbin</td>
<td>text</td>
<td></td>
<td>If not null, it is the nodeToString() representation of a default expression for the type. This is only used for domains.</td>
</tr>
<tr>
<td>typdefault</td>
<td>text</td>
<td></td>
<td>Null if the type has no associated default value. If not null, typdefault must contain a human-readable version of the default expression represented by typdefaultbin. If typdefaultbin is null and typdefault is not, then typdefault is the external representation of the type's default value, which may be fed to the type's input converter to produce a constant.</td>
</tr>
</tbody>
</table>

**pg_type_encoding**

The `pg_type_encoding` system catalog table contains the column storage type information.

**Table 182: pg_catalog.pg_type_encoding**

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>modifiers</th>
<th>storage</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typeid</td>
<td>oid</td>
<td>not null</td>
<td>plain</td>
<td>Foreign key to <code>pg_attribute</code></td>
</tr>
<tr>
<td>typoptions</td>
<td>text</td>
<td></td>
<td>extended</td>
<td>The actual options</td>
</tr>
</tbody>
</table>

**pg_user_mapping**

The `pg_user_mapping` catalog table stores the mappings from local users to remote users. You must have administrator privileges to view this catalog.
Table 183: pg_catalog.pg_user_mapping

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>umuser</td>
<td>oid</td>
<td>pg_authid.oid</td>
<td>OID of the local role being mapped, 0 if the user mapping is public</td>
</tr>
<tr>
<td>umserver</td>
<td>oid</td>
<td>pg_foreign_server.oid</td>
<td>The OID of the foreign server that contains this mapping</td>
</tr>
<tr>
<td>umoptions</td>
<td>text []</td>
<td></td>
<td>User mapping specific options, as &quot;keyword=value&quot; strings.</td>
</tr>
</tbody>
</table>

pg_window

The pg_window table stores information about window functions. Window functions are often used to compose complex OLAP (online analytical processing) queries. Window functions are applied to partitioned result sets within the scope of a single query expression. A window partition is a subset of rows returned by a query, as defined in a special OVER() clause. Typical window functions are rank, dense_rank, and row_number. Each entry in pg_window is an extension of an entry in pg_proc. The pg_proc entry carries the window function's name, input and output data types, and other information that is similar to ordinary functions.

Table 184: pg_catalog.pg_window

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>references</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>winfnoid</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>The OID in pg_proc of the window function.</td>
</tr>
<tr>
<td>winrequireorder</td>
<td>boolean</td>
<td></td>
<td>The window function requires its window specification to have an ORDER BY clause.</td>
</tr>
<tr>
<td>winallowframe</td>
<td>boolean</td>
<td></td>
<td>The window function permits its window specification to have a ROWS or RANGE framing clause.</td>
</tr>
<tr>
<td>winpeercount</td>
<td>boolean</td>
<td></td>
<td>The peer group row count is required to compute this window function, so the Window node implementation must 'look ahead' as necessary to make this available in its internal state.</td>
</tr>
<tr>
<td>wincount</td>
<td>boolean</td>
<td></td>
<td>The partition row count is required to compute this window function.</td>
</tr>
<tr>
<td>column</td>
<td>type</td>
<td>references</td>
<td>description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>winfunc</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>The OID in <code>pg_proc.oid</code> of a function to compute the value of an immediate-type window function.</td>
</tr>
<tr>
<td>winprefunc</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>The OID in <code>pg_proc.oid</code> of a preliminary window function to compute the partial value of a deferred-type window function.</td>
</tr>
<tr>
<td>winpretype</td>
<td>oid</td>
<td>pg_type.oid</td>
<td>The OID in <code>pg_type.oid</code> of the preliminary window function's result type.</td>
</tr>
<tr>
<td>winfinfunc</td>
<td>regproc</td>
<td>pg_proc.oid</td>
<td>The OID in <code>pg_proc.oid</code> of a function to compute the final value of a deferred-type window function from the partition row count and the result of <code>winprefunc</code>.</td>
</tr>
<tr>
<td>winkind</td>
<td>char</td>
<td></td>
<td>A character indicating membership of the window function in a class of related functions:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>w - ordinary window functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>n - NTILE functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>f - FIRST_VALUE functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>l - LAST_VALUE functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>g - LAG functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>d - LEAD functions</td>
</tr>
</tbody>
</table>
The gp_toolkit Administrative Schema

Greenplum Database provides an administrative schema called gp_toolkit that you can use to query the system catalogs, log files, and operating environment for system status information. The gp_toolkit schema contains a number of views that you can access using SQL commands. The gp_toolkit schema is accessible to all database users, although some objects may require superuser permissions. For convenience, you may want to add the gp_toolkit schema to your schema search path. For example:

```
=> ALTER ROLE myrole SET search_path TO myschema, gp_toolkit;
```

This documentation describes the most useful views in gp_toolkit. You may notice other objects (views, functions, and external tables) within the gp_toolkit schema that are not described in this documentation (these are supporting objects to the views described in this section).

**Warning**: Do not change database objects in the gp_toolkit schema. Do not create database objects in the schema. Changes to objects in the schema might affect the accuracy of administrative information returned by schema objects. Any changes made in the gp_toolkit schema are lost when the database is backed up and then restored with the gpcrondump and gpdbrestore utilities.

These are the categories for views in the gp_toolkit schema.

### Checking for Tables that Need Routine Maintenance

The following views can help identify tables that need routine table maintenance (VACUUM and/or ANALYZE).

- `gp_bloat_diag`
- `gp_stats_missing`

The VACUUM or VACUUM FULL command reclaims disk space occupied by deleted or obsolete rows. Because of the MVCC transaction concurrency model used in Greenplum Database, data rows that are deleted or updated still occupy physical space on disk even though they are not visible to any new transactions. Expired rows increase table size on disk and eventually slow down scans of the table.

The ANALYZE command collects column-level statistics needed by the query optimizer. Greenplum Database uses a cost-based query optimizer that relies on database statistics. Accurate statistics allow the query optimizer to better estimate selectivity and the number of rows retrieved by a query operation in order to choose the most efficient query plan.

#### gp_bloat_diag

This view shows regular heap-storage tables that have bloat (the actual number of pages on disk exceeds the expected number of pages given the table statistics). Tables that are bloated require a VACUUM or a VACUUM FULL in order to reclaim disk space occupied by deleted or obsolete rows. This view is accessible to all users, however non-superusers will only be able to see the tables that they have permission to access.

**Note**: For diagnostic functions that return append-optimized table information, see Checking Append-Optimized Tables.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bdirelid</td>
<td>Table object id.</td>
</tr>
<tr>
<td>Column</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>bdinspname</td>
<td>Schema name.</td>
</tr>
<tr>
<td>bdirelname</td>
<td>Table name.</td>
</tr>
<tr>
<td>bdiereлепages</td>
<td>Actual number of pages on disk.</td>
</tr>
<tr>
<td>bdirexp_pages</td>
<td>Expected number of pages given the table data.</td>
</tr>
<tr>
<td>bdidiag</td>
<td>Bloat diagnostic message.</td>
</tr>
</tbody>
</table>

**gp_stats_missing**

This view shows tables that do not have statistics and therefore may require an `ANALYZE` be run on the table.

**Table 186: gp_stats_missing view**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>smischema</td>
<td>Schema name.</td>
</tr>
<tr>
<td>smitable</td>
<td>Table name.</td>
</tr>
<tr>
<td>smisize</td>
<td>Does this table have statistics? False if the table does not have row count and row sizing statistics recorded in the system catalog, which may indicate that the table needs to be analyzed. This will also be false if the table does not contain any rows. For example, the parent tables of partitioned tables are always empty and will always return a false result.</td>
</tr>
<tr>
<td>smicols</td>
<td>Number of columns in the table.</td>
</tr>
<tr>
<td>smirecs</td>
<td>Number of rows in the table.</td>
</tr>
</tbody>
</table>

**Checking for Locks**

When a transaction accesses a relation (such as a table), it acquires a lock. Depending on the type of lock acquired, subsequent transactions may have to wait before they can access the same relation. For more information on the types of locks, see "Managing Data" in the *Greenplum Database Administrator Guide*. Greenplum Database resource queues (used for resource management) also use locks to control the admission of queries into the system.

The `gp_locks_*` family of views can help diagnose queries and sessions that are waiting to access an object due to a lock.

- `gp_locks_on_relation`
- `gp_locks_on_resqueue`

**gp_locks_on_relation**

This view shows any locks currently being held on a relation, and the associated session information about the query associated with the lock. For more information on the types of locks, see "Managing Data" in the *Greenplum Database Administrator Guide*. This view is accessible to all users, however non-superusers will only be able to see the locks for relations that they have permission to access.
Table 187: gp_locks_on_relation view

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lorlocktype</td>
<td>Type of the lockable object: relation, extend, page, tuple, transactionid, object, userlock, resource queue, or advisory</td>
</tr>
<tr>
<td>lordatabase</td>
<td>Object ID of the database in which the object exists, zero if the object is a shared object.</td>
</tr>
<tr>
<td>lorrelname</td>
<td>The name of the relation.</td>
</tr>
<tr>
<td>lorrelation</td>
<td>The object ID of the relation.</td>
</tr>
<tr>
<td>lortransaction</td>
<td>The transaction ID that is affected by the lock.</td>
</tr>
<tr>
<td>lorpid</td>
<td>Process ID of the server process holding or awaiting this lock. NULL if the lock is held by a prepared transaction.</td>
</tr>
<tr>
<td>lormode</td>
<td>Name of the lock mode held or desired by this process.</td>
</tr>
<tr>
<td>lorgranted</td>
<td>Displays whether the lock is granted (true) or not granted (false).</td>
</tr>
<tr>
<td>lorcurrentquery</td>
<td>The current query in the session.</td>
</tr>
</tbody>
</table>

gp_locks_on_resqueue

This view shows any locks currently being held on a resource queue, and the associated session information about the query associated with the lock. This view is accessible to all users, however non-superusers will only be able to see the locks associated with their own sessions.

Table 188: gp_locks_on_resqueue view

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lorusename</td>
<td>Name of the user executing the session.</td>
</tr>
<tr>
<td>lorrsqname</td>
<td>The resource queue name.</td>
</tr>
<tr>
<td>lorlocktype</td>
<td>Type of the lockable object: resource queue</td>
</tr>
<tr>
<td>lorobjid</td>
<td>The ID of the locked transaction.</td>
</tr>
<tr>
<td>lortransaction</td>
<td>The ID of the transaction that is affected by the lock.</td>
</tr>
<tr>
<td>lorpid</td>
<td>The process ID of the transaction that is affected by the lock.</td>
</tr>
<tr>
<td>lormode</td>
<td>The name of the lock mode held or desired by this process.</td>
</tr>
<tr>
<td>lorgranted</td>
<td>Displays whether the lock is granted (true) or not granted (false).</td>
</tr>
<tr>
<td>lorwaiting</td>
<td>Displays whether or not the session is waiting.</td>
</tr>
</tbody>
</table>
Checking Append-Optimized Tables

The `gp_toolkit` schema includes a set of diagnostic functions you can use to investigate the state of append-optimized tables.

When an append-optimized table (or column-oriented append-optimized table) is created, another table is implicitly created, containing metadata about the current state of the table. The metadata includes information such as the number of records in each of the table’s segments.

Append-optimized tables may have non-visible rows—rows that have been updated or deleted, but remain in storage until the table is compacted using `VACUUM`. The hidden rows are tracked using an auxiliary visibility map table, or visimap.

The following functions let you access the metadata for append-optimized and column-oriented tables and view non-visible rows. Some of the functions have two versions: one that takes the `oid` of the table, and one that takes the name of the table. The latter version has "_name" appended to the function name.

__gp_aovisimap_compaction_info(oid)__. 
This function displays compaction information for an append-optimized table. The information is for the on-disk data files on Greenplum Database segments that store the table data. You can use the information to determine the data files that will be compacted by a `VACUUM` operation on an append-optimized table.

Note: Until a `VACUUM` operation deletes the row from the data file, deleted or updated data rows occupy physical space on disk even though they are hidden to new transactions. The configuration parameter `gp_appendonly_compaction` controls the functionality of the `VACUUM` command.

This table describes the `__gp_aovisimap_compaction_info` function output table.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>Greenplum Database segment ID.</td>
</tr>
<tr>
<td>datafile</td>
<td>ID of the data file on the segment.</td>
</tr>
<tr>
<td>compaction_possible</td>
<td>The value is either <code>t</code> or <code>f</code>. The value <code>t</code> indicates that the data in data file be compacted when a <code>VACUUM</code> operation is performed. The server configuration parameter <code>gp_appendonly_compaction_threshold</code> affects this value.</td>
</tr>
<tr>
<td>hidden_tupcount</td>
<td>In the data file, the number of hidden (deleted or updated) rows.</td>
</tr>
<tr>
<td>total_tupcount</td>
<td>In the data file, the total number of rows.</td>
</tr>
<tr>
<td>percent_hidden</td>
<td>In the data file, the ratio (as a percentage) of hidden (deleted or updated) rows to total rows.</td>
</tr>
</tbody>
</table>

__gp_aoseg_name('table_name')__. 
This function returns metadata information contained in the append-optimized table’s on-disk segment file.
Table 190: __gp_aoseg_name output table

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segno</td>
<td>The file segment number.</td>
</tr>
<tr>
<td>eof</td>
<td>The effective end of file for this file segment.</td>
</tr>
<tr>
<td>tupcount</td>
<td>The total number of tuples in the segment, including invisible tuples.</td>
</tr>
<tr>
<td>varblockcount</td>
<td>The total number of varblocks in the file segment.</td>
</tr>
<tr>
<td>eof_uncompressed</td>
<td>The end of file if the file segment were uncompressed.</td>
</tr>
<tr>
<td>modcount</td>
<td>The number of data modification operations.</td>
</tr>
<tr>
<td>state</td>
<td>The state of the file segment. Indicates if the segment is active or ready to be dropped after compaction.</td>
</tr>
</tbody>
</table>

__gp_aoseg_history(oid)

This function returns metadata information contained in the append-optimized table's on-disk segment file. It displays all different versions (heap tuples) of the aoseg meta information. The data is complex, but users with a deep understanding of the system may find it useful for debugging.

The input argument is the oid of the append-optimized table.

Call __gp_aoseg_history_name('table_name') to get the same result with the table name as an argument.

Table 191: __gp_aoseg_history output table

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp_tid</td>
<td>The id of the tuple.</td>
</tr>
<tr>
<td>gp_xmin</td>
<td>The id of the earliest transaction.</td>
</tr>
<tr>
<td>gp_xmin_status</td>
<td>Status of the gp_xmin transaction.</td>
</tr>
<tr>
<td>gp_xmin_commit</td>
<td>The commit distribution id of the gp_xmin transaction.</td>
</tr>
<tr>
<td>gp_xmax</td>
<td>The id of the latest transaction.</td>
</tr>
<tr>
<td>gp_xmax_status</td>
<td>The status of the latest transaction.</td>
</tr>
<tr>
<td>gp_xmax_commit</td>
<td>The commit distribution id of the gp_xmax transaction.</td>
</tr>
<tr>
<td>gp_command_id</td>
<td>The id of the query command.</td>
</tr>
<tr>
<td>gp_infomask</td>
<td>A bitmap containing state information.</td>
</tr>
<tr>
<td>gp_update_tid</td>
<td>The ID of the newer tuple if the row is updated.</td>
</tr>
<tr>
<td>gp_visibility</td>
<td>The tuple visibility status.</td>
</tr>
<tr>
<td>segno</td>
<td>The number of the segment in the segment file.</td>
</tr>
<tr>
<td>tupcount</td>
<td>The number of tuples, including hidden tuples.</td>
</tr>
<tr>
<td>eof</td>
<td>The effective end of file for the segment.</td>
</tr>
<tr>
<td>Column</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>eof_uncompressed</td>
<td>The end of file for the segment if data were uncompressed.</td>
</tr>
<tr>
<td>modcount</td>
<td>A count of data modifications.</td>
</tr>
<tr>
<td>state</td>
<td>The status of the segment.</td>
</tr>
</tbody>
</table>

**__gp_aocsseg(oid)***

This function returns metadata information contained in a column-oriented append-optimized table’s on-disk segment file, excluding non-visible rows. Each row describes a segment for a column in the table.

The input argument is the oid of a column-oriented append-optimized table. Call as __gp_aocsseg_name('table_name') to get the same result with the table name as an argument.

Table 192: __gp_aocsseg(oid) output table

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp_tid</td>
<td>The table id.</td>
</tr>
<tr>
<td>segno</td>
<td>The segment number.</td>
</tr>
<tr>
<td>column_num</td>
<td>The column number.</td>
</tr>
<tr>
<td>physical_segno</td>
<td>The number of the segment in the segment file.</td>
</tr>
<tr>
<td>tupcount</td>
<td>The number of rows in the segment, excluding hidden tuples.</td>
</tr>
<tr>
<td>eof</td>
<td>The effective end of file for the segment.</td>
</tr>
<tr>
<td>eof_uncompressed</td>
<td>The end of file for the segment if the data were uncompressed.</td>
</tr>
<tr>
<td>modcount</td>
<td>A count of data modification operations for the segment.</td>
</tr>
<tr>
<td>state</td>
<td>The status of the segment.</td>
</tr>
</tbody>
</table>

**__gp_aocsseg_history(oid)***

This function returns metadata information contained in a column-oriented append-optimized table’s on-disk segment file. Each row describes a segment for a column in the table. The data is complex, but users with a deep understanding of the system may find it useful for debugging.

The input argument is the oid of a column-oriented append-optimized table. Call as __gp_aocsseg_history_name('table_name') to get the same result with the table name as argument.

Table 193: __gp_aocsseg_history output table

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp_tid</td>
<td>The oid of the tuple.</td>
</tr>
<tr>
<td>gp_xmin</td>
<td>The earliest transaction.</td>
</tr>
<tr>
<td>gp_xmin_status</td>
<td>The status of the gp_xmin transaction.</td>
</tr>
<tr>
<td>gp_xmin_</td>
<td>Text representation of gp_xmin.</td>
</tr>
</tbody>
</table>
### __gp_aovisimap(oid)\n
This function returns the tuple id, the segment file, and the row number of each non-visible tuple according to the visibility map.

The input argument is the oid of an append-optimized table.

Use __gp_aovisimap_name('table_name') to get the same result with the table name as argument.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tid</td>
<td>The tuple id.</td>
</tr>
<tr>
<td>segno</td>
<td>The number of the segment file.</td>
</tr>
<tr>
<td>row_num</td>
<td>The row number of a row that has been deleted or updated.</td>
</tr>
</tbody>
</table>

### __gp_aovisimap_hidden_info(oid)\n
This function returns the numbers of hidden and visible tuples in the segment files for an append-optimized table.

The input argument is the oid of the append-optimized table.

Call __gp_aovisimap_hidden_info_name('table_name') to get the same result with a table name argument.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segno</td>
<td>The number of the segment file.</td>
</tr>
<tr>
<td>hidden_tupcount</td>
<td>The number of hidden tuples in the segment file.</td>
</tr>
</tbody>
</table>
Column | Description
--- | ---
total_tupcount | The total number of tuples in the segment file.

__gp_aovisimap_entry(oid)\n
This function returns information about each visibility map entry for the table.

The input argument is the oid of an append-optimized table.

Call __gp_aovisimap_entry_name('table_name') to get the same result with a table name argument.

Table 194: __gp_aovisimap_entry output table

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segno</td>
<td>Segment number of the visibility map entry.</td>
</tr>
<tr>
<td>first_row_num</td>
<td>The first row number of the entry.</td>
</tr>
<tr>
<td>hidden_tupcount</td>
<td>The number of hidden tuples in the entry.</td>
</tr>
<tr>
<td>bitmap</td>
<td>A text representation of the visibility bitmap.</td>
</tr>
</tbody>
</table>

**Viewing Greenplum Database Server Log Files**

Each component of a Greenplum Database system (master, standby master, primary segments, and mirror segments) keeps its own server log files. The `gp_log_*` family of views allows you to issue SQL queries against the server log files to find particular entries of interest. The use of these views require superuser permissions.

- `gp_log_command_timings`
- `gp_log_database`
- `gp_log_master_concise`
- `gp_log_system`

**gp_log_command_timings**

This view uses an external table to read the log files on the master and report the execution time of SQL commands executed in a database session. The use of this view requires superuser permissions.

Table 195: gp_log_command_timings view

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logsession</td>
<td>The session identifier (prefixed with &quot;con&quot;).</td>
</tr>
<tr>
<td>logcmdcount</td>
<td>The command number within a session (prefixed with &quot;cmd&quot;).</td>
</tr>
<tr>
<td>logdatabase</td>
<td>The name of the database.</td>
</tr>
<tr>
<td>loguser</td>
<td>The name of the database user.</td>
</tr>
<tr>
<td>logpid</td>
<td>The process id (prefixed with &quot;p&quot;).</td>
</tr>
<tr>
<td>logtimemin</td>
<td>The time of the first log message for this command.</td>
</tr>
<tr>
<td>logtimemax</td>
<td>The time of the last log message for this command.</td>
</tr>
<tr>
<td>logduration</td>
<td>Statement duration from start to end time.</td>
</tr>
</tbody>
</table>
gp_log_database

This view uses an external table to read the server log files of the entire Greenplum system (master, segments, and mirrors) and lists log entries associated with the current database. Associated log entries can be identified by the session id (logsession) and command id (logcmdcount). The use of this view requires superuser permissions.

Table 196: gp_log_database view

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logtime</td>
<td>The timestamp of the log message.</td>
</tr>
<tr>
<td>loguser</td>
<td>The name of the database user.</td>
</tr>
<tr>
<td>logdatabase</td>
<td>The name of the database.</td>
</tr>
<tr>
<td>logpid</td>
<td>The associated process id (prefixed with &quot;p&quot;).</td>
</tr>
<tr>
<td>logthread</td>
<td>The associated thread count (prefixed with &quot;th&quot;).</td>
</tr>
<tr>
<td>loghost</td>
<td>The segment or master host name.</td>
</tr>
<tr>
<td>logport</td>
<td>The segment or master port.</td>
</tr>
<tr>
<td>logsessiontime</td>
<td>Time session connection was opened.</td>
</tr>
<tr>
<td>logtransaction</td>
<td>Global transaction id.</td>
</tr>
<tr>
<td>logsession</td>
<td>The session identifier (prefixed with &quot;con&quot;).</td>
</tr>
<tr>
<td>logcmdcount</td>
<td>The command number within a session (prefixed with &quot;cmd&quot;).</td>
</tr>
<tr>
<td>logsegment</td>
<td>The segment content identifier (prefixed with &quot;seg&quot; for primary or &quot;mir&quot; for mirror. The master always has a content id of -1).</td>
</tr>
<tr>
<td>logslice</td>
<td>The slice id (portion of the query plan being executed).</td>
</tr>
<tr>
<td>logdistxact</td>
<td>Distributed transaction id.</td>
</tr>
<tr>
<td>loglocalxact</td>
<td>Local transaction id.</td>
</tr>
<tr>
<td>logsubxact</td>
<td>Subtransaction id.</td>
</tr>
<tr>
<td>logseverity</td>
<td>LOG, ERROR, FATAL, PANIC, DEBUG1 or DEBUG2.</td>
</tr>
<tr>
<td>logstate</td>
<td>SQL state code associated with the log message.</td>
</tr>
<tr>
<td>logmessage</td>
<td>Log or error message text.</td>
</tr>
<tr>
<td>logdetail</td>
<td>Detail message text associated with an error message.</td>
</tr>
<tr>
<td>loghint</td>
<td>Hint message text associated with an error message.</td>
</tr>
<tr>
<td>logquery</td>
<td>The internally-generated query text.</td>
</tr>
<tr>
<td>logquerypos</td>
<td>The cursor index into the internally-generated query text.</td>
</tr>
<tr>
<td>Column</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>logcontext</td>
<td>The context in which this message gets generated.</td>
</tr>
<tr>
<td>logdebug</td>
<td>Query string with full detail for debugging.</td>
</tr>
<tr>
<td>logcursorpos</td>
<td>The cursor index into the query string.</td>
</tr>
<tr>
<td>logfunction</td>
<td>The function in which this message is generated.</td>
</tr>
<tr>
<td>logfile</td>
<td>The log file in which this message is generated.</td>
</tr>
<tr>
<td>logline</td>
<td>The line in the log file in which this message is generated.</td>
</tr>
<tr>
<td>logstack</td>
<td>Full text of the stack trace associated with this message.</td>
</tr>
</tbody>
</table>

**gp_log_master_concise**

This view uses an external table to read a subset of the log fields from the master log file. The use of this view requires superuser permissions.

**Table 197: gp_log_master_concise view**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logtime</td>
<td>The timestamp of the log message.</td>
</tr>
<tr>
<td>logdatabase</td>
<td>The name of the database.</td>
</tr>
<tr>
<td>logsession</td>
<td>The session identifier (prefixed with &quot;con&quot;).</td>
</tr>
<tr>
<td>logcmdcount</td>
<td>The command number within a session (prefixed with &quot;cmd&quot;).</td>
</tr>
<tr>
<td>logmessage</td>
<td>Log or error message text.</td>
</tr>
</tbody>
</table>

**gp_log_system**

This view uses an external table to read the server log files of the entire Greenplum system (master, segments, and mirrors) and lists all log entries. Associated log entries can be identified by the session id (logsession) and command id (logcmdcount). The use of this view requires superuser permissions.

**Table 198: gp_log_system view**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logtime</td>
<td>The timestamp of the log message.</td>
</tr>
<tr>
<td>loguser</td>
<td>The name of the database user.</td>
</tr>
<tr>
<td>logdatabase</td>
<td>The name of the database.</td>
</tr>
<tr>
<td>logpid</td>
<td>The associated process id (prefixed with &quot;p&quot;).</td>
</tr>
<tr>
<td>logthread</td>
<td>The associated thread count (prefixed with &quot;th&quot;).</td>
</tr>
<tr>
<td>loghost</td>
<td>The segment or master host name.</td>
</tr>
<tr>
<td>logport</td>
<td>The segment or master port.</td>
</tr>
<tr>
<td>logsessiontime</td>
<td>Time session connection was opened.</td>
</tr>
</tbody>
</table>
### Checking Server Configuration Files

Each component of a Greenplum Database system (master, standby master, primary segments, and mirror segments) has its own server configuration file (`postgresql.conf`). The following `gp_toolkit` objects can be used to check parameter settings across all primary `postgresql.conf` files in the system:

- `gp_param_setting('parameter_name')`
- `gp_param_settings_seg_value_diffs`
gp_param_setting('parameter_name')

This function takes the name of a server configuration parameter and returns the `postgresql.conf` value for the master and each active segment. This function is accessible to all users.

Table 199: gp_param_setting('parameter_name') function

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>paramsegment</td>
<td>The segment content id (only active segments are shown). The master content id is always -1.</td>
</tr>
<tr>
<td>paramname</td>
<td>The name of the parameter.</td>
</tr>
<tr>
<td>paramvalue</td>
<td>The value of the parameter.</td>
</tr>
</tbody>
</table>

Example:

```
SELECT * FROM gp_param_setting('max_connections');
```

gp_param_settings_seg_value_diffs

Server configuration parameters that are classified as `local` parameters (meaning each segment gets the parameter value from its own `postgresql.conf` file), should be set identically on all segments. This view shows local parameter settings that are inconsistent. Parameters that are supposed to have different values (such as `port`) are not included. This view is accessible to all users.

Table 200: gp_param_settings_seg_value_diffs view

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psdname</td>
<td>The name of the parameter.</td>
</tr>
<tr>
<td>psdvalue</td>
<td>The value of the parameter.</td>
</tr>
<tr>
<td>psdcount</td>
<td>The number of segments that have this value.</td>
</tr>
</tbody>
</table>

Checking for Failed Segments

The `gp_pgdatabase_invalid` view can be used to check for down segments.

gp_pgdatabase_invalid

This view shows information about segments that are marked as down in the system catalog. This view is accessible to all users.

Table 201: gp_pgdatabase_invalid view

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pgdbidbid</td>
<td>The segment dbid. Every segment has a unique dbid.</td>
</tr>
<tr>
<td>pgdbiisprimary</td>
<td>Is the segment currently acting as the primary (active) segment? (t or f)</td>
</tr>
<tr>
<td>pgdbiccontent</td>
<td>The content id of this segment. A primary and mirror will have the same content id.</td>
</tr>
</tbody>
</table>
Checking Resource Queue Activity and Status

The purpose of resource queues is to limit the number of active queries in the system at any given time in order to avoid exhausting system resources such as memory, CPU, and disk I/O. All database users are assigned to a resource queue, and every statement submitted by a user is first evaluated against the resource queue limits before it can run. The gp_resq_* family of views can be used to check the status of statements currently submitted to the system through their respective resource queue. Note that statements issued by superusers are exempt from resource queuing.

- gp_resq_activity
- gp_resq_activity_by_queue
- gp_resq_priority_statement
- gp_resq_role
- gp_resqueue_status

**gp_resq_activity**

For the resource queues that have active workload, this view shows one row for each active statement submitted through a resource queue. This view is accessible to all users.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resqprocpid</td>
<td>Process ID assigned to this statement (on the master).</td>
</tr>
<tr>
<td>resqrole</td>
<td>User name.</td>
</tr>
<tr>
<td>resqoid</td>
<td>Resource queue object id.</td>
</tr>
<tr>
<td>resqname</td>
<td>Resource queue name.</td>
</tr>
<tr>
<td>resqstart</td>
<td>Time statement was issued to the system.</td>
</tr>
<tr>
<td>resqstatus</td>
<td>Status of statement: running, waiting or cancelled.</td>
</tr>
</tbody>
</table>

**gp_resq_activity_by_queue**

For the resource queues that have active workload, this view shows a summary of queue activity. This view is accessible to all users.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resqoid</td>
<td>Resource queue object id.</td>
</tr>
<tr>
<td>resqname</td>
<td>Resource queue name.</td>
</tr>
<tr>
<td>resqlast</td>
<td>Time of the last statement issued to the queue.</td>
</tr>
<tr>
<td>Column</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>resqstatus</td>
<td>Status of last statement: running, waiting or cancelled.</td>
</tr>
<tr>
<td>resqtotal</td>
<td>Total statements in this queue.</td>
</tr>
</tbody>
</table>

**gp_resq_priority_statement**

This view shows the resource queue priority, session ID, and other information for all statements currently running in the Greenplum Database system. This view is accessible to all users.

**Table 204: gp_resq_priority_statement view**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rqpdatname</td>
<td>The database name that the session is connected to.</td>
</tr>
<tr>
<td>rqpusename</td>
<td>The user who issued the statement.</td>
</tr>
<tr>
<td>rqpsession</td>
<td>The session ID.</td>
</tr>
<tr>
<td>rqpcommand</td>
<td>The number of the statement within this session (the command id and session id uniquely identify a statement).</td>
</tr>
<tr>
<td>rqppriority</td>
<td>The resource queue priority for this statement (MAX, HIGH, MEDIUM, LOW).</td>
</tr>
<tr>
<td>rqpweight</td>
<td>An integer value associated with the priority of this statement.</td>
</tr>
<tr>
<td>rqpquery</td>
<td>The query text of the statement.</td>
</tr>
</tbody>
</table>

**gp_resq_role**

This view shows the resource queues associated with a role. This view is accessible to all users.

**Table 205: gp_resq_role view**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rrrolname</td>
<td>Role (user) name.</td>
</tr>
<tr>
<td>rrssqname</td>
<td>The resource queue name assigned to this role. If a role has not been explicitly assigned to a resource queue, it will be in the default resource queue (pg_default).</td>
</tr>
</tbody>
</table>

**gp_resqueue_status**

This view allows administrators to see status and activity for a resource queue. It shows how many queries are waiting to run and how many queries are currently active in the system from a particular resource queue.
Table 206: gp_resqueue_status view

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queueid</td>
<td>The ID of the resource queue.</td>
</tr>
<tr>
<td>rsqname</td>
<td>The name of the resource queue.</td>
</tr>
</tbody>
</table>
| rsqcountlimit | The active query threshold of the resource queue.  
A value of -1 means no limit. |
| rsqcountvalue | The number of active query slots currently being used in the resource queue. |
| rsqcostlimit | The query cost threshold of the resource queue.  
A value of -1 means no limit. |
| rsqcostvalue | The total cost of all statements currently in the resource queue.           |
| rsqmemorylimit | The memory limit for the resource queue.                                   |
| rsqmemoryvalue | The total memory used by all statements currently in the resource queue.   |
| rsqwaiters | The number of statements currently waiting in the resource queue.           |
| rsqholders | The number of statements currently running on the system from this resource queue. |

Checking Query Disk Spill Space Usage

The gp_workfile_* views show information about all the queries that are currently using disk spill space. Greenplum Database creates work files on disk if it does not have sufficient memory to execute the query in memory. This information can be used for troubleshooting and tuning queries. The information in the views can also be used to specify the values for the Greenplum Database configuration parameters gp_workfile_limit_per_query and gp_workfile_limit_per_segment.

- gp_workfile_entries
- gp_workfile_usage_per_query
- gp_workfile_usage_per_segment

gp_workfile_entries

This view contains one row for each operator using disk space for workfiles on a segment at the current time. The view is accessible to all users, however non-superusers only to see information for the databases that they have permission to access.

Table 207: gp_workfile_entries

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>References</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command_cnt</td>
<td>integer</td>
<td></td>
<td>Command ID of the query.</td>
</tr>
<tr>
<td>content</td>
<td>smallint</td>
<td></td>
<td>The content identifier for a segment instance.</td>
</tr>
<tr>
<td>current_query</td>
<td>text</td>
<td></td>
<td>Current query that the process is running.</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>References</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>datname</td>
<td>name</td>
<td></td>
<td>Greenplum database name.</td>
</tr>
<tr>
<td>directory</td>
<td>text</td>
<td></td>
<td>Path to the work file.</td>
</tr>
<tr>
<td>optype</td>
<td>text</td>
<td></td>
<td>The query operator type that created the work file.</td>
</tr>
<tr>
<td>procpid</td>
<td>integer</td>
<td></td>
<td>Process ID of the server process.</td>
</tr>
<tr>
<td>sess_id</td>
<td>integer</td>
<td></td>
<td>Session ID.</td>
</tr>
<tr>
<td>size</td>
<td>bigint</td>
<td></td>
<td>The size of the work file in bytes.</td>
</tr>
<tr>
<td>numfiles</td>
<td>bigint</td>
<td></td>
<td>The number of files created.</td>
</tr>
<tr>
<td>slice</td>
<td>smallint</td>
<td></td>
<td>The query plan slice.</td>
</tr>
<tr>
<td>state</td>
<td>text</td>
<td></td>
<td>The state of the query that created the work file.</td>
</tr>
<tr>
<td>usename</td>
<td>name</td>
<td></td>
<td>Role name.</td>
</tr>
<tr>
<td>workmem</td>
<td>integer</td>
<td></td>
<td>The amount of memory allocated to the operator in KB.</td>
</tr>
</tbody>
</table>

**gp_workfile_usage_per_query**

This view contains one row for each query using disk space for workfiles on a segment at the current time. The view is accessible to all users, however non-superusers only to see information for the databases that they have permission to access.

**Table 208: gp_workfile_usage_per_query**

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>References</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command_cnt</td>
<td>integer</td>
<td></td>
<td>Command ID of the query.</td>
</tr>
<tr>
<td>content</td>
<td>smallint</td>
<td></td>
<td>The content identifier for a segment instance.</td>
</tr>
<tr>
<td>current_query</td>
<td>text</td>
<td></td>
<td>Current query that the process is running.</td>
</tr>
<tr>
<td>datname</td>
<td>name</td>
<td></td>
<td>Greenplum database name.</td>
</tr>
<tr>
<td>procpid</td>
<td>integer</td>
<td></td>
<td>Process ID of the server process.</td>
</tr>
</tbody>
</table>
### Column	Type	References	Description

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>References</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sess_id</td>
<td>integer</td>
<td></td>
<td>Session ID.</td>
</tr>
<tr>
<td>size</td>
<td>bigint</td>
<td></td>
<td>The size of the work file in bytes.</td>
</tr>
<tr>
<td>numfiles</td>
<td>bigint</td>
<td></td>
<td>The number of files created.</td>
</tr>
<tr>
<td>state</td>
<td>text</td>
<td></td>
<td>The state of the query that created the work file.</td>
</tr>
<tr>
<td>usename</td>
<td>name</td>
<td></td>
<td>Role name.</td>
</tr>
</tbody>
</table>

**gp_workfile_usage_per_segment**

This view contains one row for each segment. Each row displays the total amount of disk space used for workfiles on the segment at the current time. The view is accessible to all users, however non-superusers only to see information for the databases that they have permission to access.

Table 209: gp_workfile_usage_per_segment

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>References</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>smallint</td>
<td></td>
<td>The content identifier for a segment instance.</td>
</tr>
<tr>
<td>size</td>
<td>bigint</td>
<td></td>
<td>The total size of the work files on a segment.</td>
</tr>
<tr>
<td>numfiles</td>
<td>bigint</td>
<td></td>
<td>The number of files created.</td>
</tr>
</tbody>
</table>

**Viewing Users and Groups (Roles)**

It is frequently convenient to group users (roles) together to ease management of object privileges: that way, privileges can be granted to, or revoked from, a group as a whole. In Greenplum Database this is done by creating a role that represents the group, and then granting membership in the group role to individual user roles.

The `gp_roles_assigned` view can be used to see all of the roles in the system, and their assigned members (if the role is also a group role).

**gp_roles_assigned**

This view shows all of the roles in the system, and their assigned members (if the role is also a group role). This view is accessible to all users.

Table 210: gp_roles_assigned view

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>raroleid</td>
<td>The role object ID. If this role has members (users), it is considered a group role.</td>
</tr>
<tr>
<td>rarolename</td>
<td>The role (user or group) name.</td>
</tr>
</tbody>
</table>
Column Description
ramemberid The role object ID of the role that is a member of this role.
ramembername Name of the role that is a member of this role.

Checking Database Object Sizes and Disk Space

The `gp_size_*` family of views can be used to determine the disk space usage for a distributed Greenplum Database, schema, table, or index. The following views calculate the total size of an object across all primary segments (mirrors are not included in the size calculations).

- `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`

The table and index sizing views list the relation by object ID (not by name). To check the size of a table or index by name, you must look up the relation name (`relname`) in the `pg_class` table. For example:

```sql
SELECT relname as name, sotdsize as size, sotdtoastsize as toast, sotdadditionalsize as other
FROM gp_size_of_table_disk as sotd, pg_class
WHERE sotd.sotdoid=pg_class.oid ORDER BY relname;
```

`gp_size_of_all_table_indexes`

This view shows the total size of all indexes for a table. This view is accessible to all users, however non-superusers will only be able to see relations that they have permission to access.

Table 211: `gp_size_of_all_table_indexes` view

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>soatioid</td>
<td>The object ID of the table</td>
</tr>
<tr>
<td>soatisize</td>
<td>The total size of all table indexes in bytes</td>
</tr>
<tr>
<td>soatischemaname</td>
<td>The schema name</td>
</tr>
<tr>
<td>soatitablename</td>
<td>The table name</td>
</tr>
</tbody>
</table>

`gp_size_of_database`

This view shows the total size of a database. This view is accessible to all users, however non-superusers will only be able to see databases that they have permission to access.
Table 212: gp_size_of_database view

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sodddatname</td>
<td>The name of the database</td>
</tr>
<tr>
<td>sodddatsize</td>
<td>The size of the database in bytes</td>
</tr>
</tbody>
</table>

**gp_size_of_index**

This view shows the total size of an index. This view is accessible to all users, however non-superusers will only be able to see relations that they have permission to access.

Table 213: gp_size_of_index view

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>soioid</td>
<td>The object ID of the index</td>
</tr>
<tr>
<td>soitableoid</td>
<td>The object ID of the table to which the index belongs</td>
</tr>
<tr>
<td>soisize</td>
<td>The size of the index in bytes</td>
</tr>
<tr>
<td>soiindexschemaname</td>
<td>The name of the index schema</td>
</tr>
<tr>
<td>soiindexname</td>
<td>The name of the index</td>
</tr>
<tr>
<td>soitableschemaname</td>
<td>The name of the table schema</td>
</tr>
<tr>
<td>soitablename</td>
<td>The name of the table</td>
</tr>
</tbody>
</table>

**gp_size_of_partition_and_indexes_disk**

This view shows the size on disk of partitioned child tables and their indexes. This view is accessible to all users, however non-superusers will only be able to see relations that they have permission to access.

Table 214: gp_size_of_partition_and_indexes_disk view

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sopaidparentoid</td>
<td>The object ID of the parent table</td>
</tr>
<tr>
<td>sopaidpartitionoid</td>
<td>The object ID of the partition table</td>
</tr>
<tr>
<td>sopaidpartitiontablesiz e</td>
<td>The partition table size in bytes</td>
</tr>
<tr>
<td>sopaidpartitionindexessize</td>
<td>The total size of all indexes on this partition</td>
</tr>
<tr>
<td>Sopaidparentschemaname</td>
<td>The name of the parent schema</td>
</tr>
<tr>
<td>Sopaidparenttablename</td>
<td>The name of the parent table</td>
</tr>
<tr>
<td>Sopaidpartitionschemaname</td>
<td>The name of the partition schema</td>
</tr>
<tr>
<td>sopaidpartitiontablename</td>
<td>The name of the partition table</td>
</tr>
</tbody>
</table>

**gp_size_of_schema_disk**

This view shows schema sizes for the public schema and the user-created schemas in the current database. This view is accessible to all users, however non-superusers will be able to see only the schemas that they have permission to access.
Table 215: gp_size_of_schema_disk view

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sosdnsp</td>
<td>The name of the schema</td>
</tr>
<tr>
<td>sosdschematablesize</td>
<td>The total size of tables in the schema in bytes</td>
</tr>
<tr>
<td>sosdschmaidxsize</td>
<td>The total size of indexes in the schema in bytes</td>
</tr>
</tbody>
</table>

gp_size_of_table_and_indexes_disk

This view shows the size on disk of tables and their indexes. This view is accessible to all users, however non-superusers will only be able to see relations that they have permission to access.

Table 216: gp_size_of_table_and_indexes_disk view

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sotaidoid</td>
<td>The object ID of the parent table</td>
</tr>
<tr>
<td>sotaidtablesize</td>
<td>The disk size of the table</td>
</tr>
<tr>
<td>sotaididxsize</td>
<td>The total size of all indexes on the table</td>
</tr>
<tr>
<td>sotaidschemaname</td>
<td>The name of the schema</td>
</tr>
<tr>
<td>sotaidtablename</td>
<td>The name of the table</td>
</tr>
</tbody>
</table>

gp_size_of_table_and_indexes_licensing

This view shows the total size of tables and their indexes for licensing purposes. The use of this view requires superuser permissions.

Table 217: gp_size_of_table_and_indexes_licensing view

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sotailoid</td>
<td>The object ID of the table</td>
</tr>
<tr>
<td>sotailtablesize</td>
<td>The total disk size of the table</td>
</tr>
<tr>
<td>sotailtablesizeuncompressed</td>
<td>If the table is a compressed append-optimized table, shows the uncompressed table size in bytes.</td>
</tr>
<tr>
<td>sotailindexessize</td>
<td>The total size of all indexes in the table</td>
</tr>
<tr>
<td>sotailschemaname</td>
<td>The schema name</td>
</tr>
<tr>
<td>sotailtablename</td>
<td>The table name</td>
</tr>
</tbody>
</table>

gp_size_of_table_disk

This view shows the size of a table on disk. This view is accessible to all users, however non-superusers will only be able to see tables that they have permission to access.

Table 218: gp_size_of_table_disk view

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sotdoid</td>
<td>The object ID of the table</td>
</tr>
<tr>
<td>Column</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>sotdsize</td>
<td>The size of the table in bytes. The size is only the main table size. The size does not include auxiliary objects such as oversized (toast) attributes, or additional storage objects for AO tables.</td>
</tr>
<tr>
<td>sotdtoastsize</td>
<td>The size of the TOAST table (oversized attribute storage), if there is one.</td>
</tr>
<tr>
<td>sotdadditionalsize</td>
<td>Reflects the segment and block directory table sizes for append-optimized (AO) tables.</td>
</tr>
<tr>
<td>sotdschemaname</td>
<td>The schema name</td>
</tr>
<tr>
<td>sotdtablename</td>
<td>The table name</td>
</tr>
</tbody>
</table>

**gp_size_of_table_uncompressed**

This view shows the uncompressed table size for append-optimized (AO) tables. Otherwise, the table size on disk is shown. The use of this view requires superuser permissions.

**Table 219: gp_size_of_table_uncompressed view**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sotuoid</td>
<td>The object ID of the table</td>
</tr>
<tr>
<td>sotusize</td>
<td>The uncomressed size of the table in bytes if it is a compressed AO table. Otherwise, the table size on disk.</td>
</tr>
<tr>
<td>sotuschemaname</td>
<td>The schema name</td>
</tr>
<tr>
<td>sotutablename</td>
<td>The table name</td>
</tr>
</tbody>
</table>

**gp_disk_free**

This external table runs the *df* (disk free) command on the active segment hosts and reports back the results. Inactive mirrors are not included in the calculation. The use of this external table requires superuser permissions.

**Table 220: gp_disk_free external table**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dfsegment</td>
<td>The content id of the segment (only active segments are shown)</td>
</tr>
<tr>
<td>dhostname</td>
<td>The hostname of the segment host</td>
</tr>
<tr>
<td>dfdevice</td>
<td>The device name</td>
</tr>
<tr>
<td>dfspace</td>
<td>Free disk space in the segment file system in kilobytes</td>
</tr>
</tbody>
</table>
Checking for Uneven Data Distribution

All tables in Greenplum Database are distributed, meaning their data is divided across all of the segments in the system. If the data is not distributed evenly, then query processing performance may suffer. The following views can help diagnose if a table has uneven data distribution:

- `gp_skew_coefficients`
- `gp_skew_idle_fractions`

**gp_skew_coefficients**

This view shows data distribution skew by calculating the coefficient of variation (CV) for the data stored on each segment. This view is accessible to all users, however non-superusers will only be able to see tables that they have permission to access

**Table 221: gp_skew_coefficients view**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>skcoid</td>
<td>The object id of the table.</td>
</tr>
<tr>
<td>skcnamespace</td>
<td>The namespace where the table is defined.</td>
</tr>
<tr>
<td>skcrelname</td>
<td>The table name.</td>
</tr>
<tr>
<td>skccoeff</td>
<td>The coefficient of variation (CV) is calculated as the standard deviation divided by the average. It takes into account both the average and variability around the average of a data series. The lower the value, the better. Higher values indicate greater data skew.</td>
</tr>
</tbody>
</table>

**gp_skew_idle_fractions**

This view shows data distribution skew by calculating the percentage of the system that is idle during a table scan, which is an indicator of processing data skew. This view is accessible to all users, however non-superusers will only be able to see tables that they have permission to access

**Table 222: gp_skew_idle_fractions view**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sifoid</td>
<td>The object id of the table.</td>
</tr>
<tr>
<td>sifnamespace</td>
<td>The namespace where the table is defined.</td>
</tr>
<tr>
<td>sifrelname</td>
<td>The table name.</td>
</tr>
<tr>
<td>siffraction</td>
<td>The percentage of the system that is idle during a table scan, which is an indicator of uneven data distribution or query processing skew. For example, a value of 0.1 indicates 10% skew, a value of 0.5 indicates 50% skew, and so on. Tables that have more than 10% skew should have their distribution policies evaluated.</td>
</tr>
</tbody>
</table>
The **gpperfmon Database**

The `gpperfmon` database is a dedicated database where data collection agents on Greenplum segment hosts save query and system statistics. The optional Greenplum Command Center management tool depends upon the `gpperfmon` database.

The `gpperfmon` database is created using the `gpperfmon_install` command-line utility. The utility creates the database and the `gpmon` database role and enables the data collection agents on the master and segment hosts. See the `gpperfmon_install` reference in the *Greenplum Database Utility Guide* for information about using the utility and configuring the data collection agents.

The `gpperfmon` database consists of three sets of tables that capture query and system status information at different stages.

- **_now** tables store current system metrics such as active queries.
- **_tail** tables are used to stage data before it is saved to the **_history** tables. The **_tail** tables are for internal use only and not to be queried by users.
- **_history** tables store historical metrics.

The data for **_now** and **_tail** tables are stored as text files on the master host file system, and are accessed in the `gpperfmon` database via external tables. The **_history** tables are regular heap database tables in the `gpperfmon` database. History is saved only for queries that run for a minimum number of seconds, 20 by default. You can set this threshold to another value by setting the `min_query_time` parameter in the `$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf` configuration file. Setting the value to 0 saves history for all queries.

The **_history** tables are partitioned by month. See *History Table Partition Retention* for information about removing old partitions.

The database contains the following categories of tables:

- **The `database_*` tables** store query workload information for a Greenplum Database instance.
- **The `diskspace_*` tables** store diskspace metrics.
- **The `filerep_*` tables** store health and status metrics for the file replication process. This process is how high-availability/mirroring is achieved in a Greenplum Database instance. Statistics are maintained for each primary-mirror pair.
- **The `log_alert_*` tables** store error and warning messages from `pg_log`.
- **The `queries_*` tables** store high-level query status information.
- **The `segment_*` tables** store memory allocation statistics for the Greenplum Database segment instances.
- **The `socket_stats_*` tables** store statistical metrics about socket usage for a Greenplum Database instance. Note: These tables are in place for future use and are not currently populated.
- **The `system_*` tables** store system utilization metrics.

The `gpperfmon` database also contains the following views:

- **The `dynamic_memory_info` view** shows an aggregate of all the segments per host and the amount of dynamic memory used per host.
- **The `memory_info` view** shows per-host memory information from the `system_history` and `segment_history` tables.

**History Table Partition Retention**

The **_history** tables** in the `gpperfmon` database are partitioned by month. Partitions are automatically added in two month increments as needed.
The **partition_age** parameter in the `$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf` file can be set to the maximum number of monthly partitions to keep. Partitions older than the specified value are removed automatically when new partitions are added.

The default value for **partition_age** is 0, which means that administrators must manually remove unneeded partitions.

### Alert Log Processing and Log Rotation

When the `gp_gpperfmon_enable` server configuration parameter is set to true, the Greenplum Database syslogger writes alert messages to a `.csv` file in the `$MASTER_DATA_DIRECTORY/gpperfmon/logs` directory.

The level of messages written to the log can be set to `none`, `warning`, `error`, `fatal`, or `panic` by setting the `gpperfmon_log_alert_level` server configuration parameter in `postgresql.conf`. The default message level is `warning`.

The directory where the log is written can be changed by setting the `log_location` configuration variable in the `$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf` configuration file.

The syslogger rotates the alert log every 24 hours or when the current log file reaches or exceeds 1MB.

A rotated log file can exceed 1MB if a single error message contains a large SQL statement or a large stack trace. Also, the syslog processes error messages in chunks, with a separate chunk for each logging process. The size of a chunk is OS-dependent; on Red Hat Enterprise Linux, for example, it is 4096 bytes. If many Greenplum Database sessions generate error messages at the same time, the log file can grow significantly before its size is checked and log rotation is triggered.

### gpperfmon Data Collection Process

When Greenplum Database starts up with gpperfmon support enabled, it forks a `gpmmon` agent process. `gpmmon` then starts a `gpsmon` agent process on the master host and every segment host in the Greenplum Database cluster. The Greenplum Database postmaster process monitors the `gpmmon` process and restarts it if needed, and the `gpmmon` process monitors and restarts `gpsmon` processes as needed.

The `gpmmon` process runs in a loop and at configurable intervals retrieves data accumulated by the `gpsmon` processes, adds it to the data files for the `_now` and `_tail` external database tables, and then into the `_history` regular heap database tables.

**Note:** The `log_alert` tables in the `gpperfmon` database follow a different process, since alert messages are delivered by the Greenplum Database system logger instead of through `gpsmon`. See Alert Log Processing and Log Rotation for more information.

Two configuration parameters in the `$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf` configuration file control how often `gpmmon` activities are triggered:

- The `quantum` parameter is how frequently, in seconds, `gpmmon` requests data from the `gpsmon` agents on the segment hosts and adds retrieved data to the `_now` and `_tail` external table data files. Valid values for the `quantum` parameter are 10, 15, 20, 30, and 60. The default is 15.
- The `harvest_interval` parameter is how frequently, in seconds, data in the `_tail` tables is moved to the `_history` tables. The `harvest_interval` must be at least 30. The default is 120.

See the `gpperfmon_install` management utility reference in the Greenplum Database Utility Guide for the complete list of gpperfmon configuration parameters.

The following steps describe the flow of data from Greenplum Database into the `gpperfmon` database when gpperfmon support is enabled.

1. While executing queries, the Greenplum Database query dispatcher and query executor processes send out query status messages in UDP datagrams. The `gp_gpperfmon_send_interval` server configuration variable determines how frequently the database sends these messages. The default is every second.
2. The \texttt{gpsmon} process on each host receives the UDP packets, consolidates and summarizes the data they contain, and adds additional host metrics, such as CPU and memory usage.

3. The \texttt{gpsmon} processes continue to accumulate data until they receive a dump command from \texttt{gpmmon}.

4. The \texttt{gpsmon} processes respond to a dump command by sending their accumulated status data and log alerts to a listening \texttt{gpmmon} event handler thread.

5. The \texttt{gpmmon} event handler saves the metrics to .txt files in the $\texttt{MASTER\_DATA\_DIRECTORY/ gpperfmon/data}$ directory on the master host.

At each quantum interval (15 seconds by default), \texttt{gpmmon} performs the following steps:

1. Sends a dump command to the \texttt{gpsmon} processes.

2. Gathers and converts the .txt files saved in the $\texttt{MASTER\_DATA\_DIRECTORY/gpperfmon/data}$ directory into .dat external data files for the \texttt{_now} and \texttt{_tail} external tables in the gpperfmon database.

   For example, disk space metrics are added to the \texttt{diskspace\_now.dat} and \texttt{diskspace\_tail.dat} delimited text files. These text files are accessed via the \texttt{diskspace\_now} and \texttt{diskspace\_tail} tables in the gpperfmon database.

At each harvest_interval (120 seconds by default), \texttt{gpmmon} performs the following steps for each \texttt{_tail} file:

1. Renames the \texttt{_tail} file to a \texttt{_stage} file.

2. Creates a new \texttt{_tail} file.

3. Appends data from the \texttt{_stage} file into the \texttt{_tail} file.

4. Runs a SQL command to insert the data from the \texttt{_tail} external table into the corresponding \texttt{_history} table.

   For example, the contents of the \texttt{database\_tail} external table is inserted into the \texttt{database\_history} regular (heap) table.

5. Deletes the \texttt{_tail} file after its contents have been loaded into the database table.

6. Gathers all of the gpdb-alert-*.csv files in the $\texttt{MASTER\_DATA\_DIRECTORY/gpperfmon/logs}$ directory (except the most recent, which the syslogger has open and is writing to) into a single file, \texttt{alert\_log\_stage}.

7. Loads the \texttt{alert\_log\_stage} file into the \texttt{log\_alert\_history} table in the gpperfmon database.

8. Truncates the \texttt{alert\_log\_stage} file.

The following topics describe the contents of the tables in the gpperfmon database.

\texttt{database\_*}

The \texttt{database\_*} tables store query workload information for a Greenplum Database instance. There are three database tables, all having the same columns:

- \texttt{database\_now} is an external table whose data files are stored in $\texttt{MASTER\_DATA\_DIRECTORY/ gpperfmon/data}$. Current query workload data is stored in \texttt{database\_now} during the period between data collection from the data collection agents and automatic commitment to the \texttt{database\_history} table.

- \texttt{database\_tail} is an external table whose data files are stored in $\texttt{MASTER\_DATA\_DIRECTORY/ gpperfmon/data}$. This is a transitional table for query workload data that has been cleared from \texttt{database\_now} but has not yet been committed to \texttt{database\_history}. It typically only contains a few minutes worth of data.

- \texttt{database\_history} is a regular table that stores historical database-wide query workload data. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctime</td>
<td>timestamp</td>
<td>Time this row was created.</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>queries_total</td>
<td>int</td>
<td>The total number of queries in Greenplum Database at data collection time.</td>
</tr>
<tr>
<td>queries_running</td>
<td>int</td>
<td>The number of active queries running at data collection time.</td>
</tr>
<tr>
<td>queries_queue</td>
<td>int</td>
<td>The number of queries waiting in a resource queue at data collection time.</td>
</tr>
</tbody>
</table>

**diskspace_**

The diskspace_ tables store diskspace metrics.

- **diskspace_now** is an external table whose data files are stored in $MASTER_DATA_DIRECTORY/gpperfmon/data. Current diskspace metrics are stored in database_now during the period between data collection from the gpperfmon agents and automatic commitment to the diskspace_history table.
- **diskspace_tail** is an external table whose data files are stored in $MASTER_DATA_DIRECTORY/gpperfmon/data. This is a transitional table for diskspace metrics that have been cleared from diskspace_now but has not yet been committed to diskspace_history. It typically only contains a few minutes worth of data.
- **diskspace_history** is a regular table that stores historical diskspace metrics. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctime</td>
<td>timestamp(0) without time zone</td>
<td>Time of diskspace measurement.</td>
</tr>
<tr>
<td>hostname</td>
<td>varchar(64)</td>
<td>The hostname associated with the diskspace measurement.</td>
</tr>
<tr>
<td>Filesystem</td>
<td>text</td>
<td>Name of the filesystem for the diskspace measurement.</td>
</tr>
<tr>
<td>total_bytes</td>
<td>bigint</td>
<td>Total bytes in the file system.</td>
</tr>
<tr>
<td>bytes_used</td>
<td>bigint</td>
<td>Total bytes used in the file system.</td>
</tr>
<tr>
<td>bytes_available</td>
<td>bigint</td>
<td>Total bytes available in file system.</td>
</tr>
</tbody>
</table>

**filerep_**

The filerep* tables store high-availability file replication process information for a Greenplum Database instance. There are three filerep tables, all having the same columns:

- **filerep_now** is an external table whose data files are stored in $MASTER_DATA_DIRECTORY/gpperfmon/data. Current file replication data is stored in filerep_now during the period between data collection from the data collection agents and automatic commitment to the filerep_history table.
- **filerep_tail** is an external table whose data files are stored in $MASTER_DATA_DIRECTORY/gpperfmon/data. This is a transitional table for file replication data that has been cleared from filerep_now but has not yet been committed to filerep_history. It typically only contains a few minutes worth of data.
filerep_history is a regular table that stores historical database-wide file replication data. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctime</td>
<td>timestamp</td>
<td>Time this row was created.</td>
</tr>
<tr>
<td>primary_measurement_microsec</td>
<td>bigint</td>
<td>The length of time over which primary metrics (contained in UDP messages) were gathered.</td>
</tr>
<tr>
<td>mirror_measurement_microsec</td>
<td>bigint</td>
<td>The length of time over which mirror metrics (contained in UDP messages) were gathered.</td>
</tr>
<tr>
<td>primary_hostname</td>
<td>varchar(64)</td>
<td>The name of the primary host.</td>
</tr>
<tr>
<td>primary_port</td>
<td>int</td>
<td>The port number of the primary host.</td>
</tr>
<tr>
<td>mirror_hostname</td>
<td>varchar(64)</td>
<td>The name of the mirror host.</td>
</tr>
<tr>
<td>mirror_port</td>
<td>int</td>
<td>The port number of the mirror host.</td>
</tr>
<tr>
<td>primary_write_syscall_bytes_avg</td>
<td>bigint</td>
<td>The average amount of data written to disk on the primary for write system calls per interval.</td>
</tr>
<tr>
<td>primary_write_syscall_byte_max</td>
<td>bigint</td>
<td>The maximum amount of data written to disk on the primary for write system calls per interval.</td>
</tr>
<tr>
<td>primary_write_syscall_microsecs_avg</td>
<td>bigint</td>
<td>The average time required for a write system call to write data to disk on the primary per interval.</td>
</tr>
<tr>
<td>primary_write_syscall_microsecs_max</td>
<td>bigint</td>
<td>The maximum time required for a write system call to write data to disk on the primary per interval.</td>
</tr>
<tr>
<td>primary_write_syscall_per_sec</td>
<td>double precision</td>
<td>The number of write system calls on the primary per second. It reflects only the time to queue the write to disk in memory.</td>
</tr>
<tr>
<td>primary_fsync_syscall_microsec_avg</td>
<td>bigint</td>
<td>The average amount of time required for a file sync system call to write data to disk on the primary per interval.</td>
</tr>
<tr>
<td>primary_fsync_syscall_microsec_max</td>
<td>bigint</td>
<td>The maximum amount of time required for a file sync system call to write data to disk on the primary per interval.</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>primary_fsync_syscall_per_sec</code></td>
<td>double precision</td>
<td>The number of file sync system calls on the primary per second. Unlike write system calls which return immediately after the data is posted/queued, file sync system calls wait for all outstanding writes to be written to disk. File sync system call values in this column reflect actual disk access times for potentially large amounts of data.</td>
</tr>
<tr>
<td><code>primary_write_shmem_bytes_avg</code></td>
<td>bigint</td>
<td>The average amount of data written to shared memory on the primary per interval.</td>
</tr>
<tr>
<td><code>primary_write_shmem_bytes_max</code></td>
<td>bigint</td>
<td>The maximum amount of data written to shared memory on the primary per interval.</td>
</tr>
<tr>
<td><code>primary_write_shmem_microsec_avg</code></td>
<td>bigint</td>
<td>The average amount of time required to write data to shared memory on the primary per interval.</td>
</tr>
<tr>
<td><code>primary_write_shmem_microsec_max</code></td>
<td>bigint</td>
<td>The maximum amount of time required to write data to shared memory on the primary per interval.</td>
</tr>
<tr>
<td><code>primary_write_shmem_per_sec</code></td>
<td>double precision</td>
<td>The number of writes to shared memory on the primary per second.</td>
</tr>
<tr>
<td><code>primary_fsync_shmem_microsec_avg</code></td>
<td>bigint</td>
<td>The average amount of time required by the file sync system call to write data to shared memory on the primary per interval.</td>
</tr>
<tr>
<td><code>primary_fsync_shmem_microsec_max</code></td>
<td>bigint</td>
<td>The maximum amount of time required by the file sync system call to write data to shared memory on the primary per interval.</td>
</tr>
<tr>
<td><code>primary_fsync_shmem_per_sec</code></td>
<td>double precision</td>
<td>The number of file sync calls to shared memory on the primary per second. File sync system call values in this column reflect actual disk access times for potentially large amounts of data.</td>
</tr>
<tr>
<td><code>primary_write_shmem_per_sec</code></td>
<td>double precision</td>
<td>The number of writes to shared memory on the primary per second.</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>primary_fsync_shmem_microsec_avg</code></td>
<td>bigint</td>
<td>The average amount of time required by the file sync system call to write data to shared memory on the primary per interval.</td>
</tr>
<tr>
<td><code>primary_fsync_shmem_microsec_max</code></td>
<td>bigint</td>
<td>The maximum amount of time required by the file sync system call to write data to shared memory on the primary per interval.</td>
</tr>
<tr>
<td><code>primary_fsync_shmem_per_sec</code></td>
<td>double precision</td>
<td>The number of file sync calls to shared memory on the primary per second. File sync system call values in this column reflect actual disk access times for potentially large amounts of data.</td>
</tr>
<tr>
<td><code>primary_roundtrip_fsync_msg_microsec_avg</code></td>
<td>bigint</td>
<td>The average amount of time required for a roundtrip file sync between the primary and the mirror per interval. This includes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. The queuing of a file sync message from the primary to the mirror.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. The message traversing the network.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. The execution of the file sync by the mirror.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. The file sync acknowledgement traversing the network back to the primary.</td>
</tr>
<tr>
<td><code>primary_roundtrip_fsync_msg_microsec_max</code></td>
<td>bigint</td>
<td>The maximum amount of time required for a roundtrip file sync between the primary and the mirror per interval. This includes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. The queuing of a file sync message from the primary to the mirror.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. The message traversing the network.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. The execution of the file sync by the mirror.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. The file sync acknowledgement traversing the network back to the primary.</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>primary_roundtrip_fsync_msg_per_sec</td>
<td>double precision</td>
<td>The number of roundtrip file syncs per second.</td>
</tr>
<tr>
<td>primary_roundtrip_test_msg_microsec_avg</td>
<td>bigint</td>
<td>The average amount of time required for a roundtrip test message between the primary and the mirror to complete per interval. This is similar to primary_roundtrip_fsync_msg_microsec_avg, except it does not include a disk access component. Because of this, this is a useful metric that shows the average amount of network delay in the file replication process.</td>
</tr>
<tr>
<td>primary_roundtrip_test_msg_microsec_max</td>
<td>bigint</td>
<td>The maximum amount of time required for a roundtrip test message between the primary and the mirror to complete per interval. This is similar to primary_roundtrip_fsync_msg_microsec_max, except it does not include a disk access component. Because of this, this is a useful metric that shows the maximum amount of network delay in the file replication process.</td>
</tr>
<tr>
<td>primary_roundtrip_test_msg_per_sec</td>
<td>double precision</td>
<td>The number of roundtrip file syncs per second. This is similar to primary_roundtrip_fsync_msg_per_sec, except it does not include a disk access component. As such, this is a useful metric that shows the amount of network delay in the file replication process. Note that test messages typically occur once per minute, so it is common to see a value of “0” for time periods not containing a test message.</td>
</tr>
<tr>
<td>mirror_write_syscall_size_avg</td>
<td>bigint</td>
<td>The average amount of data written to disk on the mirror for write system calls per interval.</td>
</tr>
<tr>
<td>mirror_write_syscall_size_max</td>
<td>bigint</td>
<td>The maximum amount of data written to disk on the mirror for write system calls per interval.</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mirror_write_syscall_microsec_avg</td>
<td>bigint</td>
<td>The average time required for a write system call to write data to disk on the mirror per interval.</td>
</tr>
<tr>
<td>mirror_write_syscall_microsec_max</td>
<td>bigint</td>
<td>The maximum time required for a write system call to write data to disk on the mirror per interval.</td>
</tr>
<tr>
<td>primary_roundtrip_test_msg_per_sec</td>
<td>double precision</td>
<td>The number of roundtrip file syncs per second. This is similar to primary_roundtrip_fsync_msg_per_sec, except it does not include a disk access component. As such, this is a useful metric that shows the amount of network delay in the file replication process. Note that test messages typically occur once per minute, so it is common to see a value of “0” for time periods not containing a test message.</td>
</tr>
<tr>
<td>mirror_write_syscall_size_avg</td>
<td>bigint</td>
<td>The average amount of data written to disk on the mirror for write system calls per interval.</td>
</tr>
<tr>
<td>mirror_write_syscall_size_max</td>
<td>bigint</td>
<td>The maximum amount of data written to disk on the mirror for write system calls per interval.</td>
</tr>
<tr>
<td>mirror_write_syscall_microsec_avg</td>
<td>bigint</td>
<td>The average time required for a write system call to write data to disk on the mirror per interval.</td>
</tr>
</tbody>
</table>

**interface_stats_***

The `interface_stats_*` tables store statistical metrics about communications over each active interface for a Greenplum Database instance.

These tables are in place for future use and are not currently populated.

There are three `interface_stats` tables, all having the same columns:

- `interface_stats_now` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`.
- `interface_stats_tail` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. This is a transitional table for statistical interface metrics that has been cleared from `interface_stats_now` but has not yet been committed to `interface_stats_history`. It typically only contains a few minutes worth of data.
- `interface_stats_history` is a regular table that stores statistical interface metrics. It is pre-partitioned into monthly partitions. Partitions are automatically added in one month increments as needed.
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interface_name</td>
<td>string</td>
<td>Name of the interface. For example: eth0, eth1, lo.</td>
</tr>
<tr>
<td>bytes_received</td>
<td>bigint</td>
<td>Amount of data received in bytes.</td>
</tr>
<tr>
<td>packets_received</td>
<td>bigint</td>
<td>Number of packets received.</td>
</tr>
<tr>
<td>receive_errors</td>
<td>bigint</td>
<td>Number of errors encountered while data was being received.</td>
</tr>
<tr>
<td>receive Drops</td>
<td>bigint</td>
<td>Number of times packets were dropped while data was being received.</td>
</tr>
<tr>
<td>receive fifo errors</td>
<td>bigint</td>
<td>Number of times FIFO (first in first out) errors were encountered while data was being received.</td>
</tr>
<tr>
<td>receive frame errors</td>
<td>bigint</td>
<td>Number of frame errors while data was being received.</td>
</tr>
<tr>
<td>receive compressed packets</td>
<td>int</td>
<td>Number of packets received in compressed format.</td>
</tr>
<tr>
<td>receive multicast packets</td>
<td>int</td>
<td>Number of multicast packets received.</td>
</tr>
<tr>
<td>bytes transmitted</td>
<td>bigint</td>
<td>Amount of data transmitted in bytes.</td>
</tr>
<tr>
<td>packets transmitted</td>
<td>bigint</td>
<td>Amount of data transmitted in bytes.</td>
</tr>
<tr>
<td>packets transmitted</td>
<td>bigint</td>
<td>Number of packets transmitted.</td>
</tr>
<tr>
<td>transmit errors</td>
<td>bigint</td>
<td>Number of errors encountered during data transmission.</td>
</tr>
<tr>
<td>transmit drops</td>
<td>bigint</td>
<td>Number of times packets were dropped during data transmission.</td>
</tr>
<tr>
<td>transmit fifo errors</td>
<td>bigint</td>
<td>Number of times fifo errors were encountered during data transmission.</td>
</tr>
<tr>
<td>transmit collision errors</td>
<td>bigint</td>
<td>Number of times collision errors were encountered during data transmission.</td>
</tr>
<tr>
<td>transmit carrier errors</td>
<td>bigint</td>
<td>Number of times carrier errors were encountered during data transmission.</td>
</tr>
<tr>
<td>transmit compressed packets</td>
<td>int</td>
<td>Number of packets transmitted in compressed format.</td>
</tr>
</tbody>
</table>

log_alert *

The log_alert_* tables store pg_log errors and warnings.
See Alert Log Processing and Log Rotation for information about configuring the system logger for gpperfmon.

There are three log_alert tables, all having the same columns:

- **log_alert_now** is an external table whose data is stored in .csv files in the $MASTER_DATA_DIRECTORY/gpperfmon/logs directory. Current pg_log errors and warnings data are available in log_alert_now during the period between data collection from the gpperfmon agents and automatic commitment to the log_alert_history table.

- **log_alert_tail** is an external table with data stored in $MASTER_DATA_DIRECTORY/gpperfmon/logs/alert_log_stage. This is a transitional table for data that has been cleared from log_alert_now but has not yet been committed to log_alert_history. The table includes records from all alert logs except the most recent. It typically contains only a few minutes' worth of data.

- **log_alert_history** is a regular table that stores historical database-wide errors and warnings data. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logtime</td>
<td>timestamp with time zone</td>
<td>Timestamp for this log</td>
</tr>
<tr>
<td>loguser</td>
<td>text</td>
<td>User of the query</td>
</tr>
<tr>
<td>logdatabase</td>
<td>text</td>
<td>The accessed database</td>
</tr>
<tr>
<td>logpid</td>
<td>text</td>
<td>Process id</td>
</tr>
<tr>
<td>logthread</td>
<td>text</td>
<td>Thread number</td>
</tr>
<tr>
<td>loghost</td>
<td>text</td>
<td>Host name or ip address</td>
</tr>
<tr>
<td>logport</td>
<td>text</td>
<td>Port number</td>
</tr>
<tr>
<td>logsessiontime</td>
<td>timestamp with time zone</td>
<td>Session timestamp</td>
</tr>
<tr>
<td>logtransaction</td>
<td>integer</td>
<td>Transaction id</td>
</tr>
<tr>
<td>logsession</td>
<td>text</td>
<td>Session id</td>
</tr>
<tr>
<td>logcmdcount</td>
<td>text</td>
<td>Command count</td>
</tr>
<tr>
<td>logsegment</td>
<td>text</td>
<td>Segment number</td>
</tr>
<tr>
<td>logslice</td>
<td>text</td>
<td>Slice number</td>
</tr>
<tr>
<td>logdistxact</td>
<td>text</td>
<td>Distributed transaction</td>
</tr>
<tr>
<td>loglocalxact</td>
<td>text</td>
<td>Local transaction</td>
</tr>
<tr>
<td>logsubxact</td>
<td>text</td>
<td>Subtransaction</td>
</tr>
<tr>
<td>logseverity</td>
<td>text</td>
<td>Log severity</td>
</tr>
<tr>
<td>logstate</td>
<td>text</td>
<td>State</td>
</tr>
<tr>
<td>logmessage</td>
<td>text</td>
<td>Log message</td>
</tr>
<tr>
<td>logdetail</td>
<td>text</td>
<td>Detailed message</td>
</tr>
<tr>
<td>loghint</td>
<td>text</td>
<td>Hint info</td>
</tr>
<tr>
<td>logquery</td>
<td>text</td>
<td>Executed query</td>
</tr>
<tr>
<td>logquerypos</td>
<td>text</td>
<td>Query position</td>
</tr>
<tr>
<td>logcontext</td>
<td>text</td>
<td>Context info</td>
</tr>
</tbody>
</table>
### Column Types and Descriptions

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logdebug</td>
<td>text</td>
<td>Debug</td>
</tr>
<tr>
<td>logcursorpos</td>
<td>text</td>
<td>Cursor position</td>
</tr>
<tr>
<td>logfunction</td>
<td>text</td>
<td>Function info</td>
</tr>
<tr>
<td>logfile</td>
<td>text</td>
<td>Source code file</td>
</tr>
<tr>
<td>logline</td>
<td>text</td>
<td>Source code line</td>
</tr>
<tr>
<td>logstack</td>
<td>text</td>
<td>Stack trace</td>
</tr>
</tbody>
</table>

### queries_*

The `queries_*` tables store high-level query status information.

The `tmid`, `ssid` and `ccnt` columns are the composite key that uniquely identifies a particular query.

There are three queries tables, all having the same columns:

- **queries_now** is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. Current query status is stored in `queries_now` during the period between data collection from the `gpperfmon` agents and automatic commitment to the `queries_history` table.
- **queries_tail** is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. This is a transitional table for query status data that has been cleared from `queries_now` but has not yet been committed to `queries_history`. It typically only contains a few minutes worth of data.
- **queries_history** is a regular table that stores historical query status data. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed.

### Column Types and Descriptions

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctime</td>
<td>timestamp</td>
<td>Time this row was created.</td>
</tr>
<tr>
<td>tmid</td>
<td>int</td>
<td>A time identifier for a particular query. All records associated with the query will have the same <code>tmid</code>.</td>
</tr>
<tr>
<td>ssid</td>
<td>int</td>
<td>The session id as shown by <code>gp_session_id</code>. All records associated with the query will have the same <code>ssid</code>.</td>
</tr>
<tr>
<td>ccnt</td>
<td>int</td>
<td>The command number within this session as shown by <code>gp_command_count</code>. All records associated with the query will have the same <code>ccnt</code>.</td>
</tr>
<tr>
<td>username</td>
<td>varchar(64)</td>
<td>Greenplum role name that issued this query.</td>
</tr>
<tr>
<td>db</td>
<td>varchar(64)</td>
<td>Name of the database queried.</td>
</tr>
<tr>
<td>cost</td>
<td>int</td>
<td>Not implemented in this release.</td>
</tr>
<tr>
<td>tsubmit</td>
<td>timestamp</td>
<td>Time the query was submitted.</td>
</tr>
<tr>
<td>tstart</td>
<td>timestamp</td>
<td>Time the query was started.</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tfinish</td>
<td>timestamp</td>
<td>Time the query finished.</td>
</tr>
<tr>
<td>status</td>
<td>varchar(64)</td>
<td>Status of the query -- start, done, or abort.</td>
</tr>
<tr>
<td>rows_out</td>
<td>bigint</td>
<td>Rows out for the query.</td>
</tr>
<tr>
<td>cpu_elapsed</td>
<td>bigint</td>
<td>CPU usage by all processes across all segments executing this query (in seconds). It is the sum of the CPU usage values taken from all active primary segments in the database system. Note that the value is logged as 0 if the query runtime is shorter than the value for the quantum. This occurs even if the query runtime is greater than the value for min_query_time, and this value is lower than the value for the quantum.</td>
</tr>
<tr>
<td>cpu_currpct</td>
<td>float</td>
<td>Current CPU percent average for all processes executing this query. The percentages for all processes running on each segment are averaged, and then the average of all those values is calculated to render this metric. Current CPU percent average is always zero in historical and tail data.</td>
</tr>
<tr>
<td>skew_cpu</td>
<td>float</td>
<td>Displays the amount of processing skew in the system for this query. Processing/CPU skew occurs when one segment performs a disproportionate amount of processing for a query. This value is the coefficient of variation in the CPU% metric across all segments for this query, multiplied by 100. For example, a value of .95 is shown as 95.</td>
</tr>
</tbody>
</table>
### Column Types and Descriptions

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>skew_rows</td>
<td>float</td>
<td>Displays the amount of row skew in the system. Row skew occurs when one segment produces a disproportionate number of rows for a query. This value is the coefficient of variation for the rows_in metric across all segments for this query, multiplied by 100. For example, a value of .95 is shown as 95.</td>
</tr>
<tr>
<td>query_hash</td>
<td>bigint</td>
<td>Not implemented in this release.</td>
</tr>
<tr>
<td>query_text</td>
<td>text</td>
<td>The SQL text of this query.</td>
</tr>
<tr>
<td>query_plan</td>
<td>text</td>
<td>Text of the query plan. Not implemented in this release.</td>
</tr>
<tr>
<td>application_name</td>
<td>varchar(64)</td>
<td>The name of the application.</td>
</tr>
<tr>
<td>rsqname</td>
<td>varchar(64)</td>
<td>The name of the resource queue.</td>
</tr>
<tr>
<td>rqppriority</td>
<td>varchar(64)</td>
<td>The priority of the query -- max, high, med, low, or min.</td>
</tr>
</tbody>
</table>

### segment_ *

The `segment_*` tables contain memory allocation statistics for the Greenplum Database segment instances. This tracks the amount of memory consumed by all postgres processes of a particular segment instance, and the remaining amount of memory available to a segment as per the setting of the `postgresql.conf` configuration parameter: `gp_vmem_protect_limit`. Query processes that cause a segment to exceed this limit will be cancelled in order to prevent system-level out-of-memory errors. See the Greenplum Database Reference Guide for more information about this parameter.

There are three segment tables, all having the same columns:

- **segment_now** is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. Current memory allocation data is stored in `segment_now` during the period between data collection from the `gpperfmon` agents and automatic commitment to the `segment_history` table.
- **segment_tail** is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. This is a transitional table for memory allocation data that has been cleared from `segment_now` but has not yet been committed to `segment_history`. It typically only contains a few minutes worth of data.
- **segment_history** is a regular table that stores historical memory allocation metrics. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed.

A particular segment instance is identified by its `hostname` and `dbid` (the unique segment identifier as per the `gp_segment_configuration` system catalog table).

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctime</td>
<td>timestamp(0)</td>
<td>The time the row was created.</td>
</tr>
</tbody>
</table>
### Column

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbid</td>
<td>int</td>
<td>The segment ID (dbid from gp_segment_configuration).</td>
</tr>
<tr>
<td>hostname</td>
<td>charvar(64)</td>
<td>The segment hostname.</td>
</tr>
<tr>
<td>dynamic_memory_used</td>
<td>bigint</td>
<td>The amount of dynamic memory (in bytes) allocated to query processes running on this segment.</td>
</tr>
<tr>
<td>dynamic_memory_available</td>
<td>bigint</td>
<td>The amount of additional dynamic memory (in bytes) that the segment can request before reaching the limit set by the gp_vmem_protect_limit parameter.</td>
</tr>
</tbody>
</table>

See also the views memory_info and dynamic_memory_info for aggregated memory allocation and utilization by host.

### socket_stats_*

The socket_stats_* tables store statistical metrics about socket usage for a Greenplum Database instance. There are three system tables, all having the same columns:

These tables are in place for future use and are not currently populated.

- socket_stats_now is an external table whose data files are stored in $MASTER_DATA_DIRECTORY/gpperfmon/data.
- socket_stats_tail is an external table whose data files are stored in $MASTER_DATA_DIRECTORY/gpperfmon/data. This is a transitional table for socket statistical metrics that has been cleared from socket_stats_now but has not yet been committed to socket_stats_history. It typically only contains a few minutes worth of data.
- socket_stats_history is a regular table that stores historical socket statistical metrics. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>total_sockets_used</td>
<td>int</td>
<td>Total sockets used in the system.</td>
</tr>
<tr>
<td>tcp_sockets_inuse</td>
<td>int</td>
<td>Number of TCP sockets in use.</td>
</tr>
<tr>
<td>tcp_sockets_orphan</td>
<td>int</td>
<td>Number of TCP sockets orphaned.</td>
</tr>
<tr>
<td>tcp_sockets_timewait</td>
<td>int</td>
<td>Number of TCP sockets in Time-Wait.</td>
</tr>
<tr>
<td>tcp_sockets_alloc</td>
<td>int</td>
<td>Number of TCP sockets allocated.</td>
</tr>
<tr>
<td>tcp_sockets_memusage_inbytes</td>
<td>int</td>
<td>Amount of memory consumed by TCP sockets.</td>
</tr>
<tr>
<td>udp_sockets_inuse</td>
<td>int</td>
<td>Number of UDP sockets in use.</td>
</tr>
<tr>
<td>udp_sockets_memusage_inbytes</td>
<td>int</td>
<td>Amount of memory consumed by UDP sockets.</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>----------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>rawsockets_inuse</td>
<td>int</td>
<td>Number of RAW sockets in use.</td>
</tr>
<tr>
<td>fragsockets_inuse</td>
<td>int</td>
<td>Number of FRAG sockets in use.</td>
</tr>
<tr>
<td>fragsockets_memusage_inbytes</td>
<td>int</td>
<td>Amount of memory consumed by FRAG sockets.</td>
</tr>
</tbody>
</table>

**system_***

The system_* tables store system utilization metrics. There are three system tables, all having the same columns:

- **system_now** is an external table whose data files are stored in $MASTER_DATA_DIRECTORY/gpperfmon/data. Current system utilization data is stored in system_now during the period between data collection from the gpperfmon agents and automatic commitment to the system_history table.
- **system_tail** is an external table whose data files are stored in $MASTER_DATA_DIRECTORY/gpperfmon/data. This is a transitional table for system utilization data that has been cleared from system_now but has not yet been committed to system_history. It typically only contains a few minutes worth of data.
- **system_history** is a regular table that stores historical system utilization metrics. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctime</td>
<td>timestamp</td>
<td>Time this row was created.</td>
</tr>
<tr>
<td>hostname</td>
<td>varchar(64)</td>
<td>Segment or master hostname associated with these system metrics.</td>
</tr>
<tr>
<td>mem_total</td>
<td>bigint</td>
<td>Total system memory in Bytes for this host.</td>
</tr>
<tr>
<td>mem_used</td>
<td>bigint</td>
<td>Used system memory in Bytes for this host.</td>
</tr>
<tr>
<td>mem_actual_used</td>
<td>bigint</td>
<td>Used actual memory in Bytes for this host (not including the memory reserved for cache and buffers).</td>
</tr>
<tr>
<td>mem_actual_free</td>
<td>bigint</td>
<td>Free actual memory in Bytes for this host (not including the memory reserved for cache and buffers).</td>
</tr>
<tr>
<td>swap_total</td>
<td>bigint</td>
<td>Total swap space in Bytes for this host.</td>
</tr>
<tr>
<td>swap_used</td>
<td>bigint</td>
<td>Used swap space in Bytes for this host.</td>
</tr>
<tr>
<td>swap_page_in</td>
<td>bigint</td>
<td>Number of swap pages in.</td>
</tr>
<tr>
<td>swap_page_out</td>
<td>bigint</td>
<td>Number of swap pages out.</td>
</tr>
<tr>
<td>cpu_user</td>
<td>float</td>
<td>CPU usage by the Greenplum system user.</td>
</tr>
<tr>
<td>cpu_sys</td>
<td>float</td>
<td>CPU usage for this host.</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cpu_idle</td>
<td>float</td>
<td>Idle CPU capacity at metric collection time.</td>
</tr>
<tr>
<td>load0</td>
<td>float</td>
<td>CPU load average for the prior one-minute period.</td>
</tr>
<tr>
<td>load1</td>
<td>float</td>
<td>CPU load average for the prior five-minute period.</td>
</tr>
<tr>
<td>load2</td>
<td>float</td>
<td>CPU load average for the prior fifteen-minute period.</td>
</tr>
<tr>
<td>quantum</td>
<td>int</td>
<td>Interval between metric collection for this metric entry.</td>
</tr>
<tr>
<td>disk_ro_rate</td>
<td>bigint</td>
<td>Disk read operations per second.</td>
</tr>
<tr>
<td>disk_wo_rate</td>
<td>bigint</td>
<td>Disk write operations per second.</td>
</tr>
<tr>
<td>disk_rb_rate</td>
<td>bigint</td>
<td>Bytes per second for disk write operations.</td>
</tr>
<tr>
<td>net_rp_rate</td>
<td>bigint</td>
<td>Packets per second on the system network for read operations.</td>
</tr>
<tr>
<td>net_wp_rate</td>
<td>bigint</td>
<td>Packets per second on the system network for write operations.</td>
</tr>
<tr>
<td>net_rb_rate</td>
<td>bigint</td>
<td>Bytes per second on the system network for read operations.</td>
</tr>
<tr>
<td>net_wb_rate</td>
<td>bigint</td>
<td>Bytes per second on the system network for write operations.</td>
</tr>
</tbody>
</table>

**dynamic_memory_info**

The `dynamic_memory_info` view shows a sum of the used and available dynamic memory for all segment instances on a segment host. Dynamic memory refers to the maximum amount of memory that Greenplum Database instance will allow the query processes of a single segment instance to consume before it starts cancelling processes. This limit is set by the `gp_vmem_protect_limit` server configuration parameter, and is evaluated on a per-segment basis.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctime</td>
<td>timestamp(0)</td>
<td>Time this row was created in the segment_history table.</td>
</tr>
<tr>
<td>hostname</td>
<td>varchar(64)</td>
<td>Segment or master hostname associated with these system memory metrics.</td>
</tr>
<tr>
<td>dynamic_memory_used_mb</td>
<td>numeric</td>
<td>The amount of dynamic memory in MB allocated to query processes running on this segment.</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dynamic_memory_available_mb</td>
<td>numeric</td>
<td>The amount of additional dynamic memory (in MB) available to the query processes running on this segment host. Note that this value is a sum of the available memory for all segments on a host. Even though this value reports available memory, it is possible that one or more segments on the host have exceeded their memory limit as set by the <code>gp_vmem_protect_limit</code> parameter.</td>
</tr>
</tbody>
</table>

**memory_info**

The `memory_info` view shows per-host memory information from the system_history and segment_history tables. This allows administrators to compare the total memory available on a segment host, total memory used on a segment host, and dynamic memory used by query processes.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctime</td>
<td>timestamp(0) without time zone</td>
<td>Time this row was created in the segment_history table.</td>
</tr>
<tr>
<td>hostname</td>
<td>varchar(64)</td>
<td>Segment or master hostname associated with these system memory metrics.</td>
</tr>
<tr>
<td>mem_total_mb</td>
<td>numeric</td>
<td>Total system memory in MB for this segment host.</td>
</tr>
<tr>
<td>mem_used_mb</td>
<td>numeric</td>
<td>Total system memory used in MB for this segment host.</td>
</tr>
<tr>
<td>mem_actual_used_mb</td>
<td>numeric</td>
<td>Actual system memory used in MB for this segment host.</td>
</tr>
<tr>
<td>mem_actual_free_mb</td>
<td>numeric</td>
<td>Actual system memory free in MB for this segment host.</td>
</tr>
<tr>
<td>swap_total_mb</td>
<td>numeric</td>
<td>Total swap space in MB for this segment host.</td>
</tr>
<tr>
<td>swap_used_mb</td>
<td>numeric</td>
<td>Total swap space used in MB for this segment host.</td>
</tr>
<tr>
<td>dynamic_memory_used_mb</td>
<td>numeric</td>
<td>The amount of dynamic memory in MB allocated to query processes running on this segment.</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dynamic_memory_available_mb</td>
<td>numeric</td>
<td>The amount of additional dynamic memory (in MB) available to the query processes running on this segment host. Note that this value is a sum of the available memory for all segments on a host. Even though this value reports available memory, it is possible that one or more segments on the host have exceeded their memory limit as set by the gp_vmem_protect_limit parameter.</td>
</tr>
</tbody>
</table>
Greenplum Database Data Types

Greenplum Database has a rich set of native data types available to users. Users may also define new data types using the `CREATE TYPE` command. This reference shows all of the built-in data types. In addition to the types listed here, there are also some internally used data types, such as `oid` (object identifier), but those are not documented in this guide.

Optional modules in the `contrib` directory may also install new data types. The `hstore` module, for example, introduces a new data type and associated functions for working with key-value pairs. See `hstore Functions`.

The following data types are specified by SQL: `bit`, `bit varying`, `boolean`, `character varying`, `varchar`, `character`, `char`, `date`, `double precision`, `integer`, `interval`, `numeric`, `decimal`, `real`, `smallint`, `time` (with or without time zone), and `timestamp` (with or without time zone).

Each data type has an external representation determined by its input and output functions. Many of the built-in types have obvious external formats. However, several types are either unique to PostgreSQL (and Greenplum Database), such as geometric paths, or have several possibilities for formats, such as the date and time types. Some of the input and output functions are not invertible. That is, the result of an output function may lose accuracy when compared to the original input.

### Table 223: Greenplum Database Built-in Data Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Alias</th>
<th>Size</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bigint</td>
<td>int8</td>
<td>8 bytes</td>
<td>-922337203 to 922337203 6854775808 to 922337203 6854775807</td>
<td>large range integer</td>
</tr>
<tr>
<td>bigserial</td>
<td>serial8</td>
<td>8 bytes</td>
<td>1 to 922337203 6854775807</td>
<td>large autoincrementing integer</td>
</tr>
<tr>
<td>bit [ (n) ]</td>
<td></td>
<td>n bits</td>
<td></td>
<td>fixed-length bit string</td>
</tr>
<tr>
<td>bit varying [ (n) ]Footnote.</td>
<td>varbit</td>
<td>actual number of bits</td>
<td>bit string constant</td>
<td>variable-length bit string</td>
</tr>
<tr>
<td>boolean</td>
<td>bool</td>
<td>1 byte</td>
<td>true/false, t/f, yes/no, y/n, 1/0</td>
<td>logical boolean (true/false)</td>
</tr>
<tr>
<td>box</td>
<td></td>
<td>32 bytes</td>
<td>((x1,y1),(x2,y2))</td>
<td>rectangular box in the plane - not allowed in distribution key columns.</td>
</tr>
<tr>
<td>byteaFootnote.</td>
<td></td>
<td>1 byte + binary string</td>
<td>sequence of octets</td>
<td>variable-length binary string</td>
</tr>
<tr>
<td>character [ (n) ]Footnote.</td>
<td>char [ (n) ]</td>
<td>1 byte + n</td>
<td>strings up to n characters in length</td>
<td>fixed-length, blank padded</td>
</tr>
</tbody>
</table>

9 For variable length data types, if the data is greater than or equal to 127 bytes, the storage overhead is 4 bytes instead of 1.
<table>
<thead>
<tr>
<th>Name</th>
<th>Alias</th>
<th>Size</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>character varying [(n)]</td>
<td>varchar [(n)]</td>
<td>1 byte + string size</td>
<td>strings up to n characters in length</td>
<td>variable-length with limit</td>
</tr>
<tr>
<td>cidr</td>
<td></td>
<td>12 or 24 bytes</td>
<td></td>
<td>IPv4 and IPv6 networks</td>
</tr>
<tr>
<td>circle</td>
<td></td>
<td>24 bytes</td>
<td>&lt;(x,y),r&gt; (center and radius)</td>
<td>circle in the plane - not allowed in distribution key columns.</td>
</tr>
<tr>
<td>date</td>
<td></td>
<td>4 bytes</td>
<td>4713 BC - 294,277 AD</td>
<td>calendar date (year, month, day)</td>
</tr>
<tr>
<td>decimal [(p, s)]</td>
<td>numeric [(p, s)]</td>
<td>variable</td>
<td>no limit</td>
<td>user-specified precision, exact</td>
</tr>
<tr>
<td>double precision</td>
<td>float8, float</td>
<td>8 bytes</td>
<td>15 decimal digits precision</td>
<td>variable-precision, inexact</td>
</tr>
<tr>
<td>inet</td>
<td></td>
<td>12 or 24 bytes</td>
<td></td>
<td>IPv4 and IPv6 hosts and networks</td>
</tr>
<tr>
<td>integer</td>
<td>int, int4</td>
<td>4 bytes</td>
<td>-2147483648 to +2147483647</td>
<td>usual choice for integer</td>
</tr>
<tr>
<td>interval [(p)]</td>
<td></td>
<td>12 bytes</td>
<td>-178000000 years - 178000000 years</td>
<td>time span</td>
</tr>
<tr>
<td>json</td>
<td></td>
<td>1 byte + json size</td>
<td>json of any length</td>
<td>variable unlimited length</td>
</tr>
<tr>
<td>lseg</td>
<td></td>
<td>32 bytes</td>
<td>((x1,y1),(x2,y2))</td>
<td>line segment in the plane - not allowed in distribution key columns.</td>
</tr>
<tr>
<td>macaddr</td>
<td></td>
<td>6 bytes</td>
<td></td>
<td>MAC addresses</td>
</tr>
<tr>
<td>money</td>
<td></td>
<td>8 bytes</td>
<td>-9223372036854775.08 to +9223372036854775.07</td>
<td>currency amount</td>
</tr>
<tr>
<td>path</td>
<td></td>
<td>16+16n bytes</td>
<td>[(x1,y1),... ]</td>
<td>geometric path in the plane - not allowed in distribution key columns.</td>
</tr>
<tr>
<td>point</td>
<td></td>
<td>16 bytes</td>
<td>(x,y)</td>
<td>geometric point in the plane - not allowed in distribution key columns.</td>
</tr>
<tr>
<td>Name</td>
<td>Alias</td>
<td>Size</td>
<td>Range</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>------------</td>
<td>------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>polygon</td>
<td></td>
<td>40+16n bytes</td>
<td>((x1,y1),...)</td>
<td>closed geometric path in the plane - not allowed in distribution key columns.</td>
</tr>
<tr>
<td>real</td>
<td>float4</td>
<td>4 bytes</td>
<td>6 decimal digits precision</td>
<td>variable-precision, inexact</td>
</tr>
<tr>
<td>serial</td>
<td>serial4</td>
<td>4 bytes</td>
<td>1 to 2147483647</td>
<td>autoincrementing integer</td>
</tr>
<tr>
<td>smallint</td>
<td>int2</td>
<td>2 bytes</td>
<td>-32768 to +32767</td>
<td>small range integer</td>
</tr>
<tr>
<td>text</td>
<td></td>
<td>1 byte + string size</td>
<td>strings of any length</td>
<td>variable unlimited length</td>
</tr>
<tr>
<td>time</td>
<td></td>
<td>8 bytes</td>
<td>00:00:00[.000000] - 24:00:00[.000000]</td>
<td>time of day only</td>
</tr>
<tr>
<td>timetz</td>
<td></td>
<td>12 bytes</td>
<td>00:00:00+1359 - 24:00:00-1359</td>
<td>time of day only, with time zone</td>
</tr>
<tr>
<td>timestamp</td>
<td></td>
<td>8 bytes</td>
<td>4713 BC - 294,277 AD</td>
<td>both date and time</td>
</tr>
<tr>
<td>timestamptz</td>
<td></td>
<td>8 bytes</td>
<td>4713 BC - 294,277 AD</td>
<td>both date and time, with time zone</td>
</tr>
<tr>
<td>uuid</td>
<td></td>
<td>32 bytes</td>
<td></td>
<td>Universally Unique Identifiers according to RFC 4122, ISO/IEC 9834-8:2005</td>
</tr>
<tr>
<td>xml</td>
<td></td>
<td>1 byte + xml size</td>
<td>xml of any length</td>
<td>variable unlimited length</td>
</tr>
<tr>
<td>txid_snapshot</td>
<td></td>
<td></td>
<td></td>
<td>user-level transaction ID snapshot</td>
</tr>
</tbody>
</table>

**Pseudo-Types**

Greenplum Database supports special-purpose data type entries that are collectively called *pseudo-types*. A pseudo-type cannot be used as a column data type, but it can be used to declare a function's argument or result type. Each of the available pseudo-types is useful in situations where a function's behavior does not correspond to simply taking or returning a value of a specific SQL data type.

Functions coded in procedural languages can use pseudo-types only as allowed by their implementation languages. The procedural languages all forbid use of a pseudo-type as an argument type, and allow only `void` and `record` as a result type.

A function with the pseudo-type `record` as a return data type returns an unspecified row type. The `record` represents an array of possibly-anonymous composite types. Since composite datums carry their own type identification, no extra knowledge is needed at the array level.

The pseudo-type `void` indicates that a function returns no value.

**Note:** Greenplum Database does not support triggers and the pseudo-type `trigger`. 
The types `anyelement`, `anyarray`, `anynonarray`, and `anyenum` are pseudo-types called polymorphic types. Some procedural languages also support polymorphic functions using the types `anyarray`, `anyelement`, `anyenum`, and `anynonarray`.

For more information about pseudo-types, see the Postgres documentation about *Pseudo-Types*.

**Polymorphic Types**

Four pseudo-types types of special interest are `anyelement`, `anyarray`, `anynonarray`, and `anyenum`, which are collectively called polymorphic types. Any function declared using these types is said to be a polymorphic function. A polymorphic function can operate on many different data types, with the specific data types being determined by the data types actually passed to it at runtime.

Polymorphic arguments and results are tied to each other and are resolved to a specific data type when a query calling a polymorphic function is parsed. Each position (either argument or return value) declared as `anyelement` is allowed to have any specific actual data type, but in any given call they must all be the same actual type. Each position declared as `anyarray` can have any array data type, but similarly they must all be the same type. If there are positions declared `anyarray` and others declared `anyelement`, the actual array type in the `anyarray` positions must be an array whose elements are the same type appearing in the `anyelement` positions. `anynonarray` is treated exactly the same as `anyelement`, but adds the additional constraint that the actual type must not be an array type. `anyenum` is treated exactly the same as `anyelement`, but adds the additional constraint that the actual type must be an `enum` type.

When more than one argument position is declared with a polymorphic type, the net effect is that only certain combinations of actual argument types are allowed. For example, a function declared as `equal(anyelement, anyelement)` takes any two input values, so long as they are of the same data type.

When the return value of a function is declared as a polymorphic type, there must be at least one argument position that is also polymorphic, and the actual data type supplied as the argument determines the actual result type for that call. For example, if there were not already an array subcripting mechanism, one could define a function that implements subscripting as `subscript(anyarray, integer) returns anyelement`. This declaration constrains the actual first argument to be an array type, and allows the parser to infer the correct result type from the actual first argument’s type. Another example is that a function declared as `myfunc(anyarray) returns anyelement` will only accept arrays of `enum` types.

Note that `anynonarray` and `anyenum` do not represent separate type variables; they are the same type as `anyelement`, just with an additional constraint. For example, declaring a function as `myfunc(anyelement, anyenum)` is equivalent to declaring it as `myfunc(anyenum, anyenum)`: both actual arguments must be the same `enum` type.

A variadic function (one taking a variable number of arguments) is polymorphic when its last parameter is declared as `VARIADIC anyarray`. For purposes of argument matching and determining the actual result type, such a function behaves the same as if you had declared the appropriate number of `anynonarray` parameters.

For more information about polymorphic types, see the Postgres documentation about *Polymorphic Arguments and Return Types*. 
Character Set Support

The character set support in Greenplum Database allows you to store text in a variety of character sets, including single-byte character sets such as the ISO 8859 series and multiple-byte character sets such as EUC (Extended Unix Code), UTF-8, and Mule internal code. All supported character sets can be used transparently by clients, but a few are not supported for use within the server (that is, as a server-side encoding). The default character set is selected while initializing your Greenplum Database array using `gpinitsystem`. It can be overridden when you create a database, so you can have multiple databases each with a different character set.

Table 224: Greenplum Database Character Sets

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Language</th>
<th>Server?</th>
<th>Bytes/Char</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIG5</td>
<td>Big Five</td>
<td>Traditional Chinese</td>
<td>No</td>
<td>1-2</td>
<td>WIN950, Windows950</td>
</tr>
<tr>
<td>EUC_CN</td>
<td>Extended UNIX Code-CN</td>
<td>Simplified Chinese</td>
<td>Yes</td>
<td>1-3</td>
<td></td>
</tr>
<tr>
<td>EUC_TW</td>
<td>Extended UNIX Code-TW</td>
<td>Traditional Chinese, Taiwanese</td>
<td>Yes</td>
<td>1-3</td>
<td></td>
</tr>
<tr>
<td>GB18030</td>
<td>National Standard</td>
<td>Chinese</td>
<td>No</td>
<td>1-2</td>
<td></td>
</tr>
<tr>
<td>GBK</td>
<td>Extended National Standard</td>
<td>Simplified Chinese</td>
<td>No</td>
<td>1-2</td>
<td>WIN936, Windows936</td>
</tr>
<tr>
<td>ISO_8859_5</td>
<td>ISO 8859-5, ECMA 113</td>
<td>Latin/Cyrillic</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ISO_8859_6</td>
<td>ISO 8859-6, ECMA 114</td>
<td>Latin/Arabic</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ISO_8859_7</td>
<td>ISO 8859-7, ECMA 118</td>
<td>Latin/Greek</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ISO_8859_8</td>
<td>ISO 8859-8, ECMA 121</td>
<td>Latin/Hebrew</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>JOHAB</td>
<td>JOHA</td>
<td>Korean (Hangul)</td>
<td>Yes</td>
<td>1-3</td>
<td></td>
</tr>
<tr>
<td>KOI8</td>
<td>KOI8-R(U)</td>
<td>Cyrillic</td>
<td>Yes</td>
<td>1</td>
<td>KOI8R</td>
</tr>
<tr>
<td>LATIN1</td>
<td>ISO 8859-1, ECMA 94</td>
<td>Western European</td>
<td>Yes</td>
<td>1</td>
<td>ISO88591</td>
</tr>
</tbody>
</table>

Not all APIs support all the listed character sets. For example, the JDBC driver does not support MULE_INTERNAL, LATIN6, LATIN8, and LATIN10.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Language</th>
<th>Server?</th>
<th>Bytes/Char</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>LATIN2</td>
<td>ISO 8859-2, ECMA 94</td>
<td>Central European</td>
<td>Yes</td>
<td>1</td>
<td>ISO88592</td>
</tr>
<tr>
<td>LATIN3</td>
<td>ISO 8859-3, ECMA 94</td>
<td>South European</td>
<td>Yes</td>
<td>1</td>
<td>ISO88593</td>
</tr>
<tr>
<td>LATIN4</td>
<td>ISO 8859-4, ECMA 94</td>
<td>North European</td>
<td>Yes</td>
<td>1</td>
<td>ISO88594</td>
</tr>
<tr>
<td>LATIN5</td>
<td>ISO 8859-9, ECMA 128</td>
<td>Turkish</td>
<td>Yes</td>
<td>1</td>
<td>ISO88599</td>
</tr>
<tr>
<td>LATIN6</td>
<td>ISO 8859-10, ECMA 144</td>
<td>Nordic</td>
<td>Yes</td>
<td>1</td>
<td>ISO885910</td>
</tr>
<tr>
<td>LATIN7</td>
<td>ISO 8859-13</td>
<td>Baltic</td>
<td>Yes</td>
<td>1</td>
<td>ISO885913</td>
</tr>
<tr>
<td>LATIN8</td>
<td>ISO 8859-14</td>
<td>Celtic</td>
<td>Yes</td>
<td>1</td>
<td>ISO885914</td>
</tr>
<tr>
<td>LATIN9</td>
<td>ISO 8859-15</td>
<td>LATIN1 with Euro and accents</td>
<td>Yes</td>
<td>1</td>
<td>ISO885915</td>
</tr>
<tr>
<td>LATIN10</td>
<td>ISO 8859-16, ASRO SR 14111</td>
<td>Romanian</td>
<td>Yes</td>
<td>1</td>
<td>ISO885916</td>
</tr>
<tr>
<td>MULE_INTERNAL</td>
<td>Mule internal code</td>
<td>Multilingual Emacs</td>
<td>Yes</td>
<td>1-4</td>
<td></td>
</tr>
<tr>
<td>SJIS</td>
<td>Shift JIS</td>
<td>Japanese</td>
<td>No</td>
<td>1-2</td>
<td>Mskanji, ShiftJIS, WIN932, Windows932</td>
</tr>
<tr>
<td>SQL_ASCII</td>
<td>unspecified</td>
<td>any</td>
<td>No</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>UHC</td>
<td>Unified Hangul Code</td>
<td>Korean</td>
<td>No</td>
<td>1-2</td>
<td>WIN949, Windows949</td>
</tr>
<tr>
<td>UTF8</td>
<td>Unicode, 8-bit</td>
<td>all</td>
<td>Yes</td>
<td>1-4</td>
<td>Unicode</td>
</tr>
<tr>
<td>WIN866</td>
<td>Windows CP866</td>
<td>Cyrillic</td>
<td>Yes</td>
<td>1</td>
<td>ALT</td>
</tr>
<tr>
<td>WIN874</td>
<td>Windows CP874</td>
<td>Thai</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>WIN1250</td>
<td>Windows CP1250</td>
<td>Central European</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>WIN1251</td>
<td>Windows CP1251</td>
<td>Cyrillic</td>
<td>Yes</td>
<td>1</td>
<td>WIN</td>
</tr>
<tr>
<td>WIN1252</td>
<td>Windows CP1252</td>
<td>Western European</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

11 The SQL_ASCII setting behaves considerably differently from the other settings. Byte values 0-127 are interpreted according to the ASCII standard, while byte values 128-255 are taken as uninterpreted characters. If you are working with any non-ASCII data, it is unwise to use the SQL_ASCII setting as a client encoding. SQL_ASCII is not supported as a server encoding.
Setting the Character Set

gpinitsystem defines the default character set for a Greenplum Database system by reading the setting of the ENCODING parameter in the gp_init_config file at initialization time. The default character set is UNICODE or UTF8.

You can create a database with a different character set besides what is used as the system-wide default. For example:

```sql
=> CREATE DATABASE korean WITH ENCODING 'EUC_KR';
```

Important: Although you can specify any encoding you want for a database, it is unwise to choose an encoding that is not what is expected by the locale you have selected. The LC_COLLATE and LC_CTYPE settings imply a particular encoding, and locale-dependent operations (such as sorting) are likely to misinterpret data that is in an incompatible encoding.

Since these locale settings are frozen by gpinitsystem, the apparent flexibility to use different encodings in different databases is more theoretical than real.

One way to use multiple encodings safely is to set the locale to C or POSIX during initialization time, thus disabling any real locale awareness.

Character Set Conversion Between Server and Client

Greenplum Database supports automatic character set conversion between server and client for certain character set combinations. The conversion information is stored in the master pg_conversion system catalog table. Greenplum Database comes with some predefined conversions or you can create a new conversion using the SQL command CREATE CONVERSION.

Table 225: Client/Server Character Set Conversions

<table>
<thead>
<tr>
<th>Server Character Set</th>
<th>Available Client Character Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIG5</td>
<td>not supported as a server encoding</td>
</tr>
<tr>
<td>EUC_CN</td>
<td>EUC_CN, MULE_INTERNAL, UTF8</td>
</tr>
<tr>
<td>EUC_JP</td>
<td>EUC_JP, MULE_INTERNAL, SJIS, UTF8</td>
</tr>
<tr>
<td>EUC_KR</td>
<td>EUC_KR, MULE_INTERNAL, UTF8</td>
</tr>
<tr>
<td>Server Character Set</td>
<td>Available Client Character Sets</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EUC_TW</td>
<td>EUC_TW, BIG5, MULE_INTERNAL, UTF8</td>
</tr>
<tr>
<td>GB18030</td>
<td>not supported as a server encoding</td>
</tr>
<tr>
<td>GBK</td>
<td>not supported as a server encoding</td>
</tr>
<tr>
<td>ISO_8859_5</td>
<td>ISO_8859_5, KOI8, MULE_INTERNAL, UTF8, WIN866, WIN1251</td>
</tr>
<tr>
<td>ISO_8859_6</td>
<td>ISO_8859_6, UTF8</td>
</tr>
<tr>
<td>ISO_8859_7</td>
<td>ISO_8859_7, UTF8</td>
</tr>
<tr>
<td>ISO_8859_8</td>
<td>ISO_8859_8, UTF8</td>
</tr>
<tr>
<td>JOHAB</td>
<td>JOHAB, UTF8</td>
</tr>
<tr>
<td>KOI8</td>
<td>KOI8, ISO_8859_5, MULE_INTERNAL, UTF8, WIN866, WIN1251</td>
</tr>
<tr>
<td>LATIN1</td>
<td>LATIN1, MULE_INTERNAL, UTF8</td>
</tr>
<tr>
<td>LATIN2</td>
<td>LATIN2, MULE_INTERNAL, UTF8, WIN1250</td>
</tr>
<tr>
<td>LATIN3</td>
<td>LATIN3, MULE_INTERNAL, UTF8</td>
</tr>
<tr>
<td>LATIN4</td>
<td>LATIN4, MULE_INTERNAL, UTF8</td>
</tr>
<tr>
<td>LATIN5</td>
<td>LATIN5, UTF8</td>
</tr>
<tr>
<td>LATIN6</td>
<td>LATIN6, UTF8</td>
</tr>
<tr>
<td>LATIN7</td>
<td>LATIN7, UTF8</td>
</tr>
<tr>
<td>LATIN8</td>
<td>LATIN8, UTF8</td>
</tr>
<tr>
<td>LATIN9</td>
<td>LATIN9, UTF8</td>
</tr>
<tr>
<td>LATIN10</td>
<td>LATIN10, UTF8</td>
</tr>
<tr>
<td>MULE_INTERNAL</td>
<td>MULE_INTERNAL, BIG5, EUC_CN, EUC_JP, EUC_KR, EUC_TW, ISO_8859_5, KOI8, LATIN1 to LATIN4, SJIS, WIN866, WIN1250</td>
</tr>
<tr>
<td>SJIS</td>
<td>not supported as a server encoding</td>
</tr>
<tr>
<td>SQL_ASCII</td>
<td>not supported as a server encoding</td>
</tr>
<tr>
<td>UHC</td>
<td>not supported as a server encoding</td>
</tr>
<tr>
<td>UTF8</td>
<td>all supported encodings</td>
</tr>
<tr>
<td>WIN866</td>
<td>WIN866</td>
</tr>
<tr>
<td>ISO_8859_5</td>
<td>KOI8, MULE_INTERNAL, UTF8, WIN1251</td>
</tr>
<tr>
<td>WIN874</td>
<td>WIN874, UTF8</td>
</tr>
<tr>
<td>WIN1250</td>
<td>WIN1250, LATIN2, MULE_INTERNAL, UTF8</td>
</tr>
<tr>
<td>WIN1251</td>
<td>WIN1251, ISO_8859_5, KOI8, MULE_INTERNAL, UTF8, WIN866</td>
</tr>
<tr>
<td>WIN1252</td>
<td>WIN1252, UTF8</td>
</tr>
<tr>
<td>WIN1253</td>
<td>WIN1253, UTF8</td>
</tr>
</tbody>
</table>
To enable automatic character set conversion, you have to tell Greenplum Database the character set (encoding) you would like to use in the client. There are several ways to accomplish this:

- **Using the \encoding command in psql**, which allows you to change client encoding on the fly.
- **Using SET client_encoding TO**.

To set the client encoding, use the following SQL command:

```sql
=> SET CLIENT_ENCODING TO 'value';
```

To query the current client encoding:

```sql
=> SHOW client_encoding;
```

To return to the default encoding:

```sql
=> RESET client_encoding;
```

- **Using the PGCLIENTENCODING environment variable**. When PGCLIENTENCODING is defined in the client’s environment, that client encoding is automatically selected when a connection to the server is made. (This can subsequently be overridden using any of the other methods mentioned above.)
- **Setting the configuration parameter client_encoding**. If client_encoding is set in the master postgresql.conf file, that client encoding is automatically selected when a connection to Greenplum Database is made. (This can subsequently be overridden using any of the other methods mentioned above.)

If the conversion of a particular character is not possible " suppose you chose EUC_JP for the server and LATIN1 for the client, then some Japanese characters do not have a representation in LATIN1 " then an error is reported.

If the client character set is defined as SQL_ASCII, encoding conversion is disabled, regardless of the server’s character set. The use of SQL_ASCII is unwise unless you are working with all-ASCII data. SQL_ASCII is not supported as a server encoding.

### Available Client Character Sets

<table>
<thead>
<tr>
<th>Server Character Set</th>
<th>Available Client Character Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIN1254</td>
<td>WIN1254, UTF8</td>
</tr>
<tr>
<td>WIN1255</td>
<td>WIN1255, UTF8</td>
</tr>
<tr>
<td>WIN1256</td>
<td>WIN1256, UTF8</td>
</tr>
<tr>
<td>WIN1257</td>
<td>WIN1257, UTF8</td>
</tr>
<tr>
<td>WIN1258</td>
<td>WIN1258, UTF8</td>
</tr>
</tbody>
</table>
Server Configuration Parameters

There are many Greenplum server configuration parameters that affect the behavior of the Greenplum Database system. Many of these configuration parameters have the same names, settings, and behaviors as in a regular PostgreSQL database system.

- **Parameter Types and Values** describes the parameter data types and values.
- **Setting Parameters** describes limitations on who can change them and where or when they can be set.
- **Parameter Categories** organizes parameters by functionality.
- **Configuration Parameters** lists the parameter descriptions in alphabetic order.

**Parameter Types and Values**

All parameter names are case-insensitive. Every parameter takes a value of one of four types: Boolean, integer, floating point, or string. Boolean values may be written as ON, OFF, TRUE, FALSE, YES, NO, 1, 0 (all case-insensitive).

Some settings specify a memory size or time value. Each of these has an implicit unit, which is either kilobytes, blocks (typically eight kilobytes), milliseconds, seconds, or minutes. Valid memory size units are kB (kilobytes), MB (megabytes), and GB (gigabytes). Valid time units are ms (milliseconds), s (seconds), min (minutes), h (hours), and d (days). Note that the multiplier for memory units is 1024, not 1000. A valid time expression contains a number and a unit. When specifying a memory or time unit using the `SET` command, enclose the value in quotes. For example:

```
SET statement_mem TO '200MB';
```

**Note:** There is no space between the value and the unit names.

**Setting Parameters**

Many of the configuration parameters have limitations on who can change them and where or when they can be set. For example, to change certain parameters, you must be a Greenplum Database superuser. Other parameters require a restart of the system for the changes to take effect. A parameter that is classified as `session` can be set at the system level (in the `postgresql.conf` file), at the database-level (using `ALTER DATABASE`), at the role-level (using `ALTER ROLE`), or at the session-level (using `SET`). System parameters can only be set in the `postgresql.conf` file.

In Greenplum Database, the master and each segment instance has its own `postgresql.conf` file (located in their respective data directories). Some parameters are considered `local` parameters, meaning that each segment instance looks to its own `postgresql.conf` file to get the value of that parameter. You must set local parameters on every instance in the system (master and segments). Others parameters are considered `master` parameters. Master parameters need only be set at the master instance.

This table describes the values in the Settable Classifications column of the table in the description of a server configuration parameter.
Table 226: Settable Classifications

<table>
<thead>
<tr>
<th>Set Classification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>master or local</td>
<td>A <em>master</em> parameter only needs to be set in the <code>postgresql.conf</code> file of the Greenplum master instance. The value for this parameter is then either passed to (or ignored by) the segments at run time. A local parameter must be set in the <code>postgresql.conf</code> file of the master AND each segment instance. Each segment instance looks to its own configuration to get the value for the parameter. Local parameters always require a system restart for changes to take effect.</td>
</tr>
<tr>
<td>session or system</td>
<td><em>Session</em> parameters can be changed on the fly within a database session, and can have a hierarchy of settings: at the system level (<code>postgresql.conf</code>), at the database level (<code>ALTER DATABASE...SET</code>), at the role level (<code>ALTER ROLE...SET</code>), or at the session level (<code>SET</code>). If the parameter is set at multiple levels, then the most granular setting takes precedence (for example, session overrides role, role overrides database, and database overrides system). A <em>system</em> parameter can only be changed via the <code>postgresql.conf file(s)</code>.</td>
</tr>
<tr>
<td>restart or reload</td>
<td>When changing parameter values in the <code>postgresql.conf</code> file(s), some require a <em>restart</em> of Greenplum Database for the change to take effect. Other parameter values can be refreshed by just reloading the server configuration file (using <code>gpstop -u</code>), and do not require stopping the system.</td>
</tr>
<tr>
<td>superuser</td>
<td>These session parameters can only be set by a database superuser. Regular database users cannot set this parameter.</td>
</tr>
<tr>
<td>read only</td>
<td>These parameters are not settable by database users or superusers. The current value of the parameter can be shown but not altered.</td>
</tr>
</tbody>
</table>

Parameter Categories

Configuration parameters affect categories of server behaviors, such as resource consumption, query tuning, and authentication. The following topics describe Greenplum configuration parameter categories.

- Connection and Authentication Parameters
- System Resource Consumption Parameters
- GPORCA Parameters
- Query Tuning Parameters
- Error Reporting and Logging Parameters
- System Monitoring Parameters
- Runtime Statistics Collection Parameters
Connection and Authentication Parameters

These parameters control how clients connect and authenticate to Greenplum Database.

Connection Parameters

- \texttt{gp\_connection\_send\_timeout}  
- \texttt{gp\_vmem\_idle\_resource\_timeout}  
- \texttt{listen\_addresses}  
- \texttt{max\_connections}  
- \texttt{max\_prepared\_transactions}  
- \texttt{superuser\_reserved\_connections}  
- \texttt{tcp\_keepalives\_count}  
- \texttt{tcp\_keepalives\_idle}  
- \texttt{tcp\_keepalives\_interval}  
- \texttt{unix\_socket\_directory}  
- \texttt{unix\_socket\_group}  
- \texttt{unix\_socket\_permissions}  

Security and Authentication Parameters

- \texttt{authentication\_timeout}  
- \texttt{db\_user\_namespace}  
- \texttt{krb\_caseins\_users}  
- \texttt{krb\_server\_keyfile}  
- \texttt{krb\_srvname}  
- \texttt{password\_encryption}  
- \texttt{password\_hash\_algorithm}  
- \texttt{ssl}  
- \texttt{ssl\_ciphers}  

System Resource Consumption Parameters

These parameters set the limits for system resources consumed by Greenplum Database.

Memory Consumption Parameters

These parameters control system memory usage. You can adjust \texttt{gp\_vmem\_protect\_limit} to avoid running out of memory at the segment hosts during query processing.

- \texttt{gp\_vmem\_idle\_resource\_timeout}  
- \texttt{gp\_vmem\_protect\_limit}  
- \texttt{gp\_vmem\_protect\_segworker\_cache\_limit}  
- \texttt{gp\_workfile\_limit\_files\_per\_query}  
- \texttt{max\_stack\_depth}  
- \texttt{shared\_buffers}  
- \texttt{temp\_buffers}
Free Space Map Parameters
These parameters control the sizing of the free space map, which contains expired rows. Use VACUUM to reclaim the free space map disk space.

max_fsm_pages
max_fsm_relations

OS Resource Parameters

max_files_per_process
shared_preload_libraries

Cost-Based Vacuum Delay Parameters

Warning: Do not use cost-based vacuum delay because it runs asynchronously among the segment instances. The vacuum cost limit and delay is invoked at the segment level without taking into account the state of the entire Greenplum Database array.

You can configure the execution cost of VACUUM and ANALYZE commands to reduce the I/O impact on concurrent database activity. When the accumulated cost of I/O operations reaches the limit, the process performing the operation sleeps for a while, then resets the counter and continues execution.

vacuum_cost_delay
vacuum_cost_limit
vacuum_cost_page_dirty
vacuum_cost_page_hit
vacuum_cost_page_miss

Transaction ID Management Parameters

xid_stop_limit
xid_warn_limit

GPORCA Parameters
These parameters control the usage of GPORCA by Greenplum Database. For information about GPORCA, see "Querying Data" in the Greenplum Database Administrator Guide.

optimizer
optimizer_analyze_root_partition
optimizer_array_expansion_threshold
optimizer_cte_inlining_bound
optimizer_control
optimizer_enable_master_only_queries
optimizer_force_multistage_agg
optimizer_join_arith_for_associativity_commutativity
optimizer_join_order_threshold
optimizer_mdcache_size
optimizer_metadata_caching
optimizer_parallel_union
optimizer_print_missing_stats
optimizer_print_optimization_stats
Query Tuning Parameters
These parameters control aspects of SQL query processing such as query operators and operator settings and statistics sampling.

Legacy Query Optimizer Operator Control Parameters
The following parameters control the types of plan operations the legacy query optimizer can use. Enable or disable plan operations to force the legacy optimizer to choose a different plan. This is useful for testing and comparing query performance using different plan types.

- enable_bitmapscan
- enable_groupagg
- enable_hashagg
- enable_hashjoin
- enable_indexscan
- enable_mergejoin
- enable_nestloop
- enable_seqscan
- enable_sort
- enable_tidscan
- gp_enable_adaptive_nestloop
- gp_enable_agg_distinct
- gp_enable_direct_dispatch
- gp_enable_fallback_plan
- gp_enable_fast_sri
- gp_enable_groupext_distinct_gather
- gp_enable_groupext_distinct_pruning
- gp_enable_multiphase_agg
- gp_enable_predicate_propagation
- gp_enable_preunique
- gp_enable_relsize_collection
- gp_enable_sort_distinct
- gp_enable_sort_limit

Legacy Query Optimizer Costing Parameters

**Warning**: Do not adjust these query costing parameters. They are tuned to reflect Greenplum Database hardware configurations and typical workloads. All of these parameters are related. Changing one without changing the others can have adverse affects on performance.

- cpu_index_tuple_cost
- cpu_operator_cost
- cpu_tuple_cost
- cursor_tuple_fraction
- effective_cache_size
- gp_motion_cost_per_row
- gp_segments_for_planner
- random_page_cost
- seq_page_cost

Database Statistics Sampling Parameters
These parameters adjust the amount of data sampled by an `ANALYZE` operation. Adjusting these parameters affects statistics collection system-wide. You can configure statistics collection on particular tables and columns by using the `ALTER TABLE SET STATISTICS` clause.

- default_statistics_target
gp_analyze_relative_error

**Sort Operator Configuration Parameters**

gp_enable_sort_distinct

gp_enable_sort_limit

**Aggregate Operator Configuration Parameters**

gp_enable_agg_distinct

gp_enable_agg_distinct_pruning

gp_enable_multiphase_agg

gp_enable_preunique

gp_enable_groupext_distinct_gather

gp_enable_groupext_distinct_pruning

gp_workfile_compress_algorithm

**Join Operator Configuration Parameters**

join_collapse_limit

gp_adjust_selectivity_for_outerjoins

gp_hashjoin_tuples_per_bucket

gp_statistics_use_fkeys

gp_workfile_compress_algorithm

**Other Legacy Query Optimizer Configuration Parameters**

from_collapse_limit

gp_enable_predicate_propagation

gp_max_plan_size

gp_statistics_pullup_from_child_partition

**Error Reporting and Logging Parameters**

These configuration parameters control Greenplum Database logging.

**Log Rotation**

log_rotation_age

log_rotation_size

log_truncate_on_rotation

**When to Log**

client_min_messages

gp_interconnect_debug_retry_interval

glog_error_verbosity

log_min_duration_statement

log_min_error_statement

log_min_messages

optimizer_minidump
**What to Log**

- `debug_pretty_print`
- `debug_print_parse`
- `debug_print_plan`
- `debug_print_prelim_plan`
- `debug_print_rewritten`
- `debug_print_slice_table`
- `log_autostats`
- `log_connections`
- `log_disconnections`
- `log_dispatch_stats`
- `log_duration`
- `log_executor_stats`
- `log_hostname`
- `gp_log_interconnect`
- `log_discovery_stats`
- `log_statement`
- `log_statement_stats`
- `log_timezone`
- `gp_debug_linger`
- `gp_log_format`
- `gp_max_csv_line_length`
- `gp_reraise_signal`

**System Monitoring Parameters**

These configuration parameters control Greenplum Database data collection and notifications related to database monitoring.

**SNMP Alerts**

The following parameters send SNMP notifications when events occur.

- `gp_snmp_community`
- `gp_snmp_monitor_address`
- `gp_snmp_use_inform_or_trap`

**Email Alerts**

The following parameters configure the system to send email alerts for fatal error events, such as a segment going down or a server crash and reset.

- `gp_email_from`
- `gp_email_smtp_password`
- `gp_email_smtp_server`
- `gp_email_smtp_userid`
- `gp_email_to`

**Greenplum Command Center Agent**

The following parameters configure the data collection agents that populate the `gpperfmon` database for Greenplum Command Center.

- `gp_enable_gpperfmon`
- `gpperfmon_log_alert_level`
- `gp_gpperfmon_send_interval`
- `gpperfmon_port`
Runtime Statistics Collection Parameters
These parameters control the server statistics collection feature. When statistics collection is enabled, you can access the statistics data using the `pg_stat` and `pg_statio` family of system catalog views.

- `stats_queue_level`
- `track_activities`
- `track_counts`
- `update_process_title`

Automatic Statistics Collection Parameters
When automatic statistics collection is enabled, you can run `ANALYZE` automatically in the same transaction as an `INSERT`, `UPDATE`, `DELETE`, `COPY` or `CREATE TABLE ... AS SELECT` statement when a certain threshold of rows is affected (on_change), or when a newly generated table has no statistics (on_no_stats). To enable this feature, set the following server configuration parameters in your Greenplum Database master `postgresql.conf` file and restart Greenplum Database:

- `gp_autostats_mode`
- `gp_autostats_mode_in_functions`
- `gp_autostats_on_change_threshold`
- `log_autostats`

**Warning:** Depending on the specific nature of your database operations, automatic statistics collection can have a negative performance impact. Carefully evaluate whether the default setting of `on_no_stats` is appropriate for your system.

Client Connection Default Parameters
These configuration parameters set defaults that are used for client connections.

**Statement Behavior Parameters**

- `check_function_bodies`
- `default_tablespace`
- `default_transaction_isolation`
- `default_transaction_read_only`
- `search_path`
- `statement_timeout`
- `vacuum_freeze_min_age`

**Locale and Formatting Parameters**

- `client_encoding`
- `DateStyle`
- `extra_float_digits`
- `IntervalStyle`
- `lc_collate`
- `lc_ctype`
- `lc_messages`
- `lc_monetary`
- `lc_numeric`
- `lc_time`
- `lc_type`
- `TimeZone`
Other Client Default Parameters

dynamic_library_path  local_preload_libraries
explain_pretty_print

Lock Management Parameters
These configuration parameters set limits for locks and deadlocks.

deadlock_timeout
max_locks_per_transaction

Resource Management Parameters (Resource Queues)
The following configuration parameters configure the Greenplum Database resource management feature (resource queues), query prioritization, memory utilization and concurrency control.

gp_resqueue.priority  max_resource_queues
gp_resqueue.priority_cpuscores_per_segment  max_resource_portals_per_transaction
gp_resqueue.priority_sweeper_interval  resource_cleanup_gangs_on_wait
gp_vmem_idle_resource_timeout  resource_select_only
gp_vmem.protect_limit  runaway_detector_activation_percent
gp_vmem.protect_segworker_cache_limit  stats_queue_level

Resource Management Parameters (Resource Groups)
Warning: Resource groups are an experimental feature and are not intended for use in a production environment. Experimental features are subject to change without notice in future releases.

The following parameters configure the Greenplum Database resource group workload management feature.

gp_resgroup_memory_policy  gp_resource_manager
gp_resource_group_cpu_limit  max_resource_groups
gp_resource_group_memory_limit  memory_spill_ratio

External Table Parameters
The following parameters configure the external tables feature of Greenplum Database.

gp_external_enable_exec  gp_initial_bad_row_limit
gp_external_max_segs  gp_reject_percent_threshold
readable_external_table_timeout  writable_external_table_bufsize
Database Table Parameters
The following parameter configures default option settings for Greenplum Database tables.

`gp_create_table_random_default_distribution`
`gp_default_storage_options`
`gp_enable_exchange_default_partition`
`gp_enable_segment_copy_checking`

Append-Optimized Table Parameters
The following parameters configure the append-optimized tables feature of Greenplum Database.

`max_appendonly_tables`
`gp_appendonly_compaction`
`gp_appendonly_compaction_threshold`
`validate_previous_free_tid`

Database and Tablespace/ Filespace Parameters
The following parameters configure the maximum number of databases, tablespaces, and filespaces allowed in a system.

`gp_max_tablespaces`
`gp_max_filespaces`
`gp_max_databases`

Past PostgreSQL Version Compatibility Parameters
The following parameters provide compatibility with older PostgreSQL versions. You do not need to change these parameters in Greenplum Database.

`add_missing_from`
`array_nulls`
`backslash_quote`
`escape_string_warning`

Greenplum Database Array Configuration Parameters
The parameters in this topic control the configuration of the Greenplum Database array and its components: segments, master, distributed transaction manager, master mirror, and interconnect.

Interconnect Configuration Parameters

`gp_interconnect_fc_method`
`gp_interconnect_hash_multiplier`
`gp_interconnect_setup_timeout`
`gp_interconnect_snd_queue_depth`
gp_interconnect_queue_depth  gp_interconnect_type
  gp_max_packet_size

**Note:** Greenplum Database supports only the UDPIFC (default) and TCP interconnect types.

**Dispatch Configuration Parameters**

- `gp_cached_segworkers_threshold`
- `gp_connections_per_thread`
- `gp_enable_direct_dispatch`

**Fault Operation Parameters**

- `gp_set_read_only`
- `gp_fts_probe_interval`
- `gp_fts_probe_retries`

**Distributed Transaction Management Parameters**

- `gp_max_local_distributed_cache`

**Read-Only Parameters**

- `gp_command_count`
- `gp_content`
- `gp_dbid`
- `gp_num_contents_in_cluster`

**Greenplum Mirroring Parameters for Master and Segments**

These parameters control the configuration of the replication between Greenplum Database primary master and standby master.

- `keep_wal_segments`
- `repl_catchup_within_range`
- `replication_timeout`
- `wal_receiver_status_interval`

This parameter controls validation between Greenplum Database primary segment and standby segment during incremental resynchronization.

- `filerep_mirrorvalidation_during_resync`

**Greenplum Database Extension Parameters**

The parameters in this topic control the configuration of Greenplum Database extensions.
**pljava_classpath**
**pljava_classpath_insecure**
**pljava_statement_cache_size**
**pljava_release_lingering_savepoints**
**pljava_vmoptions**

**XML Data Parameters**

xmlbinary
xmloption

**Configuration Parameters**

Descriptions of the Greenplum Database server configuration parameters listed alphabetically.

**add_missing_from**
Automatically adds missing table references to FROM clauses. Present for compatibility with releases of PostgreSQL prior to 8.1, where this behavior was allowed by default.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**application_name**
Sets the application name for a client session. For example, if connecting via `psql`, this will be set to `psql`. Setting an application name allows it to be reported in log messages and statistics views.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td></td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**array_nulls**
This controls whether the array input parser recognizes unquoted NULL as specifying a null array element. By default, this is on, allowing array values containing null values to be entered. Greenplum Database versions before 3.0 did not support null values in arrays, and therefore would treat NULL as specifying a normal array element with the string value 'NULL'.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>
**authentication_timeout**

Maximum time to complete client authentication. This prevents hung clients from occupying a connection indefinitely.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any valid time expression (number and unit)</td>
<td>1min</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>

**backslash_quote**

This controls whether a quote mark can be represented by \ in a string literal. The preferred, SQL-standard way to represent a quote mark is by doubling it (") but PostgreSQL has historically also accepted \\ . However, use of \\ creates security risks because in some client character set encodings, there are multibyte characters in which the last byte is numerically equivalent to ASCII \\. 

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>on (allow \ always)</td>
<td>safe_encoding</td>
<td>master, session, reload</td>
</tr>
<tr>
<td>off (reject always)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>safe_encoding (allow only if client encoding does not allow ASCII \ within a multibyte character)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**block_size**

Reports the size of a disk block.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of bytes</td>
<td>32768</td>
<td>read only</td>
</tr>
</tbody>
</table>

**bonjour_name**

Specifies the Bonjour broadcast name. By default, the computer name is used, specified as an empty string. This option is ignored if the server was not compiled with Bonjour support.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>unset</td>
<td>master, system, restart</td>
</tr>
</tbody>
</table>

**check_function_bodies**

When set to off, disables validation of the function body string during `CREATE FUNCTION`. Disabling validation is occasionally useful to avoid problems such as forward references when restoring function definitions from a dump.
### client_encoding

Sets the client-side encoding (character set). The default is to use the same as the database encoding. See [Supported Character Sets](#) in the PostgreSQL documentation.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>character set</td>
<td>UTF8</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

### client_min_messages

Controls which message levels are sent to the client. Each level includes all the levels that follow it. The later the level, the fewer messages are sent.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG5</td>
<td>NOTICE</td>
<td>master</td>
</tr>
<tr>
<td>DEBUG4</td>
<td></td>
<td>session</td>
</tr>
<tr>
<td>DEBUG3</td>
<td></td>
<td>reload</td>
</tr>
<tr>
<td>DEBUG2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEBUG1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOTICE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WARNING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERROR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FATAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PANIC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### cpu_index_tuple_cost

For the legacy query optimizer (planner), sets the estimate of the cost of processing each index row during an index scan. This is measured as a fraction of the cost of a sequential page fetch.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>floating point</td>
<td>0.005</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>
**cpu_operator_cost**

For the legacy query optimizer (planner), sets the estimate of the cost of processing each operator in a WHERE clause. This is measured as a fraction of the cost of a sequential page fetch.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>floating point</td>
<td>0.0025</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

**cpu_tuple_cost**

For the legacy query optimizer (planner), sets the estimate of the cost of processing each row during a query. This is measured as a fraction of the cost of a sequential page fetch.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>floating point</td>
<td>0.01</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

**cursor_tuple_fraction**

Tells the legacy query optimizer (planner) how many rows are expected to be fetched in a cursor query, thereby allowing the legacy optimizer to use this information to optimize the query plan. The default of 1 means all rows will be fetched.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>1</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

**custom_variable_classes**

Specifies one or several class names to be used for custom variables. A custom variable is a variable not normally known to the server but used by some add-on modules. Such variables must have names consisting of a class name, a dot, and a variable name.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>comma-separated list of class names</td>
<td>unset</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>

**data_checksums**

Reports whether checksums are enabled for heap data storage in the database system. Checksums for heap data are enabled or disabled when the database system is initialized and cannot be changed.
Heap data pages store heap tables, catalog tables, indexes, and database metadata. Append-optimized storage has built-in checksum support that is unrelated to this parameter.

Greenplum Database uses checksums to prevent loading data corrupted in the file system into memory managed by database processes. When heap data checksums are enabled, Greenplum Database computes and stores checksums on heap data pages when they are written to disk. When a page is retrieved from disk, the checksum is verified. If the verification fails, an error is generated and the page is not permitted to load into managed memory.

If the `ignore_checksum_failure` configuration parameter has been set to on, a failed checksum verification generates a warning, but the page is allowed to be loaded into managed memory. If the page is then updated, it is flushed to disk and replicated to the mirror. This can cause data corruption to propagate to the mirror and prevent a complete recovery. Because of the potential for data loss, the `ignore_checksum_failure` parameter should only be enabled when needed to recover data. See `ignore_checksum_failure` for more information.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>read only</td>
</tr>
</tbody>
</table>

**DateStyle**

Sets the display format for date and time values, as well as the rules for interpreting ambiguous date input values. This variable contains two independent components: the output format specification and the input/output specification for year/month/day ordering.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;format&gt;, &lt;date style&gt;</code> where: &lt;format&gt; is ISO, Postgres, SQL, or German &lt;date style&gt; is DMY, MDY, or YMD</td>
<td>ISO, MDY</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

**db_user_namespace**

This enables per-database user names. If on, you should create users as `username@dbname`. To create ordinary global users, simply append @ when specifying the user name in the client.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>

**deadlock_timeout**

The time to wait on a lock before checking to see if there is a deadlock condition. On a heavily loaded server you might want to raise this value. Ideally the setting should exceed your typical transaction time, so as to improve the odds that a lock will be released before the waiter decides to check for deadlock.
### debug_assertions

Turns on various assertion checks.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any valid time expression (number and unit)</td>
<td>1s</td>
<td>local system restart</td>
</tr>
</tbody>
</table>

### debug_pretty_print

Indents debug output to produce a more readable but much longer output format. `client_min_messages` or `log_min_messages` must be DEBUG1 or lower.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

### debug_print_parse

For each executed query, prints the resulting parse tree. `client_min_messages` or `log_min_messages` must be DEBUG1 or lower.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

### debug_print_plan

For each executed query, prints the Greenplum parallel query execution plan. `client_min_messages` or `log_min_messages` must be DEBUG1 or lower.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master session reload</td>
</tr>
</tbody>
</table>
debug_print_prelim_plan
For each executed query, prints the preliminary query plan. client_min-messages or log_min-messages must be DEBUG1 or lower.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

debug_print_rewritten
For each executed query, prints the query rewriter output. client_min-messages or log_min-messages must be DEBUG1 or lower.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

debug_print_slice_table
For each executed query, prints the Greenplum query slice plan. client_min-messages or log_min-messages must be DEBUG1 or lower.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

default_statistics_target
Sets the default statistics target for table columns that have not had a column-specific target set via ALTER TABLE SET STATISTICS. Larger values increase the time needed to do ANALYZE, but may improve the quality of the legacy query optimizer (planner) estimates.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>100</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

default_tablespace
The default tablespace in which to create objects (tables and indexes) when a CREATE command does not explicitly specify a tablespace.
### default_transaction_isolation

Controls the default isolation level of each new transaction.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>read committed</td>
<td>read committed</td>
<td>master, session, reload</td>
</tr>
<tr>
<td>read uncommitted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>serializable</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### default_transaction_read_only

Controls the default read-only status of each new transaction. A read-only SQL transaction cannot alter non-temporary tables.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

### dynamic_library_path

If a dynamically loadable module needs to be opened and the file name specified in the `CREATE FUNCTION` or `LOAD` command does not have a directory component (i.e. the name does not contain a slash), the system will search this path for the required file. The compiled-in PostgreSQL package library directory is substituted for `$libdir`. This is where the modules provided by the standard PostgreSQL distribution are installed.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>a list of absolute directory paths separated by colons</td>
<td>$libdir</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>

### effective_cache_size

Sets the assumption about the effective size of the disk cache that is available to a single query for the legacy query optimizer (planner). This is factored into estimates of the cost of using an index; a higher value makes it more likely index scans will be used, a lower value makes it more likely sequential scans will be used. This parameter has no effect on the size of shared memory allocated by a Greenplum server instance, nor does it reserve kernel disk cache; it is used only for estimation purposes.
<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>floating point</td>
<td>512MB</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

**enable_bitmapscan**

Enables or disables the use of bitmap-scan plan types by the legacy query optimizer (planner). Note that this is different than a Bitmap Index Scan. A Bitmap Scan means that indexes will be dynamically converted to bitmaps in memory when appropriate, giving faster index performance on complex queries against very large tables. It is used when there are multiple predicates on different indexed columns. Each bitmap per column can be compared to create a final list of selected tuples.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

**enable_groupagg**

Enables or disables the use of group aggregation plan types by the legacy query optimizer (planner).

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

**enable_hashagg**

Enables or disables the use of hash aggregation plan types by the legacy query optimizer (planner).

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

**enable_hashjoin**

Enables or disables the use of hash-join plan types by the legacy query optimizer (planner).

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master session reload</td>
</tr>
</tbody>
</table>
enable_indexscan

Enables or disables the use of index-scan plan types by the legacy query optimizer (planner).

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

enable_mergejoin

Enables or disables the use of merge-join plan types by the legacy query optimizer (planner). Merge join is based on the idea of sorting the left- and right-hand tables into order and then scanning them in parallel. So, both data types must be capable of being fully ordered, and the join operator must be one that can only succeed for pairs of values that fall at the 'same place' in the sort order. In practice this means that the join operator must behave like equality.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

enable_nestloop

Enables or disables the use of nested-loop join plans by the legacy query optimizer (planner). It's not possible to suppress nested-loop joins entirely, but turning this variable off discourages the legacy optimizer from using one if there are other methods available.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

enable_seqscan

Enables or disables the use of sequential scan plan types by the legacy query optimizer (planner). It's not possible to suppress sequential scans entirely, but turning this variable off discourages the legacy optimizer from using one if there are other methods available.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>
**enable_sort**

Enables or disables the use of explicit sort steps by the legacy query optimizer (planner). It's not possible to suppress explicit sorts entirely, but turning this variable off discourages the legacy optimizer from using one if there are other methods available.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

**enable_tidscan**

Enables or disables the use of tuple identifier (TID) scan plan types by the legacy query optimizer (planner).

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

**escape_string_warning**

When on, a warning is issued if a backslash (\) appears in an ordinary string literal (‘...’ syntax). Escape string syntax (E'...') should be used for escapes, because in future versions, ordinary strings will have the SQL standard-conforming behavior of treating backslashes literally.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

**explain.pretty_print**

Determines whether EXPLAIN VERBOSE uses the indented or non-indented format for displaying detailed query-tree dumps.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

**extra_float_digits**

Adjusts the number of digits displayed for floating-point values, including float4, float8, and geometric data types. The parameter value is added to the standard number of digits. The value can be set as high as 2, to include partially-significant digits; this is especially useful for dumping float data that needs to be restored exactly. Or it can be set negative to suppress unwanted digits.
**filerep_mirrorvalidation_during_resync**

The default setting `false` improves Greenplum Database performance during incremental resynchronization of segment mirrors. Setting the value to `true` enables checking for the existence of files for all relations on the segment mirror during incremental resynchronization. Checking for files degrades performance of the incremental resynchronization process but provides a minimal check of database objects.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>false</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**fromCollapseLimit**

The legacy query optimizer (planner) will merge sub-queries into upper queries if the resulting FROM list would have no more than this many items. Smaller values reduce planning time but may yield inferior query plans.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-(n)</td>
<td>20</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**gp_adjust_selectivity_for_outerjoins**

Enables the selectivity of NULL tests over outer joins.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**gp_analyze_relative_error**

Sets the estimated acceptable error in the cardinality of the table; a value of 0.5 is supposed to be equivalent to an acceptable error of 50% (this is the default value used in PostgreSQL). If the statistics collected during `ANALYZE` are not producing good estimates of cardinality for a particular table attribute, decreasing the relative error fraction (accepting less error) tells the system to sample more rows.
<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>floating point &lt; 1.0</td>
<td>0.25</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

**gp_appendonly_compaction**

Enables compacting segment files during VACUUM commands. When disabled, VACUUM only truncates the segment files to the EOF value, as is the current behavior. The administrator may want to disable compaction in high I/O load situations or low space situations.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

**gp_appendonly_compaction_threshold**

Specifies the threshold ratio (as a percentage) of hidden rows to total rows that triggers compaction of the segment file when VACUUM is run without the FULL option (a lazy vacuum). If the ratio of hidden rows in a segment file on a segment is less than this threshold, the segment file is not compacted, and a log message is issued.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer (%)</td>
<td>10</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

**gp_autostats_mode**

Specifies the mode for triggering automatic statistics collection with ANALYZE. The on_no_stats option triggers statistics collection for CREATE TABLE AS SELECT, INSERT, or COPY operations on any table that has no existing statistics.

The on_change option triggers statistics collection only when the number of rows affected exceeds the threshold defined by gp_autostats_on_change_threshold. Operations that can trigger automatic statistics collection with on_change are:

CREATE TABLE AS SELECT
UPDATE
DELETE
INSERT
COPY

**Default is on_no_stats.**

**Note:** For partitioned tables, automatic statistics collection is not triggered if data is inserted from the top-level parent table of a partitioned table.

Automatic statistics collection is triggered if data is inserted directly in a leaf table (where the data is stored) of the partitioned table. Statistics are collected only on the leaf table.
**gp_autostats_mode_in_functions**

Specifies the mode for triggering automatic statistics collection with `ANALYZE` for statements in procedural language functions. The `none` option disables statistics collection. The `on_no_stats` option triggers statistics collection for `CREATE TABLE AS SELECT`, `INSERT`, or `COPY` operations that are executed in functions on any table that has no existing statistics.

The `on_change` option triggers statistics collection only when the number of rows affected exceeds the threshold defined by `gp_autostats_on_change_threshold`. Operations in functions that can trigger automatic statistics collection with `on_change` are:

- `CREATE TABLE AS SELECT`
- `UPDATE`
- `DELETE`
- `INSERT`
- `COPY`

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>on_no_stats</td>
<td>master</td>
</tr>
<tr>
<td>on_change</td>
<td></td>
<td>session</td>
</tr>
<tr>
<td>on_no_stats</td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**gp_autostats_on_change_threshold**

Specifies the threshold for automatic statistics collection when `gp_autostats_mode` is set to `on_change`. When a triggering table operation affects a number of rows exceeding this threshold, `ANALYZE` is added and statistics are collected for the table.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>2147483647</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**gp_backup_directIO**

Direct I/O allows Greenplum Database to bypass the buffering of memory within the file system cache for backup. When Direct I/O is used for a file, data is transferred directly from the disk to the application buffer, without the use of the file buffer cache.

Direct I/O is supported only on Red Hat Enterprise Linux, CentOS, and SUSE.
### gp_backup_directIO_read_chunk_mb

Sets the chunk size in MB when Direct I/O is enabled with `gp_backup_directIO`. The default chunk size is 20MB.

The default value is the optimal setting. Decreasing it will increase the backup time and increasing it will result in little change to backup time.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-200</td>
<td>20 MB</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

### gp_cached_segworkers_threshold

When a user starts a session with Greenplum Database and issues a query, the system creates groups or 'gangs' of worker processes on each segment to do the work. After the work is done, the segment worker processes are destroyed except for a cached number which is set by this parameter. A lower setting conserves system resources on the segment hosts, but a higher setting may improve performance for power-users that want to issue many complex queries in a row.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer &gt; 0</td>
<td>5</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

### gp_command_count

Shows how many commands the master has received from the client. Note that a single SQL command might actually involve more than one command internally, so the counter may increment by more than one for a single query. This counter also is shared by all of the segment processes working on the command.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer &gt; 0</td>
<td>1</td>
<td>read only</td>
</tr>
</tbody>
</table>

### gp_connection_send_timeout

Timeout for sending data to unresponsive Greenplum Database user clients during query processing. A value of 0 disables the timeout, Greenplum Database waits indefinitely for a client. When the timeout is reached, the query is cancelled with this message:

> Could not send data to client: Connection timed out.
<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of seconds</td>
<td>3600 (1 hour)</td>
<td>master, system, reload</td>
</tr>
</tbody>
</table>

**gp_connections_per_thread**

Controls the number of asynchronous threads (worker threads) that a Greenplum Database query dispatcher (QD) generates when dispatching work to query executor processes on segment instances when processing SQL queries. The value sets the number of primary segment instances that a worker thread connects to when processing a query. For example, when the value is 2 and there are 64 segment instances, a QD generates 32 worker threads to dispatch a query plan work. Each thread is assigned to two segments.

For the default value, 0, a query dispatcher generates two types of threads: a main thread that manages the dispatch of query plan work, and an interconnect thread. The main thread also acts as a worker thread.

For a value greater than 0, a QD generates three types of threads: a main thread, one or more worker threads, and an interconnect thread. When the value is equal to or greater than the number of segment instances, a QD generates three threads: a main thread, a single worker thread, and an interconnect thread.

The value does not need to be changed from the default unless there are known throughput performance issues.

This parameter is master only and changing it requires a server restart.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer &gt;= 0</td>
<td>0</td>
<td>master, restart</td>
</tr>
</tbody>
</table>

**gp_content**

The local content id if a segment.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td></td>
<td>read only</td>
</tr>
</tbody>
</table>

**gp_create_table_random_default_distribution**

Controls table creation when a Greenplum Database table is created with a `CREATE TABLE` or `CREATE TABLE AS` command that does not contain a `DISTRIBUTED BY` clause.

For `CREATE TABLE`, if the value of the parameter is `off` (the default), and the table creation command does not contain a `DISTRIBUTED BY` clause, Greenplum Database chooses the table distribution key based on the command. If the `LIKE` or `INHERITS` clause is specified in table creation command, the created table uses the same distribution key as the source or parent table.

If the value of the parameter is set to `on`, Greenplum Database follows these rules to create a table when the `DISTRIBUTED BY` clause is not specified:

- If `PRIMARY KEY` or `UNIQUE` columns are not specified, the distribution of the table is random (`DISTRIBUTED RANDOMLY`). Table distribution is random even if the table creation command contains the `LIKE` or `INHERITS` clause.
• If `PRIMARY KEY` or `UNIQUE` columns are specified, a `DISTRIBUTED BY` clause must also be specified. If a `DISTRIBUTED BY` clause is not specified as part of the table creation command, the command fails.

For a `CREATE TABLE AS` command that does not contain a distribution clause:

• If the legacy query optimizer creates the table, and the value of the parameter is `off`, the table distribution policy is determined based on the command.
• If the legacy query optimizer creates the table, and the value of the parameter is `on`, the table distribution policy is random.
• If GPORCA creates the table, the table distribution policy is random. The parameter value has no affect.

For information about the legacy query optimizer and GPORCA, see "Querying Data" in the Greenplum Database Administrator Guide.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>off</td>
<td>master system reload</td>
</tr>
</tbody>
</table>

**gp_dbid**

The local content dbid if a segment.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td></td>
<td>read only</td>
</tr>
</tbody>
</table>

**gp_debug_linger**

Number of seconds for a Greenplum process to linger after a fatal internal error.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any valid time expression (number and unit)</td>
<td>0</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

**gp_default_storage_options**

Set the default values for the following table storage options when a table is created with the `CREATE TABLE` command.

- `APPENDONLY`
- `BLOCKSIZE`
- `CHECKSUM`
- `COMPRESSTYPE`
- `COMPRESSLEVEL`
- `ORIENTATION`

Specify multiple storage option values as a comma separated list.

You can set the storage options with this parameter instead of specifying the table storage options in the `WITH` of the `CREATE TABLE` command. The table storage options that are specified with the `CREATE TABLE` command override the values specified by this parameter.
Not all combinations of storage option values are valid. If the specified storage options are not valid, an error is returned. See the CREATE TABLE command for information about table storage options.

The defaults can be set for a database and user. If the server configuration parameter is set at different levels, this the order of precedence, from highest to lowest, of the table storage values when a user logs into a database and creates a table:

1. The values specified in a CREATE TABLE command with the WITH clause or ENCODING clause
2. The value of gp_default_storage_options that set for the user with the ALTER ROLE...SET command
3. The value of gp_default_storage_options that is set for the database with the ALTER DATABASE...SET command
4. The value of gp_default_storage_options that is set for the Greenplum Database system with the gpconfig utility

The parameter value is not cumulative. For example, if the parameter specifies the APPENDONLY and COMPRESSTYPE options for a database and a user logs in and sets the parameter to specify the value for the ORIENTATION option, the APPENDONLY, and COMPRESSTYPE values set at the database level are ignored.

This example ALTER DATABASE command sets the default ORIENTATION and COMPRESSTYPE table storage options for the database mystest.

```
ALTER DATABASE mytest SET gp_default_storage_options = 'orientation=column,
compresstype=rle_type'
```

To create an append-optimized table in the mystest database with column-oriented table and RLE compression. The user needs to specify only APPENDONLY=TRUE in the WITH clause.

This example gpconfig utility command sets the default storage option for a Greenplum Database system. If you set the defaults for multiple table storage options, the value must be enclosed in single quotes.

```
gpconfig -c 'gp_default_storage_options' -v 'appendonly=true,
orientation=column'
```

This example gpconfig utility command shows the value of the parameter. The parameter value must be consistent across the Greenplum Database master and all segments.

```
gpconfig -s 'gp_default_storage_options'
```

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications ¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPENDONLY= TRUE</td>
<td>FALSE</td>
<td>APPENDONLY=FALSE</td>
</tr>
<tr>
<td>BLOCKSIZE= integer between 8192 and 2097152</td>
<td>BLOCKSIZE=32768</td>
<td>master</td>
</tr>
<tr>
<td>CHECKSUM= TRUE</td>
<td>FALSE</td>
<td>CHECKSUM=TRUE</td>
</tr>
<tr>
<td>COMPRESSTYPE= ZLIB</td>
<td>QUICKLZ ²</td>
<td>NONE</td>
</tr>
<tr>
<td>COMPRESSLEVEL= integer between 0 and 9</td>
<td>COMPRESSLEVEL=0</td>
<td></td>
</tr>
<tr>
<td>ORIENTATION= ROW</td>
<td>COLUMN</td>
<td>ORIENTATION=ROW</td>
</tr>
</tbody>
</table>

Note: ¹The set classification when the parameter is set at the system level with the gpconfig utility.
**Note:** QuickLZ compression is available only in the commercial release of Pivotal Greenplum Database.

### gp_dynamic_partition_pruning

Enables plans that can dynamically eliminate the scanning of partitions.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>on/off</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

### gp_email_from

The email address used to send email alerts, in the format of:

'username@example.com'

or

'Name <username@example.com>'

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td></td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>superuser</td>
</tr>
</tbody>
</table>

### gp_email_smtp_password

The password/passphrase used to authenticate with the SMTP server.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td></td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>superuser</td>
</tr>
</tbody>
</table>

### gp_email_smtp_server

The fully qualified domain name or IP address and port of the SMTP server to use to send the email alerts. Must be in the format of:

smtp_servername.domain.com:port
### gp_email_smtp_userid

The user id used to authenticate with the SMTP server.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td></td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>superuser</td>
</tr>
</tbody>
</table>

### gp_email_to

A semi-colon (:) separated list of email addresses to receive email alert messages to in the format of:

'username@example.com'

or

'Name <username@example.com>'

If this parameter is not set, then email alerts are disabled.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td></td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>superuser</td>
</tr>
</tbody>
</table>

### gp_enable_adaptive_nestloop

Enables the use a new type of join node called "Adaptive Nestloop" at query execution time by the legacy query optimizer (planner). This causes the legacy optimizer to favor a hash-join over a nested-loop join if the number of rows on the outer side of the join exceeds a precalculated threshold. This parameter improves performance of index operations, which previously favored slower nested-loop joins.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>
gp_enable_agg_distinct

Enables or disables two-phase aggregation to compute a single distinct-qualified aggregate. This applies only to subqueries that include a single distinct-qualified aggregate function.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

gp_enable_agg_distinct_pruning

Enables or disables three-phase aggregation and join to compute distinct-qualified aggregates. This applies only to subqueries that include one or more distinct-qualified aggregate functions.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

gp_enable_direct_dispatch

Enables or disables the dispatching of targeted query plans for queries that access data on a single segment. When on, queries that target rows on a single segment will only have their query plan dispatched to that segment (rather than to all segments). This significantly reduces the response time of qualifying queries as there is no interconnect setup involved. Direct dispatch does require more CPU utilization on the master.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

gp_enable_exchange_default_partition

Controls availability of the `EXCHANGE DEFAULT PARTITION` clause for `ALTER TABLE`. The default value for the parameter is `off`. The clause is not available and Greenplum Database returns an error if the clause is specified in an `ALTER TABLE` command.

If the value is `on`, Greenplum Database returns a warning stating that exchanging the default partition might result in incorrect results due to invalid data in the default partition.

**Warning:** Before you exchange the default partition, you must ensure the data in the table to be exchanged, the new default partition, is valid for the default partition. For example, the data in the new default partition must not contain data that would be valid in other leaf child partitions of the partitioned table. Otherwise, queries against the partitioned table with the exchanged default partition that are executed by GPORCA might return incorrect results.
<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**gp_enable_fallback_plan**

Allows use of disabled plan types when a query would not be feasible without them.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**gp_enable_fast_sri**

When set to `on`, the legacy query optimizer (planner) plans single row inserts so that they are sent directly to the correct segment instance (no motion operation required). This significantly improves performance of single-row-insert statements.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**gp_enable_gpperfmon**

Enables or disables the data collection agents that populate the `gpperfmon` database for Greenplum Command Center.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**gp_enable_groupest_distinct_gather**

Enables or disables gathering data to a single node to compute distinct-qualified aggregates on grouping extension queries. When this parameter and `gp_enable_groupest_distinct_pruning` are both enabled, the legacy query optimizer (planner) uses the cheaper plan.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>
**gp_enable_groupext_distinct_pruning**

Enables or disables three-phase aggregation and join to compute distinct-qualified aggregates on grouping extension queries. Usually, enabling this parameter generates a cheaper query plan that the legacy query optimizer (planner) will use in preference to existing plan.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**gp_enable_multiphase_agg**

Enables or disables the use of two or three-stage parallel aggregation plans legacy query optimizer (planner). This approach applies to any subquery with aggregation. If gp_enable_multiphase_agg is off, then gp_enable_agg_distinct and gp_enable_agg_distinct_pruning are disabled.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**gp_enable_predicate_propagation**

When enabled, the legacy query optimizer (planner) applies query predicates to both table expressions in cases where the tables are joined on their distribution key column(s). Filtering both tables prior to doing the join (when possible) is more efficient.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**gp_enable_preunique**

Enables two-phase duplicate removal for `SELECT DISTINCT` queries (not `SELECT COUNT(DISTINCT)`). When enabled, it adds an extra `SORT DISTINCT` set of plan nodes before motioning. In cases where the distinct operation greatly reduces the number of rows, this extra `SORT DISTINCT` is much cheaper than the cost of sending the rows across the Interconnect.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>
**gp_enable_relsize_collection**

Enables the legacy query optimizer (planner) to use the estimated size of a table (pg_relation_size function) if there are no statistics for the table. By default, the planner uses a default value to estimate the number of rows if statistics are not available. The default behavior improves query optimization time and reduces resource queue usage in heavy workloads, but can lead to suboptimal plans.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**gp_enable_sequential_window_plans**

If on, enables non-parallel (sequential) query plans for queries containing window function calls. If off, evaluates compatible window functions in parallel and rejoins the results. This is an experimental parameter.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**gp_enable_segment_copy_checking**

Controls whether the distribution policy for a table (from the table DISTRIBUTED clause) is checked when data is copied into the table with the COPY FROM...ON SEGMENT command. The default is true, check the policy when copying data into the table. An error is returned in these situations.

- For a table that is not a partitioned table, an error is returned if the row of data violates the distribution policy for a segment instance.
- For a partitioned table, an error is returned if either of the following is true.
  - If the distribution policy of the child leaf partitioned table is the same as the root table and the row of data violates the distribution policy for a segment instance.
  - If the distribution policy of the child leaf partitioned table is not the same as the root table.

If the value is false, the distribution policy is not checked. The data added to the table might violate the table distribution policy for the segment instance. Manual redistribution of table data might be required. See the ALTER TABLE clause WITH REORGANIZE.

The parameter can be set for a database system or a session. The parameter cannot be set for a specific database.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>true</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>
**gp_enable_sort_distinct**

Enable duplicates to be removed while sorting.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**gp_enable_sort_limit**

Enable LIMIT operation to be performed while sorting. Sorts more efficiently when the plan requires the first limit_number of rows at most.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**gp_external_enable_exec**

Enables or disables the use of external tables that execute OS commands or scripts on the segment hosts (CREATE EXTERNAL TABLE EXECUTE syntax). Must be enabled if using the Command Center or MapReduce features.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**gp_external_max_segs**

Sets the number of segments that will scan external table data during an external table operation, the purpose being not to overload the system with scanning data and take away resources from other concurrent operations. This only applies to external tables that use the gpfdist:// protocol to access external table data.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>64</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**gp_filerep_tcp_keepalives_count**

How many keepalives may be lost before the connection is considered dead. A value of 0 uses the system default. If TCP_KEEPCNT is not supported, this parameter must be 0.
Use this parameter for all connections that are between a primary and mirror segment. Use tcp_keepalives_count for settings that are not between a primary and mirror segment.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of lost keepalives</td>
<td>2</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>

**gp_filerep_tcp_keepalives_idle**

Number of seconds between sending keepalives on an otherwise idle connection. A value of 0 uses the system default. If TCP_KEEPIDLE is not supported, this parameter must be 0.

Use this parameter for all connections that are between a primary and mirror segment. Use tcp_keepalives_idle for settings that are not between a primary and mirror segment.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of seconds</td>
<td>1 min</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>

**gp_filerep_tcp_keepalives_interval**

How many seconds to wait for a response to a keepalive before retransmitting. A value of 0 uses the system default. If TCP_KEEPINTVL is not supported, this parameter must be 0.

Use this parameter for all connections that are between a primary and mirror segment. Use tcp_keepalives_interval for settings that are not between a primary and mirror segment.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of seconds</td>
<td>30 sec</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>

**gp_fts_probe_interval**

Specifies the polling interval for the fault detection process (ftsprobe). The ftsprobe process will take approximately this amount of time to detect a segment failure.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 - 3600 seconds</td>
<td>1min</td>
<td>master, system, restart</td>
</tr>
</tbody>
</table>

**gp_fts_probe_retries**

Specifies the number of times the fault detection process (ftsprobe) attempts to connect to a segment before reporting segment failure.
### gp_fts_probe_threadcount

Specifies the number of `ftsprobe` threads to create. This parameter should be set to a value equal to or greater than the number of segments per host.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>5</td>
<td>master, system, restart</td>
</tr>
</tbody>
</table>

### gp_fts_probe_timeout

Specifies the allowed timeout for the fault detection process (`ftsprobe`) to establish a connection to a segment before declaring it down.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 - 3600 seconds</td>
<td>20 secs</td>
<td>master, system, restart</td>
</tr>
</tbody>
</table>

### gp_log_fts

Controls the amount of detail the fault detection process (`ftsprobe`) writes to the log file.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>TERSE</td>
<td>master, system, restart</td>
</tr>
</tbody>
</table>

### gp_log_interconnect

Controls the amount of information that is written to the log file about communication between Greenplum Database segment instance worker processes. The default value is `terse`. The log information is written to both the master and segment instance logs.

Increasing the amount of logging could affect performance and increase disk space usage.
<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>terse</td>
<td>master</td>
</tr>
<tr>
<td>terse</td>
<td>terse</td>
<td>session</td>
</tr>
<tr>
<td>verbose</td>
<td>terse</td>
<td>reload</td>
</tr>
<tr>
<td>debug</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**gp_log_gang**

Controls the amount of information that is written to the log file about query worker process creation and query management. The default value is **OFF**, do not log information.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>OFF</td>
<td>master</td>
</tr>
<tr>
<td>TERSE</td>
<td></td>
<td>session</td>
</tr>
<tr>
<td>VERBOSE</td>
<td></td>
<td>restart</td>
</tr>
<tr>
<td>DEBUG</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**gp_gpperfmon_send_interval**

Sets the frequency that the Greenplum Database server processes send query execution updates to the data collection agent processes used to populate the `gpperfmon` database for Command Center. Query operations executed during this interval are sent through UDP to the segment monitor agents. If you find that an excessive number of UDP packets are dropped during long-running, complex queries, you may consider increasing this value.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any valid time expression (number and unit)</td>
<td>1sec</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**gpperfmon_log_alert_level**

Controls which message levels are written to the gpperfmon log. Each level includes all the levels that follow it. The later the level, the fewer messages are sent to the log. **Note:** If the `gpperfmon` database is installed and is monitoring the database, the default value is warning.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>none</td>
<td>local</td>
</tr>
<tr>
<td>warning</td>
<td></td>
<td>system</td>
</tr>
<tr>
<td>error</td>
<td></td>
<td>restart</td>
</tr>
<tr>
<td>fatal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>panic</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**gp_hadoop_home**

When using Pivotal HD, specify the installation directory for Hadoop. For example, the default installation directory is `/usr/lib/gphd`

When using Greenplum HD 1.2 or earlier, specify the same value as the `HADOOP_HOME` environment variable.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid directory name</td>
<td>Value of <code>HADOOP_HOME</code></td>
<td>local, session, reload</td>
</tr>
</tbody>
</table>

**gp_hadoop_target_version**

The installed version of Greenplum Hadoop target.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>gphd-1.0</td>
<td>gphd-1.1</td>
<td>local, session, reload</td>
</tr>
<tr>
<td>gphd-1.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gphd-1.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gphd-2.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gphd-3.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gpmr-1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gpmr-1.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hadoop2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hdp2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cdh5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cdh3u2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cdh4.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**gp_hashjoin_tuples_per_bucket**

Sets the target density of the hash table used by HashJoin operations. A smaller value will tend to produce larger hash tables, which can increase join performance.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>5</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

**gp_idf_deduplicate**

Changes the strategy to compute and process MEDIAN, and PERCENTILE_DISC.
### gp_initial_bad_row_limit

For the parameter value $n$, Greenplum Database stops processing input rows when you import data with the `COPY` command or from an external table if the first $n$ rows processed contain formatting errors. If a valid row is processed within the first $n$ rows, Greenplum Database continues processing input rows.

Setting the value to 0 disables this limit.

The `SEGMENT REJECT LIMIT` clause can also be specified for the `COPY` command or the external table definition to limit the number of rejected rows.

**INT_MAX** is the largest value that can be stored as an integer on your system.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer $0$ - INT_MAX</td>
<td>1000</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

### gp_interconnect_debug_retry_interval

Specifies the interval, in seconds, to log Greenplum Database interconnect debugging messages when the server configuration parameter `gp_log_interconnect` is set to `DEBUG`. The default is 10 seconds.

The log messages contain information about the interconnect communication between Greenplum Database segment instance worker processes. The information can be helpful when debugging network issues between segment instances.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 &lt;=&lt; Integer &lt; 4096</td>
<td>10</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

### gp_interconnect_fc_method

Specifies the flow control method used for the default Greenplum Database UDPIFC interconnect.

For capacity based flow control, senders do not send packets when receivers do not have the capacity.

Loss based flow control is based on capacity based flow control, and also tunes the sending speed according to packet losses.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAPACITY</td>
<td>LOSS</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOSS</td>
<td></td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>
### gp_interconnect_hash_multiplier

Sets the size of the hash table used by the Greenplum Database to track interconnect connections with the default UDPIFC interconnect. This number is multiplied by the number of segments to determine the number of buckets in the hash table. Increasing the value may increase interconnect performance for complex multi-slice queries (while consuming slightly more memory on the segment hosts).

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-25</td>
<td>2</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

### gp_interconnect_queue_depth

Sets the amount of data per-peer to be queued by the Greenplum Database interconnect on receivers (when data is received but no space is available to receive it the data will be dropped, and the transmitter will need to resend it) for the default UDPIFC interconnect. Increasing the depth from its default value will cause the system to use more memory, but may increase performance. It is reasonable to set this value between 1 and 10. Queries with data skew potentially perform better with an increased queue depth. Increasing this may radically increase the amount of memory used by the system.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2048</td>
<td>4</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

### gp_interconnect_setup_timeout

Specifies the amount of time to wait for the Greenplum Database interconnect to complete setup before it times out.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any valid time expression (number and unit)</td>
<td>2 hours</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

### gp_interconnect_snd_queue_depth

Sets the amount of data per-peer to be queued by the default UDPIFC interconnect on senders. Increasing the depth from its default value will cause the system to use more memory, but may increase performance. Reasonable values for this parameter are between 1 and 4. Increasing the value might radically increase the amount of memory used by the system.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 4096</td>
<td>2</td>
<td>master session reload</td>
</tr>
</tbody>
</table>
**gp_interconnect_type**

Sets the networking protocol used for Greenplum Database interconnect traffic. UDPIFC specifies using UDP with flow control for interconnect traffic, and is the only value supported.

UDPIFC (the default) specifies using UDP with flow control for interconnect traffic. Specify the interconnect flow control method with `gp_interconnect_fc_method`.

With TCP as the interconnect protocol, Greenplum Database has an upper limit of 1000 segment instances - less than that if the query workload involves complex, multi-slice queries.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDPIFC</td>
<td>UDPIFC</td>
<td>local</td>
</tr>
<tr>
<td>TCP</td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**gp_log_format**

Specifies the format of the server log files. If using `gp_toolkit` administrative schema, the log files must be in CSV format.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>csv</td>
<td>csv</td>
<td>local</td>
</tr>
<tr>
<td>text</td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**gp_max Csv_line_length**

The maximum length of a line in a CSV formatted file that will be imported into the system. The default is 1MB (1048576 bytes). Maximum allowed is 4MB (4194184 bytes). The default may need to be increased if using the `gp_toolkit` administrative schema to read Greenplum Database log files.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of bytes</td>
<td>1048576</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**gp_max_databases**

The maximum number of databases allowed in a Greenplum Database system.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>16</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**gp_max_filespaces**

The maximum number of filespaces allowed in a Greenplum Database system.
<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>8</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**gp_max_local_distributed_cache**

Sets the maximum number of distributed transaction log entries to cache in the backend process memory of a segment instance.

The log entries contain information about the state of rows that are being accessed by an SQL statement. The information is used to determine which rows are visible to an SQL transaction when executing multiple simultaneous SQL statements in an MVCC environment. Caching distributed transaction log entries locally improves transaction processing speed by improving performance of the row visibility determination process.

The default value is optimal for a wide variety of SQL processing environments.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>1024</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**gp_max_packet_size**

Sets the tuple-serialization chunk size for the Greenplum Database interconnect.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>512-65536</td>
<td>8192</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**gp_max_plan_size**

Specifies the total maximum uncompressed size of a query execution plan multiplied by the number of Motion operators (slices) in the plan. If the size of the query plan exceeds the value, the query is cancelled and an error is returned. A value of 0 means that the size of the plan is not monitored.

You can specify a value in \( \text{kB} \), \( \text{MB} \), or \( \text{GB} \). The default unit is \( \text{kB} \). For example, a value of 200 is 200kB. A value of 1GB is the same as 1024MB or 1048576kB.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>0</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>superuser</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
</tbody>
</table>

**gp_max_tablespaces**

The maximum number of tablespaces allowed in a Greenplum Database system.
### gp_motion_cost_per_row

Sets the legacy query optimizer (planner) cost estimate for a Motion operator to transfer a row from one segment to another, measured as a fraction of the cost of a sequential page fetch. If 0, then the value used is two times the value of `cpu_tuple_cost`.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>16</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

### gp_num_contents_in_cluster

The number of primary segments in the Greenplum Database system.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>read only</td>
</tr>
</tbody>
</table>

### gp_recursive_cte_prototype

Controls the availability of the experimental keyword **RECURSIVE** in the **WITH** clause of a **SELECT** [ **INTO** ] command. The keyword allows a subquery in the **WITH** clause of a **SELECT** [ **INTO** ] command to reference itself. The default value is **false**, the keyword is not allowed in the **WITH** clause a **SELECT** [ **INTO** ] command.

**Warning:** The **RECURSIVE** keyword is an experimental feature and is not intended for use in a production environment. Experimental features are subject to change without notice in future releases.

For information about the experimental keyword **RECURSIVE**, see the **SELECT** command.

The parameter can be set for a database system, an individual database, or a session or query.

**Note:** This parameter will be removed if the **RECURSIVE** keyword is promoted from experimental or if the keyword is removed from Greenplum Database.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>false</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

### gp_reject_percent_threshold

For single row error handling on COPY and external table SELECTs, sets the number of rows processed before SEGMENT REJECT LIMIT **n PERCENT** starts calculating.
<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-n</td>
<td>300</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

**gp_reraise_signal**

If enabled, will attempt to dump core if a fatal server error occurs.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

**gp_resgroup_memory_policy**

Warning: Resource groups are an experimental feature and are not intended for use in a production environment. Experimental features are subject to change without notice in future releases.

Used by a resource group to manage memory allocation to query operators.

When set to `auto`, Greenplum Database uses resource group memory limits to distribute memory across query operators, allocating a fixed size of memory to non-memory-intensive operators and the rest to memory-intensive operators.

When you specify `eager_free`, Greenplum Database distributes memory among operators more optimally by re-allocating memory released by operators that have completed their processing to operators in a later query stage.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto, eager_free</td>
<td>eager_free</td>
<td>local, system, superuser, restart/reload</td>
</tr>
</tbody>
</table>

**gp_resource_group_cpu_limit**

Warning: Resource groups are an experimental feature and are not intended for use in a production environment. Experimental features are subject to change without notice in future releases.

Identifies the maximum percentage of system CPU resources to allocate to resource groups on each Greenplum Database segment node.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1 - 1.0</td>
<td>0.9</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>
gp_resource_group_memory_limit

Warning: Resource groups are an experimental feature and are not intended for use in a production environment. Experimental features are subject to change without notice in future releases.

Identifies the maximum percentage of system memory resources to allocate to resource groups on each Greenplum Database segment node.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1 - 1.0</td>
<td>0.9</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>

gp_resource_manager

Warning: Resource groups are an experimental feature and are not intended for use in a production environment. Experimental features are subject to change without notice in future releases.

Identifies the resource management scheme currently enabled in the Greenplum Database cluster. The default scheme is to use resource queues.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>queue</td>
<td>local, system, restart</td>
</tr>
<tr>
<td>queue</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

gp_resqueue_memory_policy

Enables Greenplum memory management features. The distribution algorithm eager_free takes advantage of the fact that not all operators execute at the same time (in Greenplum Database 4.2 and later). The query plan is divided into stages and Greenplum Database eagerly frees memory allocated to a previous stage at the end of that stage's execution, then allocates the eagerly freed memory to the new stage.

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.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>none, auto, eager_free</td>
<td>eager_free</td>
<td>local, system, restart/reload</td>
</tr>
</tbody>
</table>

gp_resqueue_priority

Enables or disables query prioritization. When this parameter is disabled, existing priority settings are not evaluated at query run time.
gp_resqueue_priority_cpucores_per_segment

Specifies the number of CPU units allocated per segment instance. For example, if a Greenplum Database cluster has 10-core segment hosts that are configured with four segments, set the value for the segment instances to 2.5. For the master instance, the value would be 10. A master host typically has only the master instance running on it, so the value for the master should reflect the usage of all available CPU cores.

Incorrect settings can result in CPU under-utilization or query prioritization not working as designed.

The default values for the Greenplum Data Computing Appliance V2 are 4 for segments and 25 for the master.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1 - 512.0</td>
<td>4</td>
<td>local system restart</td>
</tr>
</tbody>
</table>

gp_resqueue_priority_sweeper_interval

Specifies the interval at which the sweeper process evaluates current CPU usage. When a new statement becomes active, its priority is evaluated and its CPU share determined when the next interval is reached.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>500 - 15000 ms</td>
<td>1000</td>
<td>local system restart</td>
</tr>
</tbody>
</table>

gp_role

The role of this server process is set to dispatch for the master and execute for a segment.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>dispatch</td>
<td></td>
<td>read only</td>
</tr>
<tr>
<td>execute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>utility</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

gp_safefswritesize

Specifies a minimum size for safe write operations to append-optimized tables in a non-mature file system. When a number of bytes greater than zero is specified, the append-optimized writer adds padding data up to that number in order to prevent data corruption due to file system errors. Each non-mature file system has a known safe write size that must be specified here when using Greenplum Database with that type of
file system. This is commonly set to a multiple of the extent size of the file system; for example, Linux ext3 is 4096 bytes, so a value of 32768 is commonly used.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>0</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>

**gp_segment_connect_timeout**

Time that the Greenplum interconnect will try to connect to a segment instance over the network before timing out. Controls the network connection timeout between master and primary segments, and primary to mirror segment replication processes.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any valid time expression (number and unit)</td>
<td>10min</td>
<td>local, system, reload</td>
</tr>
</tbody>
</table>

**gp_segments_for_planner**

Sets the number of primary segment instances for the legacy query optimizer (planner) to assume in its cost and size estimates. If 0, then the value used is the actual number of primary segments. This variable affects the legacy optimizer's estimates of the number of rows handled by each sending and receiving process in Motion operators.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-n</td>
<td>0</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

**gp_server_version**

Reports the version number of the server as a string. A version modifier argument might be appended to the numeric portion of the version string, example: *5.0.0 beta*.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>String. Examples: 5.0.0</td>
<td>n/a</td>
<td>read only</td>
</tr>
</tbody>
</table>

**gp_server_version_num**

Reports the version number of the server as an integer. The number is guaranteed to always be increasing for each version and can be used for numeric comparisons. The major version is represented as is, the minor and patch versions are zero-padded to always be double digit wide.
### gp_session_id

A system assigned ID number for a client session. Starts counting from 1 when the master instance is first started.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-(n)</td>
<td>14</td>
<td>read only</td>
</tr>
</tbody>
</table>

### gp_set_proc_affinity

If enabled, when a Greenplum server process (postmaster) is started it will bind to a CPU.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

### gp_set_read_only

Set to on to disable writes to the database. Any in progress transactions must finish before read-only mode takes affect.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

### gp_snmp_community

Set to the community name you specified for your environment.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMP community name</td>
<td>public</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

### gp_snmp_monitor_address

The hostname:port of your network monitor application. Typically, the port number is 162. If there are multiple monitor addresses, separate them with a comma.
### gp_snmp_use_inform_or_trap

Trap notifications are SNMP messages sent from one application to another (for example, between Greenplum Database and a network monitoring application). These messages are unacknowledged by the monitoring application, but generate less network overhead.

Inform notifications are the same as trap messages, except that the application sends an acknowledgement to the application that generated the alert.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>inform</td>
<td>trap</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

### gp_statistics_pullup_from_child_partition

Enables the use of statistics from child tables when planning queries on the parent table by the legacy query optimizer (planner).

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

### gp_statistics_use_fkeys

When enabled, allows the legacy query optimizer (planner) to use foreign key information stored in the system catalog to optimize joins between foreign keys and primary keys.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

### gp_vmem_idle_resource_timeout

If a database session is idle for longer than the time specified, the session will free system resources (such as shared memory), but remain connected to the database. This allows more concurrent connections to the database at one time.
**gp_vmem_protect_limit**

Sets the amount of memory (in number of MBs) that all postgres processes of an active segment instance can consume. If a query causes this limit to be exceeded, memory will not be allocated and the query will fail. Note that this is a local parameter and must be set for every segment in the system (primary and mirrors). When setting the parameter value, specify only the numeric value. For example, to specify 4096MB, use the value 4096. Do not add the units MB to the value.

To prevent over-allocation of memory, these calculations can estimate a safe `gp_vmem_protect_limit` value.

First calculate the value `gp_vmem`. This is the Greenplum Database memory available on a host

\[
gp_vmem = \frac{(\text{SWAP} + \text{RAM}) - (7.5GB + 0.05 \times \text{RAM})}{1.7}
\]

where `SWAP` is the host swap space and `RAM` is the RAM on the host in GB.

Next, calculate the `max_acting_primary_segments`. This is the maximum number of primary segments that can be running on a host when mirror segments are activated due to a failure. With mirrors arranged in a 4-host block with 8 primary segments per host, for example, a single segment host failure would activate two or three mirror segments on each remaining host in the failed host's block. The `max_acting_primary_segments` value for this configuration is 11 (8 primary segments plus 3 mirrors activated on failure).

This is the calculation for `gp_vmem_protect_limit`. The value should be converted to MB.

\[
gp_vmem_protect_limit = \frac{gp_vmem}{\text{acting_primary_segments}}
\]

For scenarios where a large number of workfiles are generated, this is the calculation for `gp_vmem` that accounts for the workfiles.

\[
gp_vmem = \frac{(\text{SWAP} + \text{RAM}) - (7.5GB + 0.05 \times \text{RAM} - (300KB \times \text{total\#\_workfiles}))}{1.7}
\]

For information about monitoring and managing workfile usage, see the Greenplum Database Administrator Guide.

Based on the `gp_vmem` value you can calculate the value for the `vm.overcommit_ratio` operating system kernel parameter. This parameter is set when you configure each Greenplum Database host.

\[
vm.overcommit_ratio = \frac{(\text{RAM} - (0.026 \times gp_vmem))}{\text{RAM}}
\]

**Note:** The default value for the kernel parameter `vm.overcommit_ratio` in Red Hat Enterprise Linux is 50.

For information about the kernel parameter, see the Greenplum Database Installation Guide.
<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>8192</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**gp_vmem_protect_segworker_cache_limit**

If a query executor process consumes more than this configured amount, then the process will not be cached for use in subsequent queries after the process completes. Systems with lots of connections or idle processes may want to reduce this number to free more memory on the segments. Note that this is a local parameter and must be set for every segment.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of megabytes</td>
<td>500</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**gp_workfile_checksumming**

Adds a checksum value to each block of a work file (or spill file) used by HashAgg and HashJoin query operators. This adds an additional safeguard from faulty OS disk drivers writing corrupted blocks to disk. When a checksum operation fails, the query will cancel and rollback rather than potentially writing bad data to disk.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**gp_workfile_compress_algorithm**

When a hash aggregation or hash join operation spills to disk during query processing, specifies the compression algorithm to use on the spill files. If using zlib, the zlib library must be installed on all hosts in the Greenplum Database cluster.

If your Greenplum Database installation uses serial ATA (SATA) disk drives, setting the value of this parameter to zlib might help to avoid overloading the disk subsystem with IO operations.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>none</td>
<td>master</td>
</tr>
<tr>
<td>zlib</td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**gp_workfile_limit_files_per_query**

Sets the maximum number of temporary spill files (also known as workfiles) allowed per query per segment. Spill files are created when executing a query that requires more memory than it is allocated. The current query is terminated when the limit is exceeded.
Set the value to 0 (zero) to allow an unlimited number of spill files.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>100000</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

### gp_workfile_limit_per_query

Sets the maximum disk size an individual query is allowed to use for creating temporary spill files at each segment. The default value is 0, which means a limit is not enforced.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>kilobytes</td>
<td>0</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

### gp_workfile_limit_per_segment

Sets the maximum total disk size that all running queries are allowed to use for creating temporary spill files at each segment. The default value is 0, which means a limit is not enforced.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>kilobytes</td>
<td>0</td>
<td>local system restart</td>
</tr>
</tbody>
</table>

### gpperfmon_port

Sets the port on which all data collection agents (for Command Center) communicate with the master.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>8888</td>
<td>master system restart</td>
</tr>
</tbody>
</table>

### ignore_checksum_failure

Only has effect if `data_checkums` is enabled.

Greenplum Database uses checksums to prevent loading data that has been corrupted in the file system into memory managed by database processes.

By default, when a checksum verify error occurs when reading a heap data page, Greenplum Database generates an error and prevents the page from being loaded into managed memory. When `ignore_checksum_failure` is set to on and a checksum verify failure occurs, Greenplum Database generates a warning, and allows the page to be read into managed memory. If the page is then updated it is saved to disk and replicated to the mirror. If the page header is corrupt an error is reported even if this option is enabled.
**Warning:** Setting `ignore_checksum_failure` to on may propagate or hide data corruption or lead to other serious problems. However, if a checksum failure has already been detected and the page header is uncorrupted, setting `ignore_checksum_failure` to on may allow you to bypass the error and recover undamaged tuples that may still be present in the table.

The default setting is off, and it can only be changed by a superuser.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**integer_datetimes**

Reports whether PostgreSQL was built with support for 64-bit-integer dates and times.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>read only</td>
</tr>
</tbody>
</table>

**IntervalStyle**

Sets the display format for interval values. The value `sql_standard` produces output matching SQL standard interval literals. The value `postgres` produces output matching PostgreSQL releases prior to 8.4 when the `DateStyle` parameter was set to ISO.

The value `postgres_verbose` produces output matching Greenplum releases prior to 3.3 when the `DateStyle` parameter was set to non-ISO output.

The value `iso_8601` will produce output matching the time interval `format with designators` defined in section 4.4.3.2 of ISO 8601. See the PostgreSQL 8.4 documentation for more information.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>postgres</td>
<td>postgres</td>
<td></td>
</tr>
<tr>
<td>postgres_verbose</td>
<td></td>
<td>master</td>
</tr>
<tr>
<td>sql_standard</td>
<td></td>
<td>session</td>
</tr>
<tr>
<td>iso_8601</td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**joinCollapse_limit**

The legacy query optimizer (planner) will rewrite explicit inner JOIN constructs into lists of FROM items whenever a list of no more than this many items in total would result. By default, this variable is set the same as `fromCollapse_limit`, which is appropriate for most uses. Setting it to 1 prevents any reordering of inner JOINs. Setting this variable to a value between 1 and `fromCollapse_limit` might be useful to trade off planning time against the quality of the chosen plan (higher values produce better plans).

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-n</td>
<td>20</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>
**keep_wal_segments**

For Greenplum Database master mirroring, sets the maximum number of processed WAL segment files that are saved by the active Greenplum Database master if a checkpoint operation occurs.

The segment files are used to synchronize the active master on the standby master.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>5</td>
<td>master, system, reload, superuser</td>
</tr>
</tbody>
</table>

**krb_caseins_users**

Sets whether Kerberos user names should be treated case-insensitively. The default is case sensitive (off).

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master, system, restart</td>
</tr>
</tbody>
</table>

**krb_server_keyfile**

Sets the location of the Kerberos server key file.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>path and file name</td>
<td>unset</td>
<td>master, system, restart</td>
</tr>
</tbody>
</table>

**krb_srvname**

Sets the Kerberos service name.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>service name</td>
<td>postgres</td>
<td>master, system, restart</td>
</tr>
</tbody>
</table>

**lc_collate**

Reports the locale in which sorting of textual data is done. The value is determined when the Greenplum Database array is initialized.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system dependent&gt;</td>
<td></td>
<td>read only</td>
</tr>
</tbody>
</table>
lc_ctype
Reports the locale that determines character classifications. The value is determined when the Greenplum Database array is initialized.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system dependent&gt;</td>
<td></td>
<td>read only</td>
</tr>
</tbody>
</table>

lc_messages
Sets the language in which messages are displayed. The locales available depends on what was installed with your operating system - use `locale -a` to list available locales. The default value is inherited from the execution environment of the server. On some systems, this locale category does not exist. Setting this variable will still work, but there will be no effect. Also, there is a chance that no translated messages for the desired language exist. In that case you will continue to see the English messages.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system dependent&gt;</td>
<td></td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

lc_monetary
Sets the locale to use for formatting monetary amounts, for example with the `to_char` family of functions. The locales available depends on what was installed with your operating system - use `locale -a` to list available locales. The default value is inherited from the execution environment of the server.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system dependent&gt;</td>
<td></td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

lc_numeric
Sets the locale to use for formatting numbers, for example with the `to_char` family of functions. The locales available depends on what was installed with your operating system - use `locale -a` to list available locales. The default value is inherited from the execution environment of the server.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system dependent&gt;</td>
<td></td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

lc_time
This parameter currently does nothing, but may in the future.
### Value Range

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system dependent&gt;</td>
<td></td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

### listen_addresses

Specifies the TCP/IP address(es) on which the server is to listen for connections from client applications - a comma-separated list of host names and/or numeric IP addresses. The special entry * corresponds to all available IP interfaces. If the list is empty, only UNIX-domain sockets can connect.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>localhost, host names, IP addresses, * (all available IP interfaces)</td>
<td>*</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

### local_preload_libraries

Comma separated list of shared library files to preload at the start of a client session.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

### log_autostats

Logs information about automatic ANALYZE operations related to `gp_autostats_mode` and `gp_autostats_on_change_threshold`.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>superuser</td>
</tr>
</tbody>
</table>

### log_connections

This outputs a line to the server log detailing each successful connection. Some client programs, like psql, attempt to connect twice while determining if a password is required, so duplicate "connection received" messages do not always indicate a problem.
### log_disconnections

This outputs a line in the server log at termination of a client session, and includes the duration of the session.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>

### log_dispatch_stats

When set to "on," this parameter adds a log message with verbose information about the dispatch of the statement.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>

### log_duration

Causes the duration of every completed statement which satisfies `log_statement` to be logged.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master, session, reload, superuser</td>
</tr>
</tbody>
</table>

### log_error_verbosity

Controls the amount of detail written in the server log for each message that is logged.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>TERSE</td>
<td>DEFAULT</td>
<td>master, session, reload, superuser</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>DEFAULT</td>
<td>master, session, reload, superuser</td>
</tr>
<tr>
<td>VERBOSE</td>
<td></td>
<td>master, session, reload, superuser</td>
</tr>
</tbody>
</table>
**log_executor_stats**

For each query, write performance statistics of the query executor to the server log. This is a crude profiling instrument. Cannot be enabled together with `log_statement_stats`.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>

**log_hostname**

By default, connection log messages only show the IP address of the connecting host. Turning on this option causes logging of the IP address and host name of the Greenplum Database master. Note that depending on your host name resolution setup this might impose a non-negligible performance penalty.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master, system, restart</td>
</tr>
</tbody>
</table>

**log_min_duration_statement**

Logs the statement and its duration on a single log line if its duration is greater than or equal to the specified number of milliseconds. Setting this to 0 will print all statements and their durations. -1 disables the feature. For example, if you set it to 250 then all SQL statements that run 250ms or longer will be logged. Enabling this option can be useful in tracking down unoptimized queries in your applications.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of milliseconds, 0, -1</td>
<td>-1</td>
<td>master, session, reload, superuser</td>
</tr>
</tbody>
</table>

**log_min_error_statement**

Controls whether or not the SQL statement that causes an error condition will also be recorded in the server log. All SQL statements that cause an error of the specified level or higher are logged. The default is PANIC (effectively turning this feature off for normal use). Enabling this option can be helpful in tracking down the source of any errors that appear in the server log.
### log_min_messages

Controls which message levels are written to the server log. Each level includes all the levels that follow it. The later the level, the fewer messages are sent to the log.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG5</td>
<td>ERROR</td>
<td>master, session, reload, superuser</td>
</tr>
<tr>
<td>DEBUG4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEBUG3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEBUG2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEBUG1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INFO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOTICE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WARNING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERROR</td>
<td>WARNING</td>
<td>master, session, reload, superuser</td>
</tr>
<tr>
<td>FATAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PANIC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### log_parser_stats

For each query, write performance statistics of the query parser to the server log. This is a crude profiling instrument. Cannot be enabled together with `log_statement_stats`.
<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>superuser</td>
</tr>
</tbody>
</table>

**log_planner_stats**
For each query, write performance statistics of the legacy query optimizer (planner) to the server log. This is a crude profiling instrument. Cannot be enabled together with log_statement_stats.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>superuser</td>
</tr>
</tbody>
</table>

**log_rotation_age**
Determines the maximum lifetime of an individual log file. After this time has elapsed, a new log file will be created. Set to zero to disable time-based creation of new log files.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any valid time expression (number and unit)</td>
<td>1d</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**log_rotation_size**
Determines the maximum size of an individual log file. After this many kilobytes have been emitted into a log file, a new log file will be created. Set to zero to disable size-based creation of new log files.

The maximum value is INT_MAX/1024. If an invalid value is specified, the default value is used. INT_MAX is the largest value that can be stored as an integer on your system.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of kilobytes</td>
<td>0</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**log_statement**
Controls which SQL statements are logged. DDL logs all data definition commands like CREATE, ALTER, and DROP commands. MOD logs all DDL statements, plus INSERT, UPDATE, DELETE, TRUNCATE, and COPY FROM. PREPARE and EXPLAIN ANALYZE statements are also logged if their contained command is of an appropriate type.
### log_statement_stats

For each query, write total performance statistics of the query parser, planner, and executor to the server log. This is a crude profiling instrument.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>superuser</td>
</tr>
</tbody>
</table>

### log_temp_files

Controls logging of temporary file names and sizes. Temporary files can be created for sorts, hashes, temporary query results and spill files. A log entry is made in `pg_log` for each temporary file when it is deleted. Depending on the source of the temporary files, the log entry could be created on either the master and/or segments. A `log_temp_files` value of zero logs all temporary file information, while positive values log only files whose size is greater than or equal to the specified number of kilobytes. The default setting is `-1`, which disables logging. Only superusers can change this setting.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>-1</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

### log_timezone

Sets the time zone used for timestamps written in the log. Unlike `TimeZone`, this value is system-wide, so that all sessions will report timestamps consistently. The default is `unknown`, which means to use whatever the system environment specifies as the time zone.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>unknown</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

### log_truncate_on_rotation

Truncates (overwrites), rather than appends to, any existing log file of the same name. Truncation will occur only when a new file is being opened due to time-based rotation. For example, using this setting in
combination with a log_filename such as gpseg#-%H.log would result in generating twenty-four hourly log files and then cyclically overwriting them. When off, pre-existing files will be appended to in all cases.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>local system restart</td>
</tr>
</tbody>
</table>

**max_appendonly_tables**

Sets the maximum number of concurrent transactions that can write to or update append-optimized tables. Operations that are counted are INSERT, UPDATE, COPY, and VACUUM operations. The limit is only for in-progress transactions. Once a transaction ends (either aborted or committed), it is no longer counted against this limit.

For operations against a partitioned table, each subpartition (child table) that is an append-optimized table and is changed counts as a single table towards the maximum. For example, a partitioned table p_tbl is defined with three subpartitions that are append-optimized tables p_tbl_ao1, p_tbl_ao2, and p_tbl_ao3. An INSERT or UPDATE command against the partitioned table p_tbl that changes append-optimized tables p_tbl_ao1 and p_tbl_ao2 is counted as two transactions.

Increasing the limit allocates more shared memory on the master host at server start.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer &gt; 0</td>
<td>10000</td>
<td>master system restart</td>
</tr>
</tbody>
</table>

**max_connections**

The maximum number of concurrent connections to the database server. In a Greenplum Database system, user client connections go through the Greenplum master instance only. Segment instances should allow 5-10 times the amount as the master. When you increase this parameter, max_prepared_transactions must be increased as well. For more information about limiting concurrent connections, see "Configuring Client Authentication" in the Greenplum Database Administrator Guide.

Increasing this parameter may cause Greenplum Database to request more shared memory.Increasing this parameter might cause Greenplum Database to request more shared memory. See shared_buffers for information about Greenplum server instance shared memory buffers.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-n</td>
<td>250 on master 750 on segments</td>
<td>local system restart</td>
</tr>
</tbody>
</table>

**max_files_per_process**

Sets the maximum number of simultaneously open files allowed to each server subprocess. If the kernel is enforcing a safe per-process limit, you don't need to worry about this setting. Some platforms such as BSD, the kernel will allow individual processes to open many more files than the system can really support.
### max_fsm_pages
Sets the maximum number of disk pages for which free space will be tracked in the shared free-space map. Six bytes of shared memory are consumed for each page slot.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>1000</td>
<td>local system restart</td>
</tr>
</tbody>
</table>

### max_fsm_relations
Sets the maximum number of relations for which free space will be tracked in the shared memory free-space map. Should be set to a value larger than the total number of:
- tables + indexes + system tables.  
It costs about 60 bytes of memory for each relation per segment instance. It is better to allow some room for overhead and set too high rather than too low.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>1000</td>
<td>local system restart</td>
</tr>
</tbody>
</table>

### max_function_args
Reports the maximum number of function arguments.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>100</td>
<td>read only</td>
</tr>
</tbody>
</table>

### max_identifier_length
Reports the maximum identifier length.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>63</td>
<td>read only</td>
</tr>
</tbody>
</table>

### max_index_keys
Reports the maximum number of index keys.
max_locks_per_transaction

The shared lock table is created with room to describe locks on max_locks_per_transaction *(max_connections + max_prepared_transactions)* objects, so no more than this many distinct objects can be locked at any one time. This is not a hard limit on the number of locks taken by any one transaction, but rather a maximum average value. You might need to raise this value if you have clients that touch many different tables in a single transaction.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>32</td>
<td>read only</td>
</tr>
</tbody>
</table>

max_prepared_transactions

Sets the maximum number of transactions that can be in the prepared state simultaneously. Greenplum uses prepared transactions internally to ensure data integrity across the segments. This value must be at least as large as the value of max_connections on the master. Segment instances should be set to the same value as the master.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>128</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

max_resource_portals_per_transaction

Sets the maximum number of simultaneously open user-declared cursors allowed per transaction. Note that an open cursor will hold an active query slot in a resource queue. Used for resource management.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>64</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

max_resource_groups

**Warning:** Resource groups are an experimental feature and are not intended for use in a production environment. Experimental features are subject to change without notice in future releases.

Sets the maximum number of resource groups that you can create in a Greenplum Database system. Resource groups are defined system-wide.
max_resource_queues

Sets the maximum number of resource queues that can be created in a Greenplum Database system. Note that resource queues are system-wide (as are roles) so they apply to all databases in the system.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - INT_MAX</td>
<td>9</td>
<td>master, system, restart</td>
</tr>
</tbody>
</table>

max_stack_depth

Specifies the maximum safe depth of the server’s execution stack. The ideal setting for this parameter is the actual stack size limit enforced by the kernel (as set by *ulimit -s* or local equivalent), less a safety margin of a megabyte or so. Setting the parameter higher than the actual kernel limit will mean that a runaway recursive function can crash an individual backend process.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of kilobytes</td>
<td>2MB</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>

max_statement_mem

Sets the maximum memory limit for a query. Helps avoid out-of-memory errors on a segment host during query processing as a result of setting *statement_mem* too high. When *gp_resqueue_memory_policy*=auto, *statement_mem* and resource queue memory limits control query memory usage. Taking into account the configuration of a single segment host, calculate this setting as follows:

\[
\text{(seghost_physical_memory)} / \text{(average_number_concurrent_queries)}
\]

When changing both *max_statement_mem* and *statement_mem*, *max_statement_mem* must be changed first, or listed first in the *postgresql.conf* file.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of kilobytes</td>
<td>2000MB</td>
<td>master, session, reload, superuser</td>
</tr>
</tbody>
</table>
memory_spill_ratio

Warning: Resource groups are an experimental feature and are not intended for use in a production environment. Experimental features are subject to change without notice in future releases.

Sets the memory usage threshold for memory-intensive operators in a transaction. When a transaction reaches this threshold, it spills to disk.

The default memory_spill_ratio is the value defined for the resource group assigned to the currently active role. You can set memory_spill_ratio at the session level to selectively set this limit on a per-query basis. For example, if you have a specific query that spills to disk and requires more memory, you may choose to set a larger memory_spill_ratio to increase the initial memory allocation.

When you set memory_spill_ratio at the session level, Greenplum Database does not perform semantic validation on the new value until you next perform a query.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 100</td>
<td>20</td>
<td>master session</td>
</tr>
</tbody>
</table>

optimizer

Enables or disables GPORCA when running SQL queries. The default is on. If you disable GPORCA, Greenplum Database uses only the legacy query optimizer.

GPORCA co-exists with the legacy query optimizer. With GPORCA enabled, Greenplum Database uses GPORCA to generate an execution plan for a query when possible. If GPORCA cannot be used, then the legacy query optimizer is used.

The optimizer parameter can be set for a database system, an individual database, or a session or query.

For information about the legacy query optimizer and GPORCA, see "Querying Data" in the Greenplum Database Administrator Guide.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master session</td>
</tr>
</tbody>
</table>

optimizer_analyze_root_partition

For a partitioned table, collects statistics for the root partition when the ANALYZE command is run on the table. GPORCA uses the root partition statistics when generating a query plan. The legacy query optimizer does not use these statistics. If you set the value of the server configuration parameter optimizer to on, set the value of this parameter to on and run the command ANALYZE or ANALYZE ROOTPARTITION on partitioned tables to ensure the proper statistics have been collected.

For information about the legacy query optimizer and GPORCA, see "Querying Data" in the Greenplum Database Administrator Guide.
optimizer_array Expansion_threshold

When GPOCA is enabled (the default) and is processing a query that contains a predicate with a constant array, the optimizer_array_expansion_threshold parameter limits the optimization process based on the number of constants in the array. If the array in the query predicate contains more than the number elements specified by parameter, GPOCA disables the transformation of the predicate into its disjunctive normal form during query optimization.

The default value is 25.

For example, when GPOCA is executing a query that contains an IN clause with more than 25 elements, GPOCA does not transform the predicate into its disjunctive normal form during query optimization to reduce optimization time consume less memory. The difference in query processing can be seen in the filter condition for the IN clause of the query EXPLAIN plan.

Changing the value of this parameter changes the trade-off between a shorter optimization time and lower memory consumption, and the potential benefits from constraint derivation during query optimization, for example conflict detection and partition elimination.

The parameter can be set for a database system, an individual database, or a session or query.

optimizer_control

Controls whether the server configuration parameter optimizer can be changed with SET, the RESET command, or the Greenplum Database utility gpconfig. If the optimizer_control parameter value is on, users can set the optimizer parameter. If the optimizer_control parameter value is off, the optimizer parameter cannot be changed.

optimizer_cte_Inlining_bound

When GPOCA is enabled (the default), this parameter controls the amount of inlining performed for common table expression (CTE) queries (queries that contain a WHERE clause). The default value, 0, disables inlining.

The parameter can be set for a database system, an individual database, or a session or query.
### optimizer_enable_master_only_queries

When GPORCA is enabled (the default), this parameter allows GPORCA to execute catalog queries that run only on the Greenplum Database master. For the default value `off`, only the legacy query optimizer can execute catalog queries that run only on the Greenplum Database master.

The parameter can be set for a database system, an individual database, or a session or query.

**Note:** Enabling this parameter decreases performance of short running catalog queries. To avoid this issue, set this parameter only for a session or a query.

For information about GPORCA, see the *Greenplum Database Administrator Guide*.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal <code>&gt;= 0</code></td>
<td>0</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

### optimizer_force_multistage_agg

For the default settings, GPORCA is enabled and this parameter is `true`, GPORCA chooses a 3 stage aggregate plan for scalar distinct qualified aggregate when such a plan alternative is generated. When the value is `false`, GPORCA makes a cost based choice rather than a heuristic choice.

The parameter can be set for a database system, an individual database, or a session or query.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

### optimizer_force_three_stage_scalar_dqa

For the default settings, GPORCA is enabled and this parameter is `true`, GPORCA chooses a plan with multistage aggregates when such a plan alternative is generated. When the value is `false`, GPORCA makes a cost based choice rather than a heuristic choice.

The parameter can be set for a database system, an individual database, or a session, or query.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>true</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>
optimizer_join arity_for associativity commutativity

The value is an optimization hint to limit the number of join associativity and join commutativity transformations explored during query optimization. The limit controls the alternative plans that GPORCA considers during query optimization. For example, the default value of 7 is a optimization hint for GPORCA to stop exploring join associativity and join commutativity transformations when an n-ary join operator has more than 7 children during optimization.

For a query with a large number of joins, specifying a lower value improves query performance by limiting the number of alternate query plans that GPORCA evaluates. However, setting the value too low might cause GPORCA to generate a query plan that performs sub-optimally.

This parameter can be set for a database system or a session.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer &gt; 0</td>
<td>7</td>
<td>local system reload</td>
</tr>
</tbody>
</table>

optimizer_join_order_threshold

When GPORCA is enabled (the default), this parameter sets the maximum number of join children for which GPORCA will use the dynamic programming-based join ordering algorithm. You can set this value for a single query or for an entire session.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - INT_MAX</td>
<td>10</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

optimizer_mdcache_size

Sets the maximum amount of memory on the Greenplum Database master that GPORCA uses to cache query metadata (optimization data) during query optimization. The memory limit session based. GPORCA caches query metadata during query optimization with the default settings: GPORCA is enabled and optimizer_metadata_caching is on.

The default value is 16384 (16MB). This is an optimal value that has been determined through performance analysis.

You can specify a value in KB, MB, or GB. The default unit is KB. For example, a value of 16384 is 16384KB. A value of 1GB is the same as 1024MB or 1048576KB. If the value is 0, the size of the cache is not limited.

This parameter can be set for a database system, an individual database, or a session or query.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer &gt;= 0</td>
<td>16384</td>
<td>master session reload</td>
</tr>
</tbody>
</table>
optimizer_metadata_caching

When GPORCA is enabled (the default), this parameter specifies whether GPORCA caches query metadata (optimization data) in memory on the Greenplum Database master during query optimization. The default for this parameter is on, enable caching. The cache is session based. When a session ends, the cache is released. If the amount of query metadata exceeds the cache size, then old, unused metadata is evicted from the cache.

If the value is off, GPORCA does not cache metadata during query optimization.

This parameter can be set for a database system, an individual database, or a session or query.

The server configuration parameter optimizer_mdcache_size controls the size of the query metadata cache.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

optimizer_minidump

GPORCA generates minidump files to describe the optimization context for a given query. The information in the file is not in a format that can be easily used for debugging or troubleshooting. The minidump file is located under the master data directory and uses the following naming format:

Minidump_{date_time}.mdp

The minidump file contains this query related information:

- Catalog objects including data types, tables, operators, and statistics required by GPORCA
- An internal representation (DXL) of the query
- An internal representation (DXL) of the plan produced by GPORCA
- System configuration information passed to GPORCA such as server configuration parameters, cost and statistics configuration, and number of segments
- A stack trace of errors generated while optimizing the query

Setting this parameter to ALWAYS generates a minidump for all queries. Set this parameter to ONERROR to minimize total optimization time.

For information about GPORCA, see "Querying Data" in the Greenplum Database Administrator Guide.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONERROR</td>
<td>ONERROR</td>
<td>master</td>
</tr>
<tr>
<td>ALWAYS</td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

optimizer_nestloop_factor

When GPORCA is enabled (the default), this parameter controls the nested loop join cost factor to apply during query optimization. The default value, 1, specifies the default sort cost factor. The value is a ratio of increase or decrease from the default factor. For example, a value of 2.0 sets the cost factor at twice the default, and a value of 0.5 sets the factor at half the default.

The parameter can be set for a database system, an individual database, or a session or query.
### optimizer_parallel_union

When GPORCA is enabled (the default), `optimizer_parallel_union` controls the amount of parallelization that occurs for queries that contain a `UNION` or `UNION ALL` clause.

When the value is `off`, the default, GPORCA generates a query plan where each child of an `APPEND(UNION)` operator is in the same slice as the `APPEND` operator. During query execution, the children are executed in a sequential manner.

When the value is `on`, GPORCA generates a query plan where a redistribution motion node is under an `APPEND(UNION)` operator. During query execution, the children and the parent `APPEND` operator are on different slices, allowing the children of the `APPEND(UNION)` operator to execute in parallel on segment instances.

The parameter can be set for a database system, an individual database, or a session or query.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal &gt; 0</td>
<td>1</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

### optimizer_print_missing_stats

When GPORCA is enabled (the default), this parameter controls the display of table column information about columns with missing statistics for a query. The default value is `true`, display the column information to the client. When the value is `false`, the information is not sent to the client.

The information is displayed during query execution, or with the `EXPLAIN` or `EXPLAIN ANALYZE` commands.

The parameter can be set for a database system, an individual database, or a session.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>off</td>
<td>master, session, reload</td>
</tr>
</tbody>
</table>

### optimizer_print_optimization_stats

When GPORCA is enabled (the default), this parameter enables logging of GPORCA query optimization statistics for various optimization stages for a query. The default value is `off`, do not log optimization statistics. To log the optimization statistics, this parameter must be set to `on` and the parameter `client_min_messages` must be set to `log`.

- `set optimizer_print_optimization_stats = on;`
- `set client_min_messages = 'log';`

The information is logged during query execution, or with the `EXPLAIN` or `EXPLAIN ANALYZE` commands.
This parameter can be set for a database system, an individual database, or a session or query.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**optimizer_sort_factor**

When GPORCA is enabled (the default), optimizer_sort_factor controls the cost factor to apply to sorting operations during query optimization. The default value 1 specifies the default sort cost factor. The value is a ratio of increase or decrease from the default factor. For example, a value of 2.0 sets the cost factor at twice the default, and a value of 0.5 sets the factor at half the default.

The parameter can be set for a database system, an individual database, or a session or query.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal &gt; 0</td>
<td>1</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**password_encryption**

When a password is specified in `CREATE USER` or `ALTER USER` without writing either `ENCRYPTED` or `UNENCRYPTED`, this option determines whether the password is to be encrypted.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**password_hash_algorithm**

Specifies the cryptographic hash algorithm that is used when storing an encrypted Greenplum Database user password. The default algorithm is MD5.

For information about setting the password hash algorithm to protect user passwords, see "Protecting Passwords in Greenplum Database" in the [Greenplum Database Administrator Guide](https://www.greenplum.com/docs/pdca/6.1.1/r/en/gpdb-admin-guide.html).

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>MD5</td>
<td>master</td>
</tr>
<tr>
<td>SHA-256</td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>superuser</td>
</tr>
</tbody>
</table>

**pgstat_track_activity_query_size**

Sets the maximum length limit for the query text stored in `current_query` column of the system catalog view `pg_stat_activity`. The minimum length is 1024 characters.
<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>1024</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

### pljava_classpath

A colon (:) separated list of jar files or directories containing jar files needed for PL/Java functions. The full path to the jar file or directory must be specified, except the path can be omitted for jar files in the $GPHOME/lib/postgresql/java directory. The jar files must be installed in the same locations on all Greenplum hosts and readable by the gpadmin user.

The `pljava_classpath` parameter is used to assemble the PL/Java classpath at the beginning of each user session. Jar files added after a session has started are not available to that session.

If the full path to a jar file is specified in `pljava_classpath` it is added to the PL/Java classpath. When a directory is specified, any jar files the directory contains are added to the PL/Java classpath. The search does not descend into subdirectories of the specified directories. If the name of a jar file is included in `pljava_classpath` with no path, the jar file must be in the $GPHOME/lib/postgresql/java directory.

**Note:** Performance can be affected if there are many directories to search or a large number of jar files.

If `pljava_classpath_insecure` is `false`, setting the `pljava_classpath` parameter requires superuser privilege. Setting the classpath in SQL code will fail when the code is executed by a user without superuser privilege. The `pljava_classpath` parameter must have been set previously by a superuser or in the `postgresql.conf` file. Changing the classpath in the `postgresql.conf` file requires a reload (gpstop -u).

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td></td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>superuser</td>
</tr>
</tbody>
</table>

### pljava_classpath_insecure

Controls whether the server configuration parameter `pljava_classpath` can be set by a user without Greenplum Database superuser privileges. When `true`, `pljava_classpath` can be set by a regular user. Otherwise, `pljava_classpath` can be set only by a database superuser. The default is `false`.

**Warning:** Enabling this parameter exposes a security risk by giving non-administrator database users the ability to run unauthorized Java methods.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>false</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>superuser</td>
</tr>
</tbody>
</table>
pljava_statement_cache_size
Sets the size in KB of the JRE MRU (Most Recently Used) cache for prepared statements.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of kilobytes</td>
<td>10</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>superuser</td>
</tr>
</tbody>
</table>

pljava_release_lingering_savepoints
If true, lingering savepoints used in PL/Java functions will be released on function exit. If false, savepoints will be rolled back.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>true</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>superuser</td>
</tr>
</tbody>
</table>

pljava_vmoptions
Defines the startup options for the Java VM. The default value is an empty string (""").

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td></td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>superuser</td>
</tr>
</tbody>
</table>

port
The database listener port for a Greenplum instance. The master and each segment has its own port. Port numbers for the Greenplum system must also be changed in the gp_segment_configuration catalog. You must shut down your Greenplum Database system before changing port numbers.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>any valid port number</td>
<td>5432</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>
random_page_cost
Sets the estimate of the cost of a nonsequentially fetched disk page for the legacy query optimizer (planner). This is measured as a multiple of the cost of a sequential page fetch. A higher value makes it more likely a sequential scan will be used, a lower value makes it more likely an index scan will be used.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>floating point</td>
<td>100</td>
<td>master session reload</td>
</tr>
</tbody>
</table>

readable_external_table_timeout
When an SQL query reads from an external table, the parameter value specifies the amount of time in seconds that Greenplum Database waits before cancelling the query when data stops being returned from the external table.

The default value of 0, specifies no time out. Greenplum Database does not cancel the query.

If queries that use gpfdist run a long time and then return the error "intermittent network connectivity issues", you can specify a value for readable_external_table_timeout. If no data is returned by gpfdist for the specified length of time, Greenplum Database cancels the query.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer &gt;= 0</td>
<td>0</td>
<td>master system reload</td>
</tr>
</tbody>
</table>

repl_catchup_within_range
For Greenplum Database master mirroring, controls updates to the active master. If the number of WAL segment files that have not been processed by the walsender exceeds this value, Greenplum Database updates the active master.

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.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 64</td>
<td>1</td>
<td>master system reload superuser</td>
</tr>
</tbody>
</table>

replication_timeout
For Greenplum Database master mirroring, sets the maximum time in milliseconds that the walsender process on the active master waits for a status message from the walreceiver process on the standby master. If a message is not received, the walsender logs an error message.

The wal_receiver_status_interval controls the interval between walreceiver status messages.
### Value Range

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - INT_MAX</td>
<td>60000 ms (60 seconds)</td>
<td>master, system, reload, superuser</td>
</tr>
</tbody>
</table>

### regex_flavor

The 'extended' setting may be useful for exact backwards compatibility with pre-7.4 releases of PostgreSQL.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>advanced</td>
<td>advanced</td>
<td>master, session, reload</td>
</tr>
<tr>
<td>extended</td>
<td></td>
<td></td>
</tr>
<tr>
<td>basic</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### resource_cleanup_gangs_on_wait

If a statement is submitted through a resource queue, clean up any idle query executor worker processes before taking a lock on the resource queue.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master, system, restart</td>
</tr>
</tbody>
</table>

### resource_select_only

Sets the types of queries managed by resource queues. If set to on, then SELECT, SELECT INTO, CREATE TABLE AS SELECT, and DECLARE CURSOR commands are evaluated. If set to off INSERT, UPDATE, and DELETE commands will be evaluated as well.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master, system, restart</td>
</tr>
</tbody>
</table>

### runaway_detector_activation_percent

Sets the percentage of Greenplum Database vmem memory that triggers the termination of queries. If the percentage of vmem memory that is utilized for a Greenplum Database segment exceeds the specified value, Greenplum Database terminates queries based on memory usage, starting with the query consuming the largest amount of memory. Queries are terminated until the percentage of utilized vmem is below the specified percentage.

Specify the maximum vmem value for active Greenplum Database segment instances with the server configuration parameter `gp_vmem_protect_limit`.
For example, if vmem memory is set to 10GB, and the value of `runaway_detector_activation_percent` is 90 (90%), Greenplum Database starts terminating queries when the utilized vmem memory exceeds 9 GB.

A value of 0 disables the automatic termination of queries based on percentage of vmem that is utilized.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>percentage (integer)</td>
<td>90</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**search_path**

Specifies the order in which schemas are searched when an object is referenced by a simple name with no schema component. When there are objects of identical names in different schemas, the one found first in the search path is used. The system catalog schema, `pg_catalog`, is always searched, whether it is mentioned in the path or not. When objects are created without specifying a particular target schema, they will be placed in the first schema listed in the search path. The current effective value of the search path can be examined via the SQL function `current_schemas()`. `current_schemas()` shows how the requests appearing in `search_path` were resolved.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>a comma-separated list of schema names</td>
<td>$user,public</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**seq_page_cost**

For the legacy query optimizer (planner), sets the estimate of the cost of a disk page fetch that is part of a series of sequential fetches.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>floating point</td>
<td>1</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**server_encoding**

Reports the database encoding (character set). It is determined when the Greenplum Database array is initialized. Ordinarily, clients need only be concerned with the value of `client_encoding`.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;system dependent&gt;</td>
<td>UTF8</td>
<td>read only</td>
</tr>
</tbody>
</table>

**server_version**

Reports the version of PostgreSQL that this release of Greenplum Database is based on.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>8.3.23</td>
<td>read only</td>
</tr>
</tbody>
</table>
**server_version_num**

Reports the version of PostgreSQL that this release of Greenplum Database is based on as an integer.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>80323</td>
<td>read only</td>
</tr>
</tbody>
</table>

**shared_buffers**

Sets the amount of memory a Greenplum Database segment instance uses for shared memory buffers. This setting must be at least 128KB and at least 16KB times `max_connections`.

Each Greenplum Database segment instance calculates and attempts to allocate certain amount of shared memory based on the segment configuration. The value of `shared_buffers` is significant portion of this shared memory calculation, but is not all it. When setting `shared_buffers`, the values for the operating system parameters `SHMMAX` or `SHMALL` might also need to be adjusted.

The operating system parameter `SHMMAX` specifies maximum size of a single shared memory allocation. The value of `SHMMAX` must be greater than this value:

\[
\text{shared_buffers} + \text{other_seg_shmem}
\]

The value of `other_seg_shmem` is the portion the Greenplum Database shared memory calculation that is not accounted for by the `shared_buffers` value. The `other_seg_shmem` value will vary based on the segment configuration.

With the default Greenplum Database parameter values, the value for `other_seg_shmem` is approximately 111MB for Greenplum Database segments and approximately 79MB for the Greenplum Database master.

The operating system parameter `SHMALL` specifies the maximum amount of shared memory on the host. The value of `SHMALL` must be greater than this value:

\[
(\text{num_instances_per_host} \times (\text{shared_buffers} + \text{other_seg_shmem})) + \text{other_app_shared_mem}
\]

The value of `other_app_shared_mem` is the amount of shared memory that is used by other applications and processes on the host.

When shared memory allocation errors occur, possible ways to resolve shared memory allocation issues are to increase `SHMMAX` or `SHMALL`, or decrease `shared_buffers` or `max_connections`.

See the *Greenplum Database Installation Guide* for information about the Greenplum Database values for the parameters `SHMMAX` and `SHMALL`.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer &gt; 16K * <code>max_connections</code></td>
<td>125MB</td>
<td>local system restart</td>
</tr>
</tbody>
</table>

**shared_preload_libraries**

A comma-separated list of shared libraries that are to be preloaded at server start. PostgreSQL procedural language libraries can be preloaded in this way, typically by using the syntax `$libdir/plXXX` where `XXX` is pgsql, perl, tcl, or python. By preloading a shared library, the library startup time is avoided when the library is first used. If a specified library is not found, the server will fail to start.
### ssl

Enables SSL connections.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
</tbody>
</table>

### ssl_ciphers

Specifies a list of SSL ciphers that are allowed to be used on secure connections. See the openssl manual page for a list of supported ciphers.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>ALL</td>
<td>master</td>
</tr>
</tbody>
</table>

### standard_conforming_strings

Determines whether ordinary string literals ("...") treat backslashes literally, as specified in the SQL standard. The default value is on. Turn this parameter off to treat backslashes in string literals as escape characters instead of literal backslashes. Applications may check this parameter to determine how string literals are processed. The presence of this parameter can also be taken as an indication that the escape string syntax (E'...') is supported.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>master</td>
</tr>
</tbody>
</table>

### statement_mem

Allocates segment host memory per query. The amount of memory allocated with this parameter cannot exceed `max_statement_mem` or the memory limit on the resource queue through which the query was submitted. When `gp_resqueue_memory_policy = auto`, `statement_mem` and resource queue memory limits control query memory usage.

If additional memory is required for a query, temporary spill files on disk are used.

This calculation can be used to estimate a reasonable value for a wide variety of situations.

\[( \text{gp_vmem_protect_limitGB} \times .9 ) / \text{max_expected_concurrent_queries} \]
With the `gp_vmem_protect_limit` set to 8192MB (8GB) and assuming a maximum of 40 concurrent queries with a 10% buffer

\[(8\text{GB} \times .9) / 40 = .18\text{GB} = 184\text{MB}\]

When changing both `max_statement_mem` and `statement_mem`, `max_statement_mem` must be changed first, or listed first in the `postgresql.conf` file.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of kilobytes</td>
<td>128MB</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**statement_timeout**

Abort any statement that takes over the specified number of milliseconds. 0 turns off the limitation.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of milliseconds</td>
<td>0</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**stats_queue_level**

Collects resource queue statistics on database activity.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**superuser_reserved_connections**

Determines the number of connection slots that are reserved for Greenplum Database superusers.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer &lt; <code>max_connections</code></td>
<td>3</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**tcp_keepalives_count**

How many keepalives may be lost before the connection is considered dead. A value of 0 uses the system default. If TCP_KEEPCNT is not supported, this parameter must be 0.

Use this parameter for all connections that are not between a primary and mirror segment. Use `gp_filerep_tcp_keepalives_count` for settings that are between a primary and mirror segment.
Value Range | Default | Set Classifications
--- | --- | ---
number of lost keepalives | 0 | local system restart

tcp_keepalives_idle

Number of seconds between sending keepalives on an otherwise idle connection. A value of 0 uses the system default. If TCP_KEEPIDLE is not supported, this parameter must be 0.

Use this parameter for all connections that are not between a primary and mirror segment. Use gp_filerep_tcp_keepalives_idle for settings that are between a primary and mirror segment.

Value Range | Default | Set Classifications
--- | --- | ---
number of seconds | 0 | local system restart

tcp_keepalives_interval

How many seconds to wait for a response to a keepalive before retransmitting. A value of 0 uses the system default. If TCP_KEEPINTVL is not supported, this parameter must be 0.

Use this parameter for all connections that are not between a primary and mirror segment. Use gp_filerep_tcp_keepalives_interval for settings that are between a primary and mirror segment.

Value Range | Default | Set Classifications
--- | --- | ---
number of seconds | 0 | local system restart

temp_buffers

Sets the maximum number of temporary buffers used by each database session. These are session-local buffers used only for access to temporary tables. The setting can be changed within individual sessions, but only up until the first use of temporary tables within a session. The cost of setting a large value in sessions that do not actually need a lot of temporary buffers is only a buffer descriptor, or about 64 bytes, per increment. However if a buffer is actually used, an additional 8192 bytes will be consumed.

Value Range | Default | Set Classifications
--- | --- | ---
integer | 1024 | master session reload

TimeZone

Sets the time zone for displaying and interpreting time stamps. The default is to use whatever the system environment specifies as the time zone. See Date/Time Keywords in the PostgreSQL documentation.
### timezone_abbreviations

Sets the collection of time zone abbreviations that will be accepted by the server for date time input. The default is `Default`, which is a collection that works in most of the world. Australia and India, and other collections can be defined for a particular installation. Possible values are names of configuration files stored in `$GPHOME/share/postgresql/timezonesets/`.

To configure Greenplum Database to use a custom collection of timezones, copy the file that contains the timezone definitions to the directory `$GPHOME/share/postgresql/timezonesets/` on the Greenplum Database master and segment hosts. Then set value of the server configuration parameter `timezone_abbreviations` to the file. For example, to use a file `custom` that contains the default timezones and the WIB (Waktu Indonesia Barat) timezone.

1. Copy the file `Default` from the directory `$GPHOME/share/postgresql/timezonesets/` the file `custom`. Add the WIB timezone information from the file `Asia.txt` to the `custom`.
2. Copy the file `custom` to the directory `$GPHOME/share/postgresql/timezonesets/` on the Greenplum Database master and segment hosts.
3. Set value of the server configuration parameter `timezone_abbreviations` to `custom`.
4. Reload the server configuration file (`gpstop -u`).

### track_activities

Enables the collection of statistics on the currently executing command of each session, along with the time at which that command began execution. When enabled, this information is not visible to all users, only to superusers and the user owning the session. This data can be accessed via the `pg_stat_activity` system view.

### track_counts

Enables the collection of row and block level statistics on database activity. If enabled, the data that is produced can be accessed via the `pg_stat` and `pg_statio` family of system views.
### transaction_isolation

Sets the current transaction's isolation level.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>read committed</td>
<td>read committed</td>
<td>master</td>
</tr>
<tr>
<td>serializable</td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

### transaction_read_only

Sets the current transaction's read-only status.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

### transform_null_equals

When on, expressions of the form `expr = NULL` (or `NULL = expr`) are treated as `expr IS NULL`, that is, they return true if `expr` evaluates to the null value, and false otherwise. The correct SQL-spec-compliant behavior of `expr = NULL` is to always return null (unknown).

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

### unix_socket_directory

Specifies the directory of the UNIX-domain socket on which the server is to listen for connections from client applications.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory path</td>
<td>unset</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>
**unix_socket_group**
Sets the owning group of the UNIX-domain socket. By default this is an empty string, which uses the default group for the current user.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX group name</td>
<td>unset</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**unix_socket_permissions**
Sets the access permissions of the UNIX-domain socket. UNIX-domain sockets use the usual UNIX file system permission set. Note that for a UNIX-domain socket, only write permission matters.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>numeric UNIX file permission mode (as accepted by the chmod or umask commands)</td>
<td>511</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**update_process_title**
Enables updating of the process title every time a new SQL command is received by the server. The process title is typically viewed by the `ps` command.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>on</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**vacuum_cost_delay**
The length of time that the process will sleep when the cost limit has been exceeded. 0 disables the cost-based vacuum delay feature.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>milliseconds &lt; 0 (in multiples of 10)</td>
<td>0</td>
<td>local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restart</td>
</tr>
</tbody>
</table>

**vacuum_cost_limit**
The accumulated cost that will cause the vacuuming process to sleep.
### vacuum_cost_page_dirty

The estimated cost charged when vacuum modifies a block that was previously clean. It represents the extra I/O required to flush the dirty block out to disk again.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer &gt; 0</td>
<td>200</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>

### vacuum_cost_page_hit

The estimated cost for vacuuming a buffer found in the shared buffer cache. It represents the cost to lock the buffer pool, lookup the shared hash table and scan the content of the page.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer &gt; 0</td>
<td>1</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>

### vacuum_cost_page_miss

The estimated cost for vacuuming a buffer that has to be read from disk. This represents the effort to lock the buffer pool, lookup the shared hash table, read the desired block in from the disk and scan its content.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer &gt; 0</td>
<td>10</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>

### vacuum_freeze_min_age

Specifies the cutoff age (in transactions) that VACUUM should use to decide whether to replace transaction IDs with FrozenXID while scanning a table.

For information about VACUUM and transaction ID management, see "Managing Data" in the Greenplum Database Administrator Guide and the PostgreSQL documentation.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer 0-1000000000000</td>
<td>1000000000</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>
**validate_previous_free_tid**

Enables a test that validates the free tuple ID (TID) list. The list is maintained and used by Greenplum Database. Greenplum Database determines the validity of the free TID list by ensuring the previous free TID of the current free tuple is a valid free tuple. The default value is `true`, enable the test.

If Greenplum Database detects a corruption in the free TID list, the free TID list is rebuilt, a warning is logged, and a warning is returned by queries for which the check failed. Greenplum Database attempts to execute the queries.

**Note:** If a warning is returned, please contact Pivotal Support.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>true</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**vmem_process_interrupt**

Enables checking for interrupts before reserving vmem memory for a query during Greenplum Database query execution. Before reserving further vmem for a query, check if the current session for the query has a pending query cancellation or other pending interrupts. This ensures more responsive interrupt processing, including query cancellation requests. The default is `off`.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>off</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>

**wal_receiver_status_interval**

For Greenplum Database master mirroring, sets the interval in seconds between `walreceiver` process status messages that are sent to the active master. Under heavy loads, the time might be longer.

The value of `replication_timeout` controls the time that the `walsender` process waits for a `walreceiver` message.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer 0- INT_MAX/1000</td>
<td>10 sec</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>superuser</td>
</tr>
</tbody>
</table>

**writable_external_table_bufsize**

Size of the buffer (in KB) that Greenplum Database uses for network communication, such as the `gpfdist` utility and external web tables (that use http). Greenplum Database stores data in the buffer before writing the data out. For information about `gpfdist`, see the *Greenplum Database Utility Guide*. 
### xid_stop_limit

The number of transaction IDs prior to the ID where transaction ID wraparound occurs. When this limit is reached, Greenplum Database stops creating new transactions to avoid data loss due to transaction ID wraparound.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer 32 - 131072 (32KB - 128MB)</td>
<td>64</td>
<td>local, system, reload</td>
</tr>
</tbody>
</table>

### xid_warn_limit

The number of transaction IDs prior to the limit specified by `xid_stop_limit`. When Greenplum Database reaches this limit, it issues a warning to perform a VACUUM operation to avoid data loss due to transaction ID wraparound.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer 10000000 - 2000000000</td>
<td>1000000000</td>
<td>local, system, restart</td>
</tr>
</tbody>
</table>

### xmlbinary

Specifies how binary values are encoded in XML data. For example, when `bytea` values are converted to XML. The binary data can be converted to either base64 encoding or hexadecimal encoding. The default is base64.

The parameter can be set for a database system, an individual database, or a session.

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>base64</td>
<td>base64</td>
<td>master, session, reload</td>
</tr>
<tr>
<td>hex</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### xmloption

Specifies whether XML data is to be considered as an XML document (document) or XML content fragment (content) for operations that perform implicit parsing and serialization. The default is content.

This parameter affects the validation performed by `xml_is_well_formed()`. If the value is `document`, the function checks for a well-formed XML document. If the value is `content`, the function checks for a well-formed XML content fragment.
**Note:** An XML document that contains a document type declaration (DTD) is not considered a valid XML content fragment. If `xmloption` set to `content`, XML that contains a DTD is not considered valid XML.

To cast a character string that contains a DTD to the `xml` data type, use the `xmlparse` function with the `document` keyword, or change the `xmloption` value to `document`.

The parameter can be set for a database system, an individual database, or a session. The SQL command to set this option for a session is also available in Greenplum Database.

```
SET XML OPTION { DOCUMENT | CONTENT }
```

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Default</th>
<th>Set Classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>document</td>
<td>content</td>
<td>master</td>
</tr>
<tr>
<td>content</td>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reload</td>
</tr>
</tbody>
</table>
Summary of Built-in Functions

Greenplum Database supports built-in functions and operators including analytic functions and window functions that can be used in window expressions. For information about using built-in Greenplum Database functions see, "Using Functions and Operators" in the Greenplum Database Administrator Guide.

• Greenplum Database Function Types
• Built-in Functions and Operators
• JSON Functions and Operators
• Window Functions
• Advanced Aggregate Functions

Greenplum Database Function Types

Greenplum Database evaluates functions and operators used in SQL expressions. Some functions and operators are only allowed to execute on the master since they could lead to inconsistencies in Greenplum Database segment instances. This table describes the Greenplum Database Function Types.

Table 227: Functions in Greenplum Database

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Greenplum Support</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMMUTABLE</td>
<td>Yes</td>
<td>Relies only on information directly in its argument list. Given the same argument values, always returns the same result.</td>
<td></td>
</tr>
<tr>
<td>STABLE</td>
<td>Yes, in most cases</td>
<td>Within a single table scan, returns the same result for same argument values, but results change across SQL statements.</td>
<td>Results depend on database lookups or parameter values. current_timestamp family of functions is STABLE; values do not change within an execution.</td>
</tr>
<tr>
<td>VOLATILE</td>
<td>Restricted</td>
<td>Function values can change within a single table scan. For example: random(), timeofday().</td>
<td>Any function with side effects is volatile, even if its result is predictable. For example: setval().</td>
</tr>
</tbody>
</table>

In Greenplum Database, data is divided up across segments — each segment is a distinct PostgreSQL database. To prevent inconsistent or unexpected results, do not execute functions classified as VOLATILE at the segment level if they contain SQL commands or modify the database in any way. For example, functions such as setval() are not allowed to execute on distributed data in Greenplum Database because they can cause inconsistent data between segment instances.

To ensure data consistency, you can safely use VOLATILE and STABLE functions in statements that are evaluated on and run from the master. For example, the following statements run on the master (statements without a FROM clause):

```
SELECT setval('myseq', 201);
SELECT foo();
```
If a statement has a `FROM` clause containing a distributed table and the function in the `FROM` clause returns a set of rows, the statement can run on the segments:

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

Greenplum Database does not support functions that return a table reference (`rangeFuncs`) or functions that use the `refCursor` datatype.

**Built-in Functions and Operators**

The following table lists the categories of built-in functions and operators supported by PostgreSQL. All functions and operators are supported in Greenplum Database as in PostgreSQL with the exception of `STABLE` and `VOLATILE` functions, which are subject to the restrictions noted in Greenplum Database Function Types. See the Functions and Operators section of the PostgreSQL documentation for more information about these built-in functions and operators.

**Table 228: Built-in functions and operators**

<table>
<thead>
<tr>
<th>Operator/Function Category</th>
<th>VOLATILE Functions</th>
<th>STABLE Functions</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical Operators</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comparison Operators</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mathematical Functions and Operators</td>
<td>random</td>
<td>convert</td>
<td></td>
</tr>
<tr>
<td>String Functions and Operators</td>
<td>All built-in conversion functions</td>
<td>pg_client_encoding</td>
<td></td>
</tr>
<tr>
<td>Binary String Functions and Operators</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit String Functions and Operators</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pattern Matching</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Type Formatting Functions</td>
<td>to_char</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date/Time Functions and Operators</td>
<td>timeofday</td>
<td>age</td>
<td></td>
</tr>
<tr>
<td>Enum Support Functions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometric Functions and Operators</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operator/Function Category</td>
<td>VOLATILE Functions</td>
<td>STABLE Functions</td>
<td>Restrictions</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------</td>
<td>-----------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Network Address Functions and Operators</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence Manipulation Functions</td>
<td>nextval()</td>
<td>setval()</td>
<td></td>
</tr>
<tr>
<td>Conditional Expressions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Array Functions and Operators</td>
<td></td>
<td>All array functions</td>
<td></td>
</tr>
<tr>
<td>Aggregate Functions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subquery Expressions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Row and Array Comparisons</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set Returning Functions</td>
<td>generate_series</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Information Functions</td>
<td></td>
<td>All session information functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>All access privilege inquiry functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>All schema visibility inquiry functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>All system catalog information functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>All comment information functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>All transaction ids and snapshots</td>
<td></td>
</tr>
<tr>
<td>System Administration Functions</td>
<td>set_config</td>
<td>current_setting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pg_cancel_backend</td>
<td></td>
<td>Note: The function pg_column_size displays bytes required to store the value, possibly with TOAST compression.</td>
</tr>
<tr>
<td></td>
<td>pg_reload_conf</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pg_rotate_logfile</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pg_start_backup</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pg_stop_backup</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pg_size.pretty</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pg_ls_dir</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pg_read_file</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pg_stat_file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operator/Function Category</td>
<td>VOLATILE Functions</td>
<td>STABLE Functions</td>
<td>Restrictions</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------</td>
<td>------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>XML Functions and function-like expressions</td>
<td></td>
<td>cursor_to_xml(cursor refcursor, count int, nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>cursor_to_xmlschema(cursor refcursor, nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>database_to_xml(nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>database_to_xmlschema(nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>database_to_xml_and_xmlschema(nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>query_to_xml(query text, nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>query_to_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>query_to_xml_and_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>schema_to_xml(schema name, nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>schema_to_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>schema_to_xml_and_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>table_to_xml(tbl regclass, nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>table_to_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text)</td>
<td></td>
</tr>
<tr>
<td>Operator/Function Category</td>
<td>VOLATILE Functions</td>
<td>STABLE Functions</td>
<td>Restrictions</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------</td>
<td>-----------------</td>
<td>--------------</td>
</tr>
<tr>
<td></td>
<td>table_to_xml_and_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetsns text)</td>
<td>xmlagg(xml)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>xmlconcat(xml[, ...])</td>
<td>xmlconcat(xml[, ...])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>xmlelement(name name [, xmlattributes(value [AS attname] [, ...])] [, content, ...])</td>
<td>xmlexists(text, xml)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>xmlforest(content [AS name] [, ...])</td>
<td>xml_is_well_formed(text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>xml_is_well_formed_document(text)</td>
<td>xml_is_well_formed_document(text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>xml_is_well_formed_content(text)</td>
<td>xml_is_well_formed_content(text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>xmlparse ( { DOCUMENT</td>
<td>CONTENT } value)</td>
<td>xpath(text, xml)</td>
</tr>
<tr>
<td></td>
<td>xpath(text, xml, text[])</td>
<td>xpath_exists(text, xml)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>xpath_exists(text, xml, text[])</td>
<td>xmlpi(name target [, content])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>xmlroot(xml, version text</td>
<td>no value [, standalone yes</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>xmlserialize ( { DOCUMENT</td>
<td>CONTENT } value AS type )</td>
<td>xml(text)</td>
</tr>
<tr>
<td></td>
<td>text(xml)</td>
<td>text(xml)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>xmlcomment(xml)</td>
<td>xmlconcat2(xml, xml)</td>
<td></td>
</tr>
</tbody>
</table>

**JSON Functions and Operators**

Built-in functions and operators that create and manipulate JSON data.

- **JSON Operators**
- **JSON Creation Functions**
- **JSON Processing Functions**

**Note:** For `json` values, all key/value pairs are kept even if a JSON object contains duplicate key/value pairs. The processing functions consider the last value as the operative one.
JSON Operators

This table describes the operators that are available for use with the `json` data type.

**Table 229: json Operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Right Operand Type</th>
<th>Description</th>
<th>Example</th>
<th>Example Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-&gt;</code></td>
<td>int</td>
<td>Get JSON array element (indexed from zero).</td>
<td>'[[&quot;a&quot;:&quot;foo&quot;],[&quot;b&quot;:&quot;bar&quot;],[&quot;c&quot;:&quot;baz&quot;]]'::json-&gt;&gt;2</td>
<td>{&quot;c&quot;:&quot;baz&quot;}</td>
</tr>
<tr>
<td><code>-&gt;</code></td>
<td>text</td>
<td>Get JSON object field by key.</td>
<td>'{&quot;a&quot;:{&quot;b&quot;:&quot;foo&quot;}}'::json-&gt;&gt;'a'</td>
<td>&quot;{b&quot;:&quot;foo&quot;}</td>
</tr>
<tr>
<td><code>&gt;&gt;</code></td>
<td>int</td>
<td>Get JSON array element as text.</td>
<td>'[1,2,3]'::json-&gt;&gt;&gt;2</td>
<td>3</td>
</tr>
<tr>
<td><code>&gt;&gt;</code></td>
<td>text</td>
<td>Get JSON object field as text.</td>
<td>'{&quot;a&quot;:1,&quot;b&quot;:2}'::json-&gt;&gt;'b'</td>
<td>2</td>
</tr>
<tr>
<td><code>#&gt;</code></td>
<td>text[]</td>
<td>Get JSON object at specified path.</td>
<td>'{&quot;a&quot;:{&quot;b&quot;:{&quot;c&quot;:&quot;foo&quot;}}}':'json#&gt;'{a,b}'</td>
<td>{&quot;c&quot;: &quot;foo&quot;}</td>
</tr>
<tr>
<td><code>#&gt;&gt;</code></td>
<td>text[]</td>
<td>Get JSON object at specified path as text.</td>
<td>'{&quot;a&quot;:[1,2,3],&quot;b&quot;: [4,5,6]}':'json#&gt;&gt;'{a,2}'</td>
<td>3</td>
</tr>
</tbody>
</table>

JSON Creation Functions

This table describes the functions that create `json` values.

**Table 230: JSON Creation Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
<th>Example Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>array_to_json(anyarray [, pretty_bool])</code></td>
<td>Returns the array as a JSON array. A Greenplum Database multidimensional array becomes a JSON array of arrays. Line feeds are added between dimension 1 elements if <code>pretty_bool</code> is true.</td>
<td><code>array_to_json('{{1,5},{99,100}}'::int[])</code></td>
<td><code>[[1,5],[99,100]]</code></td>
</tr>
<tr>
<td><code>row_to_json(record [, pretty_bool])</code></td>
<td>Returns the row as a JSON object. Line feeds are added between level 1 elements if <code>pretty_bool</code> is true.</td>
<td><code>row_to_json(row(1,'foo'))</code></td>
<td><code>{&quot;f1&quot;:1,&quot;f2&quot;:&quot;foo&quot;}</code></td>
</tr>
</tbody>
</table>
Greenplum Database Reference Guide

Release Notes

JSON Processing Functions
This table describes the functions that process json values.
Table 231: JSON Processing Functions
Function

Return Type

Description

Example

json_
each(json)

setof key
text, value
json

Expands the
outermost JSON
object into a set of
key/value pairs.

select *
key | value
from json_
-----+------| "foo"
each('{"a":"foo", a
b
| "bar"
"b":"bar"}')

setof key
text, value
jsonb

Example Result

json_each_
text(json)

setof key
text, value
text

Expands the
outermost JSON
object into a set
of key/value pairs.
The returned
values are of type
text.

select * from
key | value
json_each_
-----+------| foo
text('{"a":"foo", a
b
| bar
"b":"bar"}')

json_extract_
path(from_json
json, VARIADIC
path_elems
text[])

json

Returns the JSON
value specified to
by path_elems.
Equivalent to #>
operator.

json_extract_
{"f5":99,"f6":"foo"}
path('{"f2":
{"f3":1},"f4":
{"f5":99,"f6":"foo"}}','f4')

json_
extract_path_
text(from_json
json, VARIADIC
path_elems
text[])

text

Returns the JSON
value specified to
by path_elems
as text. Equivalent
to #>> operator.

json_extract_
foo
path_text('{"f2":
{"f3":1},"f4":
{"f5":99,"f6":"foo"}}','f4',
'f6')

json_object_
keys(json)

setof text

Returns set
of keys in the
outermost JSON
object.

json_object_
json_object_
keys('{"f1":"abc","f2":
keys
-----------------{"f3":"a",
f1
"f4":"b"}}')
f2

json_populate_
record(base
anyelement,
from_json
json)

anyelement

Expands the
object in from_
json to a row
whose columns
match the record
type defined by
base. See Note.

select * from
a | b
json_populate_
---+--1 | 2
record(null::myrowtype,
'{"a":1,"b":2}')

1340


<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Description</th>
<th>Example</th>
<th>Example Result</th>
</tr>
</thead>
</table>
| json_populate_recordset(base anyelement, from_json json) | setof anyelement | Expands the outermost array of objects in from_json to a set of rows whose columns match the record type defined by base. See Note. | 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)                   | setof json  | Expands a JSON array to a set of JSON values.                               | select * from json_array_elements('[1,true,
[2,false]]') | value
|                                              |             |                                                                              |                                              | 1
|                                              |             |                                                                              |                                              | true
|                                              |             |                                                                              |                                              | [2,false] |

**Note:** Many of these functions and operators convert Unicode escapes in JSON strings to regular characters. The functions throw an error for characters that cannot be represented in the database encoding.

For `json_populate_record` and `json_populate_recordset`, type coercion from JSON is best effort and might not result in desired values for some types. JSON keys are matched to identical column names in the target row type. JSON fields that do not appear in the target row type are omitted from the output, and target columns that do not match any JSON field return NULL.

### Window Functions

The following built-in window functions are Greenplum extensions to the PostgreSQL database. All window functions are **immutable**. For more information about window functions, see "Window Expressions" in the *Greenplum Database Administrator Guide*.

**Table 232: Window functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Full Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cume_dist()</td>
<td>double precision</td>
<td>CUME_DIST() OVER ( [PARTITION BY expr ] ORDER BY expr )</td>
<td>Calculates the cumulative distribution of a value in a group of values. Rows with equal values always evaluate to the same cumulative distribution value.</td>
</tr>
<tr>
<td>dense_rank()</td>
<td>bigint</td>
<td>DENSE_RANK () OVER ( [PARTITION BY expr ] ORDER BY expr )</td>
<td>Computes the rank of a row in an ordered group of rows without skipping rank values. Rows with equal values are given the same rank value.</td>
</tr>
<tr>
<td>first_value(expr)</td>
<td>same as input expr type</td>
<td>FIRST_VALUE(expr) OVER ( [PARTITION BY expr ] ORDER BY expr [ROWS</td>
<td>RANGE frame_expr ] )</td>
</tr>
<tr>
<td>Function</td>
<td>Return Type</td>
<td>Full Syntax</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>lag(expr [, offset] [, default])</td>
<td>same as input expr type</td>
<td>LAG( expr [, offset] [, default]) OVER ( [PARTITION BY expr] ORDER BY expr )</td>
<td>Provides access to more than one row of the same table without doing a self join. Given a series of rows returned from a query and a position of the cursor, LAG provides access to a row at a given physical offset prior to that position. The default offset is 1. <code>default</code> sets the value that is returned if the offset goes beyond the scope of the window. If <code>default</code> is not specified, the default value is null.</td>
</tr>
<tr>
<td>last_value(expr)</td>
<td>same as input expr type</td>
<td>LAST_VALUE(expr) OVER ( [PARTITION BY expr] ORDER BY expr [ROWS</td>
<td>RANGE frame_expr] )</td>
</tr>
<tr>
<td>lead(expr [, offset] [, default])</td>
<td>same as input expr type</td>
<td>LEAD(expr [,offset] [,exprrdefault]) OVER ( [PARTITION BY expr] ORDER BY expr )</td>
<td>Provides access to more than one row of the same table without doing a self join. Given a series of rows returned from a query and a position of the cursor, lead provides access to a row at a given physical offset after that position. If offset is not specified, the default offset is 1. <code>default</code> sets the value that is returned if the offset goes beyond the scope of the window. If <code>default</code> is not specified, the default value is null.</td>
</tr>
<tr>
<td>ntile(expr)</td>
<td>bigint</td>
<td>NTILE(expr) OVER ( [PARTITION BY expr] ORDER BY expr )</td>
<td>Divides an ordered data set into a number of buckets (as defined by expr) and assigns a bucket number to each row.</td>
</tr>
<tr>
<td>percent_rank()</td>
<td>double precision</td>
<td>PERCENT_RANK () OVER ( [PARTITION BY expr] ORDER BY expr )</td>
<td>Calculates the rank of a hypothetical row R minus 1, divided by 1 less than the number of rows being evaluated (within a window partition).</td>
</tr>
</tbody>
</table>
### Advanced Aggregate Functions

The following built-in advanced analytic functions are Greenplum extensions of the PostgreSQL database. Analytic functions are immutable.

**Note:** The Greenplum MADlib Extension for Analytics provides additional advanced functions to perform statistical analysis and machine learning with Greenplum Database data. See Greenplum MADlib Extension for Analytics.

Table 233: Advanced Aggregate Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Full Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rank()</code></td>
<td>bigint</td>
<td><code>RANK () OVER ( [PARTITION BY expr] ORDER BY expr )</code></td>
<td>Calculates the rank of a row in an ordered group of values. Rows with equal values for the ranking criteria receive the same rank. The number of tied rows are added to the rank number to calculate the next rank value. Ranks may not be consecutive numbers in this case.</td>
</tr>
<tr>
<td><code>row_number()</code></td>
<td>bigint</td>
<td><code>ROW_NUMBER () OVER ( [PARTITION BY expr] ORDER BY expr )</code></td>
<td>Assigns a unique number to each row to which it is applied (either each row in a window partition or each row of the query).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Full Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MEDIAN (expr)</code></td>
<td>timestamp,</td>
<td><code>MEDIAN (expression)</code></td>
<td>Can take a two-dimensional array as input. Treats such arrays as matrices.</td>
</tr>
<tr>
<td></td>
<td>timestamptz,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>interval, float</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>PERCENTILE_CONT(expr)</code></td>
<td>timestamp,</td>
<td><code>PERCENTILE_CONT(percentage) WITHIN GROUP (ORDER BY expression)</code></td>
<td>Performs an inverse distribution function that assumes a continuous distribution model. It takes a percentile value and a sort specification and returns the same datatype as the numeric datatype of the argument. This returned value is a computed result after performing linear interpolation. Null are ignored in this calculation.</td>
</tr>
<tr>
<td></td>
<td>timestamptz,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>interval, float</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>PERCENTILE_CONT(expr)</code></td>
<td>tween, float</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WITHIN GROUP (ORDER BY expr [DESC/ASC])</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Return Type</td>
<td>Full Syntax</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PERCENTILE_DISC(expr)</td>
<td>timestamp</td>
<td>PERCENTILE_DISC(expr) WITHIN GROUP (ORDER BY expr)</td>
<td>Performs an inverse distribution function that assumes a discrete distribution model. It takes a percentile value and a sort specification. This returned value is an element from the set. Null are ignored in this calculation.</td>
</tr>
<tr>
<td></td>
<td>timestamp</td>
<td>PERCENTILE_DISC(percentage) WITHIN GROUP (ORDER BY expression)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>interval</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>float</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SELECT department_id, PERCENTILE_DISC (0.5) WITHIN GROUP (ORDER BY salary DESC) &quot;Median_desc&quot; FROM employees GROUP BY department_id;</td>
</tr>
<tr>
<td>sum(array[])</td>
<td>smallint[]</td>
<td>sum([1,2],[3,4])</td>
<td>Performs matrix summation. Can take as input a two-dimensional array that is treated as a matrix.</td>
</tr>
<tr>
<td></td>
<td>bigint[]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>float[]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pivot_sum(label[], label, expr)</td>
<td>int[], bigint[], float[]</td>
<td>pivot_sum( array['A1','A2'], attr, value)</td>
<td>A pivot aggregation using sum to resolve duplicate entries.</td>
</tr>
<tr>
<td>unnest(array[])</td>
<td>set of anyelement</td>
<td>unnest( array['one', 'row', 'per', 'item'])</td>
<td>Transforms a one dimensional array into rows. Returns a set of anyelement, a polymorphic pseudotype in PostgreSQL.</td>
</tr>
</tbody>
</table>
Greenplum MapReduce Specification

This specification describes the document format and schema for defining Greenplum MapReduce jobs. MapReduce is a programming model developed by Google for processing and generating large data sets on an array of commodity servers. Greenplum MapReduce allows programmers who are familiar with the MapReduce model to write map and reduce functions and submit them to the Greenplum Database parallel engine for processing.

To enable Greenplum to process MapReduce functions, define the functions in a document, then pass the document to the Greenplum MapReduce program, `gpmapreduce`, for execution by the Greenplum Database parallel engine. The Greenplum Database system distributes the input data, executes the program across a set of machines, handles machine failures, and manages the required inter-machine communication.

See the Greenplum Database Utility Guide for information about `gpmapreduce`.

Greenplum MapReduce Document Format

This section explains some basics of the Greenplum MapReduce document format to help you get started creating your own Greenplum MapReduce documents. Greenplum uses the YAML 1.1 document format and then implements its own schema for defining the various steps of a MapReduce job.

All Greenplum MapReduce files must first declare the version of the YAML specification they are using. After that, three dashes (---) denote the start of a document, and three dots (... ) indicate the end of a document without starting a new one. Comment lines are prefixed with a pound symbol (#). It is possible to declare multiple Greenplum MapReduce documents in the same file:

```
---
# Begin Document 1
# ...
---
# Begin Document 2
# ...
```

Within a Greenplum MapReduce document, there are three basic types of data structures or nodes: scalars, sequences and mappings.

A scalar is a basic string of text indented by a space. If you have a scalar input that spans multiple lines, a preceding pipe ( | ) denotes a literal style, where all line breaks are significant. Alternatively, a preceding angle bracket ( > ) folds a single line break to a space for subsequent lines that have the same indentation level. If a string contains characters that have reserved meaning, the string must be quoted or the special character must be escaped with a backslash ( \ ).

```
# 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"
```

A sequence is a list with each entry in the list on its own line denoted by a dash and a space ( - ). Alternatively, you can specify an inline sequence as a comma-separated list within square brackets.
A sequence provides a set of data and gives it an order. When you load a list into the Greenplum MapReduce program, the order is kept.

```plaintext
# list sequence
- this
- is
- a list
- with
- five scalar values
# inline sequence
[this, is, a list, with, five scalar values]
```

A **mapping** is used to pair up data values with identifiers called **keys**. Mappings use a colon and space (:) for each key: value pair, or can also be specified inline as a comma-separated list within curly braces. The **key** is used as an index for retrieving data from a mapping.

```plaintext
# 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}
```

Keys are used to associate meta information with each node and specify the expected node type (**scalar**, **sequence** or **mapping**). See [Greenplum MapReduce Document Schema](#) for the keys expected by the Greenplum MapReduce program.

The Greenplum MapReduce program processes the nodes of a document in order and uses indentation (spaces) to determine the document hierarchy and the relationships of the nodes to one another. The use of white space is significant. White space should not be used simply for formatting purposes, and tabs should not be used at all.

---

**Greenplum MapReduce Document Schema**

Greenplum MapReduce uses the YAML document framework and implements its own YAML schema. The basic structure of a Greenplum MapReduce document is:

```yaml
%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 |

| RUN |
| SOURCE: input_or_task_name |
| TARGET: output_name |
| MAP: map_function_name |
| REDUCE: reduce_function_name... |
VERSION
Required. The version of the Greenplum MapReduce YAML specification. Current versions are 1.0.0.1.

DATABASE
Optional. Specifies which database in Greenplum to connect to. If not specified, defaults to the default database or $PGDATABASE if set.

USER
Optional. Specifies which database role to use to connect. If not specified, defaults to the current user or $PGUSER if set. You must be a Greenplum superuser to run functions written in untrusted Python and Perl. Regular database users can run functions written in trusted Perl. You also must be a database superuser to run MapReduce jobs that contain FILE, GPFDIST and EXEC input types.

HOST
Optional. Specifies Greenplum master host name. If not specified, defaults to localhost or $PGHOST if set.

PORT
Optional. Specifies Greenplum master port. If not specified, defaults to 5432 or $PGPORT if set.

DEFINE
Required. A sequence of definitions for this MapReduce document. The DEFINE section must have at least one INPUT definition.

INPUT
Required. Defines the input data. Every MapReduce document must have at least one input defined. Multiple input definitions are allowed in a document, but each input definition can specify only one of these access types: a file, a gpfdist file distribution program, a table in the database, an SQL command, or an operating system command. See the Greenplum Database Utility Guide for information about gpfdist.

NAME
A name for this input. Names must be unique with regards to the names of other objects in this MapReduce job (such as map function, task, reduce function and output names). Also, names cannot conflict with existing objects in the database (such as tables, functions or views).

FILE
A sequence of one or more input files in the format: seghostname:/path/to/filename. You must be a Greenplum Database superuser to run MapReduce jobs with FILE input. The file must reside on a Greenplum segment host.

GPFDIST
A sequence of one or more running gpfdist file distribution programs in the format: hostname[:port]/file_pattern. You must be a Greenplum Database superuser to run MapReduce jobs with GPFDIST input, unless the server configuration parameter Server Configuration Parameters is set to on.

TABLE
The name of an existing table in the database.

QUERY
A SQL SELECT command to run within the database.

EXEC
An operating system command to run on the Greenplum segment hosts. The command is run by all segment instances in the system by default. For example, if you have four
segment instances per segment host, the command will be run four times on each host.
You must be a Greenplum Database superuser to run MapReduce jobs with \texttt{EXEC} input
and the server configuration parameter \texttt{Server Configuration Parameters} is set to \texttt{on}.

**COLUMNS**

Optional. Columns are specified as: \texttt{column_name [data_type]}. If not specified, the
default is \texttt{value text}. The \texttt{DELIMITER} character is what separates two data value fields
(columns). A row is determined by a line feed character (0x0a).

**FORMAT**

Optional. Specifies the format of the data - either delimited text (TEXT) or comma
separated values (CSV) format. If the data format is not specified, defaults to \texttt{TEXT}.

**DELIMITER**

Optional for \texttt{FILE, GPFDIST} and \texttt{EXEC} inputs. Specifies a single character that separates
data values. The default is a tab character in TEXT mode, a comma in CSV mode. The
delimiter character must only appear between any two data value fields. Do not place a
delimiter at the beginning or end of a row.

**ESCAPE**

Optional for \texttt{FILE, GPFDIST} and \texttt{EXEC} inputs. Specifies the single character that is
used for C escape sequences (such as \texttt{\n, \t, \100}, and so on) and for escaping data
characters that might otherwise be taken as row or column delimiters. Make sure to
choose an escape character that is not used anywhere in your actual column data. The
default escape character is a \texttt{(backslash)} for text-formatted files and a " (double quote)
for csv-formatted files, however it is possible to specify another character to represent
an escape. It is also possible to disable escaping by specifying the value \texttt{'OFF'} as the
escape value. This is very useful for data such as text-formatted web log data that has
many embedded backslashes that are not intended to be escapes.

**NULL**

Optional for \texttt{FILE, GPFDIST} and \texttt{EXEC} inputs. Specifies the string that represents a null
value. The default is \texttt{\n} in TEXT format, and an empty value with no quotations in CSV
format. You might prefer an empty string even in TEXT mode for cases where you do not
want to distinguish nulls from empty strings. Any input data item that matches this string
will be considered a null value.

**QUOTE**

Optional for \texttt{FILE, GPFDIST} and \texttt{EXEC} inputs. Specifies the quotation character for CSV
formatted files. The default is a double quote ("). In CSV formatted files, data value fields
must be enclosed in double quotes if they contain any commas or embedded new lines.
Fields that contain double quote characters must be surrounded by double quotes, and
the embedded double quotes must each be represented by a pair of consecutive double
quotes. It is important to always open and close quotes correctly in order for data rows to
be parsed correctly.

**ERROR\_LIMIT**

If the input rows have format errors they will be discarded provided that the error limit count
is not reached on any Greenplum segment instance during input processing. If the error
limit is not reached, all good rows will be processed and any error rows discarded.

**ENCODING**

Character set encoding to use for the data. Specify a string constant (such as
\texttt{`SQL\_ASCII`}), an integer encoding number, or \texttt{DEFAULT} to use the default client
encoding. See \texttt{Character Set Support} for more information.

**OUTPUT**
Optional. Defines where to output the formatted data of this MapReduce job. If output is not defined, the default is \texttt{STDOUT} (standard output of the client). You can send output to a file on the client host or to an existing table in the database.

**NAME**

A name for this output. The default output name is \texttt{STDOUT}. Names must be unique with regards to the names of other objects in this MapReduce job (such as map function, task, reduce function and input names). Also, names cannot conflict with existing objects in the database (such as tables, functions or views).

**FILE**

Specifies a file location on the MapReduce client machine to output data in the format: /path/to/filename.

**TABLE**

Specifies the name of a table in the database to output data. If this table does not exist prior to running the MapReduce job, it will be created using the distribution policy specified with \texttt{KEYS}.

**KEYS**

Optional for \texttt{TABLE} output. Specifies the column(s) to use as the Greenplum Database distribution key. If the \texttt{EXECUTE} task contains a \texttt{REDUCE} definition, then the \texttt{REDUCE} keys will be used as the table distribution key by default. Otherwise, the first column of the table will be used as the distribution key.

**MODE**

Optional for \texttt{TABLE} output. If not specified, the default is to create the table if it does not already exist, but error out if it does exist. Declaring \texttt{APPEND} adds output data to an existing table (provided the table schema matches the output format) without removing any existing data. Declaring \texttt{REPLACE} will drop the table if it exists and then recreate it. Both \texttt{APPEND} and \texttt{REPLACE} will create a new table if one does not exist.

**MAP**

Required. Each \texttt{MAP} function takes data structured in \texttt{(key, value)} pairs, processes each pair, and generates zero or more output \texttt{(key, value)} pairs. The Greenplum MapReduce framework then collects all pairs with the same key from all output lists and groups them together. This output is then passed to the \texttt{REDUCE} task, which is comprised of \texttt{TRANSITION | CONSOLIDATE | FINALIZE} functions. There is one predefined \texttt{MAP} function named \texttt{IDENTITY} that returns \texttt{(key, value)} pairs unchanged. Although \texttt{(key, value)} are the default parameters, you can specify other prototypes as needed.

**TRANSITION | CONSOLIDATE | FINALIZE**

\texttt{TRANSITION, CONSOLIDATE and FINALIZE} are all component pieces of \texttt{REDUCE}. A \texttt{TRANSITION} function is required. \texttt{CONSOLIDATE} and \texttt{FINALIZE} functions are optional. By default, all take \texttt{state} as the first of their input \texttt{PARAMETERS}, but other prototypes can be defined as well.

A \texttt{TRANSITION} function iterates through each value of a given key and accumulates values in a \texttt{state} variable. When the transition function is called on the first value of a key, the \texttt{state} is set to the value specified by \texttt{INITIALIZE} of a \texttt{REDUCE} job (or the default state value for the data type). A transition takes two arguments as input; the current state of the key reduction, and the next value, which then produces a new \texttt{state}.

If a \texttt{CONSOLIDATE} function is specified, \texttt{TRANSITION} processing is performed at the segment-level before redistributing the keys across the Greenplum interconnect for final aggregation (two-phase aggregation). Only the resulting \texttt{state} value for a given key is redistributed, resulting in lower interconnect traffic and greater parallelism. \texttt{CONSOLIDATE} is handled like a \texttt{TRANSITION}, except that instead of \texttt{(state + value) => state}, it is \texttt{(state + state) => state}.
If a `FINALIZE` function is specified, it takes the final state produced by `CONSOLIDATE` (if present) or `TRANSITION` and does any final processing before emitting the final result. `TRANSITION` and `CONSOLIDATE` functions cannot return a set of values. If you need a `REDUCE` job to return a set, then a `FINALIZE` is necessary to transform the final state into a set of output values.

**NAME**

Required. A name for the function. Names must be unique with regards to the names of other objects in this MapReduce job (such as function, task, input and output names). You can also specify the name of a function built-in to Greenplum Database. If using a built-in function, do not supply `LANGUAGE` or a `FUNCTION` body.

**FUNCTION**

Optional. Specifies the full body of the function using the specified `LANGUAGE`. If `FUNCTION` is not specified, then a built-in database function corresponding to `NAME` is used.

**LANGUAGE**

Required when `FUNCTION` is used. Specifies the implementation language used to interpret the function. This release has language support for perl, python, and C. If calling a built-in database function, `LANGUAGE` should not be specified.

**LIBRARY**

Required when `LANGUAGE` is C (not allowed for other language functions). To use this attribute, `VERSION` must be 1.0.0.2. The specified library file must be installed prior to running the MapReduce job, and it must exist in the same file system location on all Greenplum hosts (master and segments).

**PARAMETERS**

Optional. Function input parameters. The default type is `text`.

- `MAP` default - `key text, value text`
- `TRANSITION` default - `state text, value text`
- `CONSOLIDATE` default - `state1 text, state2 text` (must have exactly two input parameters of the same data type)
- `FINALIZE` default - `state text` (single parameter only)

**RETURNS**

Optional. The default return type is `text`.

- `MAP` default - `key text, value text`
- `TRANSITION` default - `state text` (single return value only)
- `CONSOLIDATE` default - `state text` (single return value only)
- `FINALIZE` default - `value text`

**OPTIMIZE**

Optional optimization parameters for the function:

- `STRICT` - function is not affected by NULL values
- `IMMUTABLE` - function will always return the same value for a given input

**MODE**

Optional. Specifies the number of rows returned by the function.

- `MULTI` - returns 0 or more rows per input record. The return value of the function must be an array of rows to return, or the function must be written as an iterator using `yield`
in Python or return_next in Perl. MULTI is the default mode for MAP and FINALIZE functions.

SINGLE - returns exactly one row per input record. SINGLE is the only mode supported for TRANSITION and CONSOLIDATE functions. When used with MAP and FINALIZE functions, SINGLE mode can provide modest performance improvement.

**REDUCE**

Required. A REDUCE definition names the TRANSITION | CONSOLIDATE | FINALIZE functions that comprise the reduction of (key, value) pairs to the final result set. There are also several predefined REDUCE jobs you can execute, which all operate over a column named value:

- **IDENTITY** - returns (key, value) pairs unchanged
- **SUM** - calculates the sum of numeric data
- **AVG** - calculates the average of numeric data
- **COUNT** - calculates the count of input data
- **MIN** - calculates minimum value of numeric data
- **MAX** - calculates maximum value of numeric data

**NAME**

Required. The name of this REDUCE job. Names must be unique with regards to the names of other objects in this MapReduce job (function, task, input and output names). Also, names cannot conflict with existing objects in the database (such as tables, functions or views).

**TRANSITION**

Required. The name of the TRANSITION function.

**CONSOLIDATE**

Optional. The name of the CONSOLIDATE function.

**FINALIZE**

Optional. The name of the FINALIZE function.

**INITIALIZE**

Optional for text and float data types. Required for all other data types. The default value for text is ''. The default value for float is 0.0. Sets the initial state value of the TRANSITION function.

**KEYS**

Optional. Defaults to [key, *]. When using a multi-column reduce it may be necessary to specify which columns are key columns and which columns are value columns. By default, any input columns that are not passed to the TRANSITION function are key columns, and a column named key is always a key column even if it is passed to the TRANSITION function. The special indicator * indicates all columns not passed to the TRANSITION function. If this indicator is not present in the list of keys then any unmatched columns are discarded.

**TASK**

Optional. A TASK defines a complete end-to-end INPUT/Map/REDUCE stage within a Greenplum MapReduce job pipeline. It is similar to EXECUTE except it is not immediately executed. A task object can be called as INPUT to further processing stages.

**NAME**

Required. The name of this task. Names must be unique with regards to the names of other objects in this MapReduce job (such as map function, reduce function, input and
output names). Also, names cannot conflict with existing objects in the database (such as tables, functions or views).

### SOURCE
The name of an **INPUT** or another **TASK**.

### MAP
Optional. The name of a **MAP** function. If not specified, defaults to **IDENTITY**.

### REDUCE
Optional. The name of a **REDUCE** function. If not specified, defaults to **IDENTITY**.

### EXECUTE
Required. **EXECUTE** defines the final **INPUT/MAP/REDUCE** stage within a Greenplum MapReduce job pipeline.

### RUN

#### SOURCE
Required. The name of an **INPUT** or **TASK**.

#### TARGET
Optional. The name of an **OUTPUT**. The default output is **STDOUT**.

#### MAP
Optional. The name of a **MAP** function. If not specified, defaults to **IDENTITY**.

#### REDUCE
Optional. The name of a **REDUCE** function. Defaults to **IDENTITY**.

---

### Example Greenplum MapReduce Document

```yaml
%YAML 1.1
---

DATABASE:webdata
USER:jsmith

DEFINE:
  - 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
    # 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
    # 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

Flow Diagram for MapReduce Example
The following diagram shows the job flow of the MapReduce job defined in the example:
Greenplum PL/pgSQL Procedural Language

This section contains an overview of the Greenplum Database PL/pgSQL language.

- About Greenplum Database PL/pgSQL
- PL/pgSQL Plan Caching
- PL/pgSQL Examples
- References

About Greenplum Database PL/pgSQL

Greenplum Database PL/pgSQL is a loadable procedural language that is installed and registered by default with Greenplum Database. You can create user-defined functions using SQL statements, functions, and operators.

With PL/pgSQL you can group a block of computation and a series of SQL queries inside the database server, thus having the power of a procedural language and the ease of use of SQL. Also, with PL/pgSQL you can use all the data types, operators and functions of Greenplum Database SQL.

The PL/pgSQL language is a subset of Oracle PL/SQL. Greenplum Database PL/pgSQL is based on Postgres PL/pgSQL. The Postgres PL/pgSQL documentation is at https://www.postgresql.org/docs/8.3/static/plpgsql.html

When using PL/pgSQL functions, function attributes affect how Greenplum Database creates query plans. You can specify the attribute IMMUTABLE, STABLE, or VOLATILE as part of the LANGUAGE clause to classify the type of function. For information about the creating functions and function attributes, see the CREATE FUNCTION command in the Greenplum Database Reference Guide.

You can run PL/SQL code blocks as anonymous code blocks. See the DO command in the Greenplum Database Reference Guide.

Greenplum Database SQL Limitations

When using Greenplum Database PL/pgSQL, limitations include

- Triggers are not supported
- Cursors are forward moving only (not scrollable)
- Updatable cursors (UPDATE...WHERE CURRENT OF and DELETE...WHERE CURRENT OF) are not supported.

For information about Greenplum Database SQL conformance, see Summary of Greenplum Features in the Greenplum Database Reference Guide.

The PL/pgSQL Language

PL/pgSQL is a block-structured language. The complete text of a function definition must be a block. A block is defined as:

```sql
[ label ]
[ DECLARE
   declarations ]
BEGIN
   statements
END [ label ];
```
Each declaration and each statement within a block is terminated by a semicolon (;). A block that appears within another block must have a semicolon after END, as shown in the previous block. The END that concludes a function body does not require a semicolon.

**Important:** Do not confuse the use of the BEGIN and END keywords for grouping statements in PL/pgSQL with the database commands for transaction control. The PL/pgSQL BEGIN and END keywords are only for grouping; they do not start or end a transaction. Functions are always executed within a transaction established by an outer query — they cannot start or commit that transaction, since there would be no context for them to execute in. However, a PL/pgSQL block that contains an EXCEPTION clause effectively forms a subtransaction that can be rolled back without affecting the outer transaction. For more about the EXCEPTION clause, see the post the Postgres documentation on error trapping at https://www.postgresql.org/docs/8.3/static/plpgsql-control-structures.html#PLPGSQL-ERROR-TRAPPING.

All key words and identifiers can be written in mixed upper and lower case. Identifiers are implicitly converted to lowercase unless enclosed in double-quotes (" ).

You can add comments in PL/pgSQL in the following ways:

- A double dash (--) starts a comment that extends to the end of the line.
- A /* starts a block comment that extends to the next occurrence of */.

Block comments cannot be nested, but double dash comments can be enclosed into a block comment and a double dash can hide the block comment delimiters /* and */.

Any statement in the statement section of a block can be a subblock. Subblocks can be used for logical grouping or to localize variables to a small group of statements.

The variables declared in the declarations section preceding a block are initialized to their default values every time the block is entered, not only once per function call. For example declares the variable quantity several times:

```sql
CREATE FUNCTION testfunc() RETURNS integer AS $$
DECLARE
    quantity integer := 30;
BEGIN
    RAISE NOTICE 'Quantity here is %', quantity;
    -- Quantity here is 30
    quantity := 50;
    --
    -- Create a subblock
    --
    DECLARE
        quantity integer := 80;
    BEGIN
        RAISE NOTICE 'Quantity here is %', quantity;
        -- Quantity here is 80
        END;
    RAISE NOTICE 'Quantity here is %', quantity;
    -- Quantity here is 50
    RETURN quantity;
END;
$$ LANGUAGE plpgsql;
```

**Executing SQL Commands**

You can execute SQL commands with PL/pgSQL statements such as EXECUTE, PERFORM, and SELECT ... INTO. For information about the PL/pgSQL statements, see https://www.postgresql.org/docs/8.3/static/plpgsql-statements.html.

**Note:** The PL/pgSQL statement SELECT INTO is not supported in the EXECUTE statement.
PL/pgSQL Plan Caching

A PL/pgSQL function’s volatility classification has implications on how Greenplum Database caches plans that reference the function. Refer to Function Volatility and Plan Caching in the Greenplum Database Administrator Guide for information on plan caching considerations for Greenplum Database function volatility categories.

When a PL/pgSQL function executes for the first time in a database session, the PL/pgSQL interpreter parses the function’s SQL expressions and commands. The interpreter creates a prepared execution plan as each expression and SQL command is first executed in the function. The PL/pgSQL interpreter reuses the execution plan for a specific expression and SQL command for the life of the database connection. While this reuse substantially reduces the total amount of time required to parse and generate plans, errors in a specific expression or command cannot be detected until run time when that part of the function is executed.

Greenplum Database will automatically re-plan a saved query plan if there is any schema change to any relation used in the query, or if any user-defined function used in the query is redefined. This makes the re-use of a prepared plan transparent in most cases.

The SQL commands that you use in a PL/pgSQL function must refer to the same tables and columns on every execution. You cannot use a parameter as the name of a table or a column in an SQL command.

PL/pgSQL caches a separate query plan for each combination of actual argument types in which you invoke a polymorphic function to ensure that data type differences do not cause unexpected failures.

Refer to the PostgreSQL Plan Caching documentation for a detailed discussion of plan caching considerations in the PL/pgSQL language.

PL/pgSQL Examples

The following are examples of PL/pgSQL user-defined functions.

Example: Aliases for Function Parameters

Parameters passed to functions are named with identifiers such as $1, $2. Optionally, aliases can be declared for $n parameter names for increased readability. Either the alias or the numeric identifier can then be used to refer to the parameter value.

There are two ways to create an alias. The preferred way is to give a name to the parameter in the CREATE FUNCTION command, for example:

```
CREATE FUNCTION sales_tax(subtotal real) RETURNS real AS $$
BEGIN
    RETURN subtotal * 0.06;
END;
$$ LANGUAGE plpgsql;
```

You can also explicitly declare an alias, using the declaration syntax:

```
name ALIAS FOR $n;
```

This example, creates the same function with the DECLARE syntax:

```
CREATE FUNCTION sales_tax(real) RETURNS real AS $$
DECLARE
    subtotal ALIAS FOR $1;
BEGIN
    RETURN subtotal * 0.06;
END;
```
Example: Using the Data Type of a Table Column

When declaring a variable, you can use %TYPE to specify the data type of a variable or table column. This is the syntax for declaring a variable with the data type of a table column:

```plaintext
name table.column_name%TYPE;
```

You can use this to declare variables that will hold database values. For example, if you have a column named `user_id` in your `users` table. To declare the variable `my_userid` with the same data type as the `users.user_id` column:

```plaintext
my_userid users.user_id%TYPE;
```

%TYPE is particularly valuable in polymorphic functions, since the data types needed for internal variables may change from one call to the next. Appropriate variables can be created by applying %TYPE to the function's arguments or result placeholders.

Example: Composite Type Based on a Table Row

The following syntax declares a composite variable based on table row:

```plaintext
name table_name%ROWTYPE;
```

Such a row variable can hold a whole row of a SELECT or FOR query result, so long as that query column set matches the declared type of the variable. The individual fields of the row value are accessed using the usual dot notation, for example `rowvar.column`.

Parameters to a function can be composite types (complete table rows). In that case, the corresponding identifier $n will be a row variable, and fields can be selected from it, for example `$1.user_id`.

Only the user-defined columns of a table row are accessible in a row-type variable, not the OID or other system columns. The fields of the row type inherit the table's field size or precision for data types such as `char(n)`.

The next example function uses a row variable composite type. Before creating the function, create the table that is used by the function with this command.

```sql
CREATE TABLE table1 (  
    f1 text,  
    f2 numeric,  
    f3 integer  
) distributed by (f1);
```

This INSERT command adds data to the table.

```sql
INSERT INTO table1 values  
    ('test1', 14.1, 3),  
    ('test2', 52.5, 2),  
    ('test3', 32.22, 6),  
    ('test4', 12.1, 4) ;
```

This function uses a variable and ROWTYPE composite variable based on `table1`.

```sql
CREATE OR REPLACE FUNCTION t1_calc( name text) RETURNS integer  
AS $$
DECLARE
    tl_row   table1%ROWTYPE;
$$
```

### Greenplum Database Reference Guide

**Release Notes**

---

**Example: Composite Type Based on a Table Row**

The following syntax declares a composite variable based on table row:

```plaintext
name table_name%ROWTYPE;
```

Such a row variable can hold a whole row of a SELECT or FOR query result, so long as that query column set matches the declared type of the variable. The individual fields of the row value are accessed using the usual dot notation, for example `rowvar.column`.

Parameters to a function can be composite types (complete table rows). In that case, the corresponding identifier $n will be a row variable, and fields can be selected from it, for example `$1.user_id`.

Only the user-defined columns of a table row are accessible in a row-type variable, not the OID or other system columns. The fields of the row type inherit the table’s field size or precision for data types such as `char(n)`.

The next example function uses a row variable composite type. Before creating the function, create the table that is used by the function with this command.

```sql
CREATE TABLE table1 (  
    f1 text,  
    f2 numeric,  
    f3 integer  
) distributed by (f1);
```

This INSERT command adds data to the table.

```sql
INSERT INTO table1 values  
    ('test1', 14.1, 3),  
    ('test2', 52.5, 2),  
    ('test3', 32.22, 6),  
    ('test4', 12.1, 4) ;
```

This function uses a variable and ROWTYPE composite variable based on `table1`.

```sql
CREATE OR REPLACE FUNCTION t1_calc( name text) RETURNS integer  
AS $$
DECLARE
    tl_row   table1%ROWTYPE;
$$
```
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: The previous function is classified as a VOLATILE function because function values could change within a single table scan.

The following SELECT command uses the function.

select tl_calc( 'test1' );

Note: The example PL/pgSQL function uses SELECT with the INTO clause. It is different from the SQL command SELECT INTO. If you want to create a table from a SELECT result inside a PL/pgSQL function, use the SQL command CREATE TABLE AS.

Example: Using a Variable Number of Arguments

You can declare a PL/pgSQL function to accept variable numbers of arguments, as long as all of the optional arguments are of the same data type. You must mark the last argument of the function as VARIADIC and declare the argument using an array type. You can refer to a function that includes VARIADIC arguments as a variadic function.

For example, this variadic function returns the minimum value of a variable array of numerics:

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)

Effectively, all of the actual arguments at or beyond the VARIADIC position are gathered up into a one-dimensional array.

You can pass an already-constructed array into a variadic function. This is particularly useful when you want to pass arrays between variadic functions. Specify VARIADIC in the function call as follows:

SELECT mleast(VARIADIC ARRAY[10, -1, 5, 4.4]);

This prevents PL/pgSQL from expanding the function's variadic parameter into its element type.

Example: Using Default Argument Values

You can declare PL/pgSQL functions with default values for some or all input arguments. The default values are inserted whenever the function is called with fewer than the declared number of arguments. Because arguments can only be omitted from the end of the actual argument list, you must provide default values for all arguments after an argument defined with a default value.
For example:

```sql
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)
```

You can also use the = sign in place of the keyword DEFAULT.

**Example: Using Polymorphic Data Types**

PL/pgSQL supports the polymorphic `anyelement`, `anyarray`, `anyenum`, and `anynonarray` types. Using these types, you can create a single PL/pgSQL function that operates on multiple data types. Refer to Greenplum Database Data Types for additional information on polymorphic type support in Greenplum Database.

A special parameter named $0 is created when the return type of a PL/pgSQL function is declared as a polymorphic type. The data type of $0 identifies the return type of the function as deduced from the actual input types.

In this example, you create a polymorphic function that returns the sum of two values:

```sql
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;

Execute add_two_values() providing integer input values:

```sql
SELECT add_two_values(1, 2);
add_two_values
------------
3
(1 row)
```
The return type of `add_two_values()` is integer, the type of the input arguments. Now execute `add_two_values()` providing float input values:

```sql
SELECT add_two_values (1.1, 2.2);
```

```
add_two_values
---------------
  3.3
(1 row)
```

The return type of `add_two_values()` in this case is float.

You can also specify `VARIADIC` arguments in polymorphic functions.

**Example: Anonymous Block**

This example executes the function in the previous example as an anonymous block with the `DO` command. In the example, the anonymous block retrieves the input value from a temporary table.

```sql
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 ;
```

**References**


Also, see the `CREATE FUNCTION` command in the Greenplum Database Reference Guide.

For a summary of built-in Greenplum Database functions, see Summary of Built-in Functions in the Greenplum Database Reference Guide. For information about using Greenplum Database functions see "Querying Data" in the Greenplum Database Administrator Guide.

Greenplum PostGIS Extension

This chapter contains the following information:

- About PostGIS
- Greenplum PostGIS Extension
- Enabling and Removing PostGIS Support
- Usage
- PostGIS Extension Support and Limitations
- PostGIS Support Scripts

About PostGIS

PostGIS is a spatial database extension for PostgreSQL that allows GIS (Geographic Information Systems) objects to be stored in the database. The Greenplum Database PostGIS extension includes support for GiST-based R-Tree spatial indexes and functions for analysis and processing of GIS objects.

The Greenplum Database PostGIS extension supports the optional PostGIS raster data type and most PostGIS Raster functions. With the PostGIS Raster objects, PostGIS geometry data type offers a single set of overlay SQL functions (such as ST.Intersects) operating seamlessly on vector and raster geospatial data. PostGIS Raster uses the GDAL (Geospatial Data Abstraction Library) translator library for raster geospatial data formats that presents a single raster abstract data model to a calling application.

For information about Greenplum Database PostGIS extension support, see PostGIS Extension Support and Limitations.

For information about PostGIS, see http://postgis.refractions.net/

For information about GDAL, see http://www.gdal.org.

Greenplum PostGIS Extension

The Greenplum Database PostGIS extension package is available from Pivotal Network. You can install the package using the Greenplum Package Manager (gppkg). For details, see gppkg in the Greenplum Database Utility Guide.

Greenplum Database supports the PostGIS extension with these component versions.

- PostGIS 2.1.5
- Proj 4.8.0
- Geos 3.4.2
- GDAL 1.11.1
- Json 0.12
- Expat 2.1.0

For the information about supported extension packages and software versions see the Greenplum Database Release Notes.

Major enhancements and changes in PostGIS 2.1.5 from 2.0.3 include new PostGIS Raster functions. For a list of new and enhanced functions in PostGIS 2.1, see the PostGIS documentation PostGIS Functions new or enhanced in 2.1.

For a list of breaking changes in PostGIS 2.1, see PostGIS functions breaking changes in 2.1.

For a comprehensive list of PostGIS changes in PostGIS 2.1.5 and earlier, see PostGIS 2.1 Appendix A Release 2.1.5.
**Greenplum Database PostGIS Limitations**

The Greenplum Database PostGIS extension does not support the following features:

- Topology
- A small number of user defined functions and aggregates
- PostGIS long transaction support
- Geometry and geography type modifier

For information about Greenplum Database PostGIS support, see [PostGIS Extension Support and Limitations](#).

**Enabling and Removing PostGIS Support**

The Greenplum Database PostGIS extension contains the `postgis_manager.sh` script that installs or removes both the PostGIS and PostGIS Raster features in a database. After the PostGIS extension package is installed, the script is in `$GPHOME/share/postgresql/contrib/postgis-2.1/`. The `postgis_manager.sh` script runs SQL scripts that install or remove PostGIS and PostGIS Raster from a database.

For information about the PostGIS and PostGIS Raster SQL scripts, and required PostGIS Raster environment variables, see [PostGIS Support Scripts](#).

**Enabling PostGIS Support**

Run the `postgis_manager.sh` script specifying the database and with the `install` option to install PostGIS and PostGIS Raster. This example installs PostGIS and PostGIS Raster objects in the database `mydatabase`.

```bash
postgis_manager.sh mydatabase install
```

The script runs all the PostGIS SQL scripts that enable PostGIS in a database: `install/postgis.sql`, `install/rtpostgis.sql`, `install/spatial_ref_sys.sql`, `install/postgis_comments.sql`, and `install/raster_comments.sql`.

The PostGIS package installation adds these lines to the `greenplum_path.sh` file for PostGIS Raster support.

```bash
export GDAL_DATA=$GPHOME/share/gdal
export POSTGIS_ENABLE_OUTDB_RASTERS=0
export POSTGIS_GDAL_ENABLED_DRIVERS=ENABLE_ALL
```

**Removing PostGIS Support**

Run the `postgis_manager.sh` script specifying the database and with the `uninstall` option to remove PostGIS and PostGIS Raster. This example removes PostGIS and PostGIS Raster support from the database `mydatabase`.

```bash
postgis_manager.sh mydatabase uninstall
```

The script runs both the PostGIS SQL scripts that remove PostGIS and PostGIS Raster from a database: `uninstall_rtpostgis.sql` and `uninstall_postgis.sql`.

The `postgis_manager.sh` script does not remove these PostGIS Raster environment variables the `greenplum_path.sh` file: GDAL DATA, POSTGIS_ENABLE_OUTDB_RASTERS, POSTGIS_GDAL_ENABLED_DRIVERS. The environment variables are removed when you uninstall the PostGIS extension package with the `gppkg` utility.
Usage

The following example SQL statements create non-OpenGIS tables and geometries.

```
CREATE TABLE geom_test ( gid int4, geom geometry,
    name varchar(25) );
INSERT INTO geom_test ( gid, geom, name )
    VALUES ( 1, 'POLYGON((0 0 0 0,0 5 0,5 5 0,5 0 0,0 0 0))', '3D Square');
INSERT INTO geom_test ( gid, geom, name )
    VALUES ( 2, 'LINESTRING(1 1,5 5,7 7,5 5)', '3D Line' );
INSERT INTO geom_test ( gid, geom, name )
    VALUES ( 3, 'MULTIPOINT(3 4,9)', '2D Aggregate Point' );
SELECT * from geom_test WHERE geom &&
    Box3D(ST_GeomFromEWKT('LINESTRING(2 2 0, 3 3 0)'));
```

The following example SQL statements create a table and add a geometry column to the table with a SRID integer value that references an entry in the `SPATIAL_REF_SYS` table. The `INSERT` statements add two geopoints to the table.

```
CREATE TABLE geotest (id INT4, name VARCHAR(32) );
SELECT AddGeometryColumn('geotest','geopoint', 4326,'POINT',2);
INSERT INTO geotest (id, name, geopoint)
    VALUES (1, 'Olympia', ST_GeometryFromText('POINT(-122.90 46.97)', 4326));
INSERT INTO geotest (id, name, geopoint)
    VALUES (2, 'Renton', ST_GeometryFromText('POINT(-122.22 47.50)', 4326));
SELECT name,ST_AsText(geopoint) FROM geotest;
```

Spatial Indexes

PostgreSQL provides support for GiST spatial indexing. The GiST scheme offers indexing even on large objects. It uses a system of lossy indexing in which smaller objects act as proxies for larger ones in the index. In the PostGIS indexing system, all objects use their bounding boxes as proxies in the index.

Building a Spatial Index

You can build a GiST index as follows:

```
CREATE INDEX indexname
    ON tablename
    USING GIST ( geometryfield );
```

PostGIS Extension Support and Limitations

This section describes Greenplum PostGIS extension feature support and limitations.

- Supported PostGIS Data Types
- Supported PostGIS Index
- Supported PostGIS Raster Data Types
- PostGIS Extension Limitations

The Greenplum Database PostGIS extension does not support the following features:

- Topology
- Some Raster Functions

Supported PostGIS Data Types

Greenplum Database PostGIS extension supports these PostGIS data types:
For a list of PostGIS data types, operators, and functions, see the PostGIS reference documentation.

**Supported PostGIS Raster Data Types**

Greenplum Database PostGIS supports these PostGIS Raster data types.

- geomval
- addbandarg
- rastbandarg
- raster
- reclassarg
- summarystats
- unionarg


For a list of PostGIS Raster data types, operators, and functions, see the PostGIS Raster reference documentation.

**Supported PostGIS Index**

Greenplum Database PostGIS extension supports the GiST (Generalized Search Tree) index.

**PostGIS Extension Limitations**

This section lists the Greenplum Database PostGIS extension limitations for user-defined functions (UDFs), data types, and aggregates.

- Data types and functions related to PostGIS topology functionality, such as TopoGeometry, are not supported by Greenplum Database.
- Functions that perform ANALYZE operations for user-defined data types are not supported. For example, the ST_Estimated_Extent function is not supported. The function requires table column statistics for user defined data types that are not available with Greenplum Database.
- These PostGIS aggregates are not supported by Greenplum Database:
  - ST_MemCollect
  - ST_MakeLine

On a Greenplum Database with multiple segments, the aggregate might return different answers if it is called several times repeatedly.

- Greenplum Database does not support PostGIS long transactions.

PostGIS relies on triggers and the PostGIS table `public.authorization_table` for long transaction support. When PostGIS attempts to acquire locks for long transactions, Greenplum Database reports errors citing that the function cannot access the relation, `authorization_table`.

- Greenplum Database does not support type modifiers for user defined types.

The workaround is to use the `AddGeometryColumn` function for PostGIS geometry. For example, a table with PostGIS geometry cannot be created with the following SQL command:

```sql
CREATE TABLE geometries(id INTEGER, geom geometry(LINESTRING));
```
Use the `AddGeometryColumn` function to add PostGIS geometry to a table. For example, these following SQL statements create a table and add PostGIS geometry to the table:

```
CREATE TABLE geometries(id INTEGER);
SELECT AddGeometryColumn('public', 'geometries', 'geom', 0, 'LINESTRING', 2);
```

**PostGIS Support Scripts**

After installing the PostGIS extension package, you enable PostGIS support for each database that requires its use. To enable or remove PostGIS support in your database, you can run SQL scripts that are supplied with the PostGIS package in `$GPHOME/share/postgresql/contrib/postgis-2.1/`. 

- **Scripts that Enable PostGIS and PostGIS Raster Support**
- **Scripts that Remove PostGIS and PostGIS Raster Support**

Instead of running the scripts individually, you can use the `postgis_manager.sh` script to run SQL scripts that enable or remove PostGIS support. See [Enabling and Removing PostGIS Support](#).

You can run the PostGIS SQL scripts individually to enable or remove PostGIS support. For example, these commands run the SQL scripts `postgis.sql`, `rtpostgis.sql`, and `spatial_ref_sys.sql` in the database `mydatabase`.

```
psql -d mydatabase -f $GPHOME/share/postgresql/contrib/postgis-2.1/install/postgis.sql
psql -d mydatabase -f $GPHOME/share/postgresql/contrib/postgis-2.1/install/rtpostgis.sql
psql -d mydatabase -f $GPHOME/share/postgresql/contrib/postgis-2.1/install/spatial_ref_sys.sql
```

After running the scripts, the database is enabled with both PostGIS and PostGIS Raster.

**Scripts that Enable PostGIS and PostGIS Raster Support**

These scripts enable PostGIS, and the optional PostGIS Raster in a database.

- `install/postgis.sql` - Load the PostGIS objects and function definitions.
- `install/rtpostgis.sql` - Load the PostGIS raster object and function definitions.

  **Note:** If you are installing PostGIS Raster, PostGIS objects must be installed before PostGIS Raster. PostGIS Raster depends on PostGIS objects. Greenplum Database returns an error if `rtpostgis.sql` is run before `postgis.sql`.

These SQL scripts add data and comments to a PostGIS enabled database.

- `install/spatial_ref_sys.sql` - Populate the `spatial_ref_sys` table with a complete set of EPSG coordinate system definition identifiers. With the definition identifiers you can perform `ST_Transform()` operations on geometries.

  **Note:** If you have overridden standard entries and want to use those overrides, do not load the `spatial_ref_sys` file when creating the new database.

- `install/postgis_comments.sql` - Add comments to the PostGIS functions.
- `install/raster_comments.sql` - Add comments to the PostGIS Raster functions.

You can view comments with the `psql` meta-command `\dd function_name` or from any tool that can show Greenplum Database function comments.
**PostGIS Raster Environment Variables**

The postGIS package installation adds these lines to the `greenplum_path.sh` file for PostGIS Raster support.

```bash
export GDAL_DATA=$GPHOME/share/gdal
export POSTGIS_ENABLE_OUTDB_RASTERS=0
export POSTGIS_GDAL_ENABLED_DRIVERS=ENABLE_ALL
```

- **GDAL_DATA** specifies the location of GDAL utilities and support files used by the GDAL library. For example, the directory contains EPSG support files such as `gcs.csv` and `pcs.csv` (so called dictionaries, mostly in CSV format). The GDAL library requires the support files to properly evaluate EPSG codes.

- **POSTGIS_GDAL_ENABLED_DRIVERS** sets the enabled GDAL drivers in the PostGIS environment.

- **POSTGIS_ENABLE_OUTDB_RASTERS** is a boolean configuration option to enable access to out of database raster bands.

**Scripts that Remove PostGIS and PostGIS Raster Support**

To remove PostGIS support from a database, run SQL scripts that are supplied with the PostGIS extension package in `$GPHOME/share/postgresql/contrib/postgis-2.1/`

**Note:** If you installed PostGIS Raster, you must uninstall PostGIS Raster before you uninstall the PostGIS objects. PostGIS Raster depends on PostGIS objects. Greenplum Database returns an error if PostGIS objects are removed before PostGIS Raster.

These scripts remove PostGIS and PostGIS Raster objects from a database.

- `uninstall/uninstall_rtpostgis.sql` - Removes the PostGIS Raster object and function definitions.
- `uninstall/uninstall_postgis.sql` - Removes the PostGIS objects and function definitions.

After PostGIS support has been removed from all databases in the Greenplum Database system, you can remove the PostGIS extension package. For example this `gppkg` command removes the PostGIS extension package.

```
gppkg -r postgis-ossv2.1.5_pv2.1_gpdb5.0
```

Restart Greenplum Database after removing the package.

```
gpstop -r
```

Ensure that these lines for PostGIS Raster support are removed from the `greenplum_path.sh` file.

```bash
export GDAL_DATA=$GPHOME/share/gdal
export POSTGIS_ENABLE_OUTDB_RASTERS=0
export POSTGIS_GDAL_ENABLED_DRIVERS=ENABLE_ALL
```
Greenplum Database Reference Guide

Greenplum PL/R Language Extension

This chapter contains the following information:

- About Greenplum Database PL/R
- Installing PL/R
- Uninstalling PL/R
- Enabling PL/R Language Support
- Examples
- Downloading and Installing R Packages
- Displaying R Library Information
- References

About Greenplum Database PL/R

PL/R is a procedural language. With the Greenplum Database PL/R extension you can write database functions in the R programming language and use R packages that contain R functions and data sets.

For information about supported PL/R versions, see the Greenplum Database Release Notes.

Installing PL/R

The PL/R extension is available as a package. Download the package from Pivotal Network and install it with the Greenplum Package Manager (gppkg).

The gppkg utility installs Greenplum Database extensions, along with any dependencies, on all hosts across a cluster. It also automatically installs extensions on new hosts in the case of system expansion and segment recovery.

For information about gppkg, see the Greenplum Database Utility Guide.

Installing the Extension Package

Before you install the PL/R extension, make sure that your Greenplum Database is running, you have sourced greenplum_path.sh, and that the $MASTER_DATA_DIRECTORY and $GPHOME variables are set.

1. Download the PL/R extension package from Pivotal Network.
2. Copy the PL/R package to the Greenplum Database master host.
3. Install the software extension package by running the gppkg command. This example installs the PL/R extension on a Linux system:

   ```bash
   $ gppkg -i plr-2.3.1-gp5-rhel6-x86_64.gppkg
   ```
4. Restart the database.

   `$ gpstop -r`
5. Source the file $GPHOME/greenplum_path.sh.

The extension and the R environment is installed in this directory:

```
$GPHOME/ext/R-3.3.3/
```

Note: The version of some shared libraries installed with the operating system might not be compatible with the Greenplum Database PL/R extension.
If a shared library is not compatible, edit the file `$GPHOME/greenplum_path.sh` in all Greenplum Database master and segment hosts and set environment variable `LD_LIBRARY_PATH` to specify the location that is installed with the PL/R extension.

```bash
export LD_LIBRARY_PATH=
    $GPHOME/ext/R-3.3.3/lib:$LD_LIBRARY_PATH
```

### Uninstalling PL/R

- **Remove PL/R Support for a Database**
- **Uninstall the Extension Package**

When you remove PL/R language support from a database, the PL/R routines that you created in the database will no longer work.

#### Remove PL/R Support for a Database

For a database that no longer requires the PL/R language, remove support for PL/R with the SQL command `DROP LANGUAGE` or the Greenplum Database `droplang` utility. Because PL/R is an untrusted language, only superusers can remove support for the PL/R language from a database. For example, running this command as the `gpadmin` user removes support for PL/R from the database named `testdb`:

```
$ droplang plr -d testdb
```

#### Uninstall the Extension Package

If no databases have PL/R as a registered language, uninstall the Greenplum PL/R extension with the `gppkg` utility. This example uninstalls PL/R package version 2.3.1

```
$ gppkg -r plr-2.3.1
```

You can run the `gppkg` utility with the options `-q --all` to list the installed extensions and their versions.

Restart the database.

```
$ gpstop -r
```

### Enabling PL/R Language Support

For each database that requires its use, register the PL/R language with the SQL command `CREATE LANGUAGE` or the utility `createlang`. Because PL/R is an untrusted language, only superusers can register PL/R with a database. For example, running this command as the `gpadmin` system user registers the language with the database named `testdb`:

```
$ createlang plr -d testdb
```

PL/R is registered as an untrusted language.

### Examples

The following are simple PL/R examples.

**Example 1: Using PL/R for single row operators**

This function generates an array of numbers with a normal distribution using the R function `rnorm()`.

```sql
CREATE OR REPLACE FUNCTION r_norm(n integer, mean float8, std_dev float8) RETURNS float8[ ] AS
```
The following CREATE TABLE command uses the r_norm function to populate the table. The r_norm function creates an array of 10 numbers.

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);

Example 2: Returning PL/R data.frames in Tabular Form

Assuming your PL/R function returns an R data.frame as its output, unless you want to use arrays of arrays, some work is required to see your data.frame from PL/R as a simple SQL table:

- Create a TYPE in a Greenplum database with the same dimensions as your R data.frame:

  ```sql
  CREATE TYPE t1 AS ...
  ```

- Use this TYPE when defining your PL/R function

  ```sql
  ... RETURNS SET OF t1 AS ...
  ```

Sample SQL for this is given in the next example.

Example 3: Hierarchical Regression using PL/R

The SQL below defines a TYPE and runs hierarchical regression using PL/R:

```sql
--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 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));

Downloading and Installing R Packages

R packages are modules that contain R functions and data sets. You can install R packages to extend R and PL/R functionality in Greenplum Database.

Greenplum Database provides a collection of data science-related R libraries that can be used with the Greenplum Database PL/R language. You can download these libraries in .gppkg format from Pivotal Network. For information about the libraries, see R Data Science Library Package.

Note: If you expand Greenplum Database and add segment hosts, you must install the R packages in the R installation of the new hosts.

1. For an R package, identify all dependent R packages and each package web URL. The information can be found by selecting the given package from the following navigation page:
   http://cran.r-project.org/web/packages/available_packages_by_name.html
   As an example, the page for the R package arm indicates that the package requires the following R libraries: Matrix, lattice, lm4, R2WinBUGS, coda, abind, foreign, and MASS.
   You can also try installing the package with R CMD INSTALL command to determine the dependent packages.
   For the R installation included with the Greenplum Database PL/R extension, the required R packages are installed with the PL/R extension. However, the Matrix package requires a newer version.

2. From the command line, use the wget utility to download the tar.gz files for the arm package to the Greenplum Database master host:
   
   wget http://cran.r-project.org/src/contrib/Archive/arm/arm_1.5-03.tar.gz
   wget http://cran.r-project.org/src/contrib/Archive/Matrix/
   Matrix_0.9996875-1.tar.gz

3. Use the gscp utility and the hosts_all file to copy the tar.gz files to the same directory on all nodes of the Greenplum cluster. The hosts_all file contains a list of all the Greenplum Database segment hosts. You might require root access to do this.
   
   gscp -f hosts_all Matrix_0.9996875-1.tar.gz =:/home/gpadmin
   gscp -f /hosts_all arm_1.5-03.tar.gz =:/home/gpadmin

4. Use the gpssh utility in interactive mode to log into each Greenplum Database segment host (gpssh -f all_hosts). Install the packages from the command prompt using the R CMD INSTALL command. Note that this may require root access. For example, this R install command installs the packages for the arm package.
   
   $R_HOME/bin/R CMD INSTALL Matrix_0.9996875-1.tar.gz arm_1.5-03.tar.gz
5. Ensure that the package is installed in the $R\_HOME/library directory on all the segments (the 
gpssh can be used to install the package). For example, this gpssh command list the contents of the R 
library directory.

```bash
gpssh -f all_hosts "ls $R\_HOME/library"
```

6. Test if the R package can be loaded.

   This function performs a simple test to if an R package can be loaded:

   ```sql
   CREATE OR REPLACE FUNCTION R_test_require(fname text)
   RETURNS boolean AS
   "$BODY$
   return(require(fname,character.only=T))
   $BODY$
   LANGUAGE 'plr';
   ```

   This SQL command checks if the R package arm can be loaded:

   ```sql
   SELECT R_test_require('arm');
   ```

### Displaying R Library Information

You can use the R command line to display information about the installed libraries and functions on the 
Greenplum Database host. You can also add and remove libraries from the R installation. To start the R 
command line on the host, log into the host as the gadmin user and run the script R from the directory

```
GPHOME/ext/R-3.3.3/bin
```

This R function lists the available R packages from the R command line:

```
> library()
```

Display the documentation for a particular R package

```
> library(help="package_name")
> help(package="package_name")
```

Display the help file for an R function:

```
> help("function_name")
> ?function_name
```

   To see what packages are installed, use the R command `installed.packages()`. This will return a 
   matrix with a row for each package that has been installed. Below, we look at the first 5 rows of this matrix.

   ```r
   > installed.packages()
   ```

   Any package that does not appear in the installed packages matrix must be installed and loaded before its 
   functions can be used.

   An R package can be installed with `install.packages()`:

   ```r
   > install.packages("package_name")
   > install.packages("mypkg", dependencies = TRUE, type="source")
   ```

   Load a package from the R command line.

   ```r
   > library(" package_name ")
   ```
An R package can be removed with `remove.packages`

```r
> remove.packages("package_name")
```

You can use the R command `-e` option to run functions from the command line. For example, this command displays help on the R package MASS.

```r
$ R -e 'help("MASS")'
```

**References**

http://www.r-project.org/ - The R Project home page

https://cran.r-project.org/web/packages/PivotalR/ - The home page for PivotalR, a package that provides an R interface to operate on Greenplum Database tables and views that is similar to the R `data.frame`. PivotalR also supports using the machine learning package `MADlib` directly from R.

R documentation is installed with the Greenplum R package:

```
$GPHOME/ext/R-3.3.1/doc
```

**R Functions and Arguments**


**Passing Data Values in R**


**Aggregate Functions in R**

Greenplum PL/Python Language Extension

This section contains an overview of the Greenplum Database PL/Python Language.

- About Greenplum PL/Python
- Enabling and Removing PL/Python support
- Developing Functions with PL/Python
- Installing Python Modules
- Examples
- References

About Greenplum PL/Python

PL/Python is a loadable procedural language. With the Greenplum Database PL/Python extension, you can write a Greenplum Database user-defined functions in Python that take advantage of Python features and modules to quickly build robust database applications.

You can run PL/Python code blocks as anonymous code blocks. See the `DO` command in the Greenplum Database Reference Guide.

The Greenplum Database PL/Python extension is installed by default with Greenplum Database. Greenplum Database installs a version of Python and PL/Python. This is location of the Python installation that Greenplum Database uses:

```
$GPHOME/ext/python/
```

Greenplum Database PL/Python Limitations

- Greenplum Database does not support PL/Python triggers.
- PL/Python is available only as a Greenplum Database untrusted language.
- Updatable cursors (`UPDATE...WHERE CURRENT OF` and `DELETE...WHERE CURRENT OF`) are not supported.

Enabling and Removing PL/Python support

The PL/Python language is installed with Greenplum Database. To create and run a PL/Python user-defined function (UDF) in a database, you must register the PL/Python language with the database.

Enabling PL/Python Support

For each database that requires its use, register the PL/Python language with the SQL command `CREATE LANGUAGE` or the Greenplum Database utility `createlang`. Because PL/Python is an untrusted language, only superusers can register PL/Python with a database. For example, running this command as the `gpadmin` system user registers PL/Python with the database named `testdb`:

```
$ createlang plpythonu -d testdb
```

PL/Python is registered as an untrusted language.

Removing PL/Python Support

For a database that no longer requires the PL/Python language, remove support for PL/Python with the SQL command `DROP LANGUAGE` or the Greenplum Database `droplang` utility. Because PL/Python is an untrusted language, only superusers can remove support for the PL/Python language from a database.
For example, running this command as the `gpadmin` system user removes support for PL/Python from the database named `testdb`:

```
$ droplang plpythonu -d testdb
```

When you remove support for PL/Python, the PL/Python user-defined functions that you created in the database will no longer work.

**Developing Functions with PL/Python**

The body of a PL/Python user-defined function is a Python script. When the function is called, its arguments are passed as elements of the array `args[]`. Named arguments are also passed as ordinary variables to the Python script. The result is returned from the PL/Python function with `return` statement, or `yield` statement in case of a result-set statement.

**Arrays and Lists**

You pass SQL array values into PL/Python functions with a Python list. Similarly, PL/Python functions return SQL array values as a Python list. In the typical PL/Python usage pattern, you will specify an array with `[ ]`.

The following example creates a PL/Python function that returns an array of integers:

```
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 treats multi-dimensional arrays as lists of lists. You pass a multi-dimensional array to a PL/Python function using nested Python lists. When a PL/Python function returns a multi-dimensional array, the inner lists at each level must all be of the same size.

The following example creates a PL/Python function that takes a multi-dimensional array of integers as input. The function displays the type of the provided argument, and returns the multi-dimensional array:

```
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], <type 'list'>)
-- CONTEXT:  PL/Python function "return_multidim_py_type"
-- return_multidim_py_array
-- {[[1, 2, 3], [4, 5, 6]]}
(1 row)
```

PL/Python also accepts other Python sequences, such as tuples, as function arguments for backwards compatibility with Greenplum versions where multi-dimensional arrays were not supported. In such cases,
the Python sequences are always treated as one-dimensional arrays because they are ambiguous with composite types.

## Composite Types

You pass composite-type arguments to a PL/Python function using Python mappings. The element names of the mapping are the attribute names of the composite types. If an attribute has the null value, its mapping value is `None`.

You can return a composite type result as a sequence type (tuple or list). You must specify a composite type as a tuple, rather than a list, when it is used in a multi-dimensional array. You cannot return an array of composite types as a list because it would be ambiguous to determine whether the list represents a composite type or another array dimension. In the typical usage pattern, you will specify composite type tuples with `()`.

In the following example, you create a composite type and a PL/Python function that returns an array of the composite type:

```sql
CREATE TYPE type_record AS (
    first text,
    second int4
);

CREATE FUNCTION composite_type_as_list()
    RETURNS type_record[]
AS $$
    return [[('first', 1), ('second', 1)], [('first', 2), ('second', 2)],
    [('first', 3), ('second', 3)]];
$$ LANGUAGE plpythonu;

SELECT * FROM composite_type_as_list();
```

Refer to the PostgreSQL *Arrays, Lists* documentation for additional information on PL/Python handling of arrays and composite types.

## Executing and Preparing SQL Queries

The PL/Python `plpy` module provides two Python functions to execute an SQL query and prepare an execution plan for a query, `plpy.execute` and `plpy.prepare`. Preparing the execution plan for a query is useful if you run the query from multiple Python functions.

### `plpy.execute`

Calling `plpy.execute` with a query string and an optional limit argument causes the query to be run and the result to be returned in a Python result object. The result object emulates a list or dictionary object. The rows returned in the result object can be accessed by row number and column name. The result set row numbering starts with 0 (zero). The result object can be modified. The result object has these additional methods:

- `nrows` that returns the number of rows returned by the query.
- `status` which is the SPI_execute() return value.
For example, this Python statement in a PL/Python user-defined function executes a query.

```python
rsv = plpy.execute("SELECT * FROM my_table", 5)
```

The `plpy.execute` function returns up to 5 rows from `my_table`. The result set is stored in the `rv` object. If `my_table` has a column `my_column`, it would be accessed as:

```python
my_col_data = rv[i]["my_column"]
```

Since the function returns a maximum of 5 rows, the index `i` can be an integer between 0 and 4.

**plpy.prepare**

The function `plpy.prepare` prepares the execution plan for a query. It is called with a query string and a list of parameter types, if you have parameter references in the query. For example, this statement can be in a PL/Python user-defined function:

```python
plan = plpy.prepare("SELECT last_name FROM my_users WHERE first_name = $1", [ "text" ])
```

The string `text` is the data type of the variable that is passed for the variable `$1`. After preparing a statement, you use the function `plpy.execute` to run it:

```python
rv = plpy.execute(plan, [ "Fred" ], 5)
```

The third argument is the limit for the number of rows returned and is optional.

When you prepare an execution plan using the PL/Python module the plan is automatically saved. See the Postgres Server Programming Interface (SPI) documentation for information about the execution plans [https://www.postgresql.org/docs/8.3/static/spi.html](https://www.postgresql.org/docs/8.3/static/spi.html).

To make effective use of saved plans across function calls you use one of the Python persistent storage dictionaries SD or GD.

The global dictionary SD is available to store data between function calls. This variable is private static data. The global dictionary GD is public data, available to all Python functions within a session. Use GD with care.

Each function gets its own execution environment in the Python interpreter, so that global data and function arguments from `myfunc` are not available to `myfunc2`. The exception is the data in the GD dictionary, as mentioned previously.

This example uses the SD dictionary:

```python
CREATE FUNCTION usesavedplan() RETURNS trigger AS $$
  if SD.has_key("plan"):
    plan = SD["plan"]
  else:
    plan = plpy.prepare("SELECT 1")
  SD["plan"] = plan
  # rest of function
$$ LANGUAGE plpythonu;
```

### Handling Python Errors and Messages

The Python module `plpy` implements these functions to manage errors and messages:

- `plpy.debug`
- `plpy.log`
The message functions `plpy.error` and `plpy.fatal` raise a Python exception which, if uncaught, propagates out to the calling query, causing the current transaction or subtransaction to be aborted. The functions `raise plpy.ERROR(msg)` and `raise plpy.FATAL(msg)` are equivalent to calling `plpy.error` and `plpy.fatal`, respectively. The other message functions only generate messages of different priority levels.

Whether messages of a particular priority are reported to the client, written to the server log, or both is controlled by the Greenplum Database server configuration parameters `log_min_messages` and `client_min_messages`. For information about the parameters see the Greenplum Database Reference Guide.

### Using the dictionary GD To Improve PL/Python Performance

In terms of performance, importing a Python module is an expensive operation and can affect performance. If you are importing the same module frequently, you can use Python global variables to load the module on the first invocation and not require importing the module on subsequent calls. The following PL/Python function uses the GD persistent storage dictionary to avoid importing a module if it has already been imported and is in the GD.

```sql
psql=#
    CREATE FUNCTION pytest() returns text as $$
    if 'mymodule' not in GD:
        import mymodule
        GD['mymodule'] = mymodule
    return GD['mymodule'].sumd([1,2,3])
$$;
```

### Installing Python Modules

When you install a Python module on Greenplum Database, the Greenplum Database Python environment must have the module added to it across all segment hosts and mirror hosts in the cluster. When expanding Greenplum Database, you must add the Python modules to the new segment hosts. You can use the Greenplum Database utilities `gpssh` and `gpscp` run commands on Greenplum Database hosts and copy files to the hosts. For information about the utilities, see the Greenplum Database Utility Guide.

As part of the Greenplum Database installation, the `gpadmin` user environment is configured to use Python that is installed with Greenplum Database.

To check the Python environment, you can use the `which` command:

```
which python
```

The command returns the location of the Python installation. The Python installed with Greenplum Database is in the Greenplum Database `ext/python` directory.

```
/path_to_greenplum-db/ext/python/bin/python
```

Greenplum Database provides a collection of data science-related Python libraries that can be used with the Greenplum Database PL/Python language. You can download these libraries in .gppkg format from Pivotal Network. For information about the libraries, see Python Data Science Module Package.
If you are building a Python module, you must ensure that the build creates the correct executable. For example on a Linux system, the build should create a 64-bit executable.

Before building a Python module to be installed, ensure that the appropriate software to build the module is installed and properly configured. The build environment is required only on the host where you build the module.

These are examples of installing and testing Python modules:

- Simple Python Module Installation Example (setuptools)
- Complex Python Installation Example (NumPy)
- Testing Installed Python Modules

**Simple Python Module Installation Example (setuptools)**

This example manually installs the Python `setuptools` module from the Python Package Index repository. The module lets you easily download, build, install, upgrade, and uninstall Python packages.

This example first builds the module from a package and installs the module on a single host. Then the module is built and installed on segment hosts.

1. Get the module package from the Python Package Index site. For example, run this `wget` command on a Greenplum Database host as the `gpadmin` user to get the tar file.

   ```
   wget --no-check-certificate https://pypi.python.org/packages/source/s/setuptools/setuptools-18.4.tar.gz
   ```

2. Extract the files from the tar file.

   ```
   tar -xzvf distribute-0.6.21.tar.gz
   ```

3. Go to the directory that contains the package files and run the Python scripts to build and install the Python package.

   ```
   cd setuptools-18.4
   python setup.py build && python setup.py install
   ```

4. The following Python command returns no errors if the module is available to Python.

   ```
   python -c "import setuptools"
   ```

5. Copy the package to the Greenplum Database hosts with the `gpscp` utility. For example, this command copies the tar file from the current host to the host systems listed in the file `remote-hosts`.

   ```
   gpscp -f remote-hosts setuptools-18.4.tar.gz =:/home/gpadmin
   ```

6. Run the commands to build, install, and test the package with `gpssh` utility on the hosts listed in the file `remote-hosts`. The file `remote-hosts` lists all the remote Greenplum Database segment hosts:

   ```
   gpssh -f remote_hosts
   >>> tar -xzvf distribute-0.6.21.tar.gz
   >>> cd setuptools-18.4
   >>> python setup.py build && python setup.py install
   >>> python -c "import setuptools"
   >>> exit
   ```

The `setuptools` package installs the `easy_install` utility that lets you install Python packages from the Python Package Index repository. For example, this command installs Python PIP utility from the Python Package Index site.

```
easy_install pip
```
You can use the `gpsssh` utility to run the `easy-install` command on all the Greenplum Database segment hosts.

**Complex Python Installation Example (NumPy)**

This example builds and installs the Python module NumPy. NumPy is a module for scientific computing with Python. For information about NumPy, see [http://www.numpy.org/](http://www.numpy.org/).

Building the NumPy package requires this software:

- OpenBLAS libraries, an open source implementation of BLAS (Basic Linear Algebra Subprograms).
- The gcc compilers: gcc, gcc-gfortran, and gcc-c++. The compilers are required to build the OpenBLAS libraries. See [OpenBLAS Prerequisites](#).

This example process assumes `yum` is installed on the Greenplum Database segment hosts and the `gpadmin` user is a member of `sudoers` with root privileges on the hosts.

Download the OpenBLAS and NumPy source files. For example, these `wget` commands download tar files into the directory `packages`:

```
wget --directory-prefix=packages http://github.com/xianyi/OpenBLAS/tarball/v0.2.8
wget --directory-prefix=packages http://sourceforge.net/projects/numpy/files/NumPy/1.8.0/numpy-1.8.0.tar.gz/download
```

Distribute the software to the Greenplum Database hosts. For example, if you download the software to `/home/gpadmin/packages` these commands create the directory on the hosts and copies the software to `hosts for the hosts listed in the gpdb_remotes file`.

```
gpssh -f gpdb_remotes mkdir packages
gscp -f gpdb_remotes packages/* =:/home/gpadmin/packages
```

**OpenBLAS Prerequisites**

If needed, use `yum` to install gcc compilers from system repositories. The compilers are required on all hosts where you compile OpenBLAS:

```
sudo yum -y install gcc gcc-gfortran gcc-c++
```

**Note:** If you cannot install the correct compiler versions with yum, you can download the gcc compilers, including gfortran, from source and install them. These two commands download and install the compilers:

```
wget http://gfortran.com/download/x86_64/snapshots/gcc-4.4.tar.xz
tar xf gcc-4.4.tar.xz -C /usr/local/
```

If you installed gcc manually from a tar file, add the new gcc binaries to `PATH` and `LD_LIBRARY_PATH`:

```
export PATH=$PATH:/usr/local/gcc-4.4/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/gcc-4.4/lib
```

Create a symbolic link to g++ and call it gxx

```
sudo ln -s /usr/bin/g++ /usr/bin/gxx
```

You might also need to create symbolic links to any libraries that have different versions available for example `libppl_c.so.4` to `libppl_c.so.2`.
If needed, you can use the `gpscp` utility to copy files to Greenplum Database hosts and the `gpssh` utility to run commands on the hosts.

**Build and Install OpenBLAS Libraries**

Before build and install the NumPy module, you install the OpenBLAS libraries. This section describes how to build and install the libraries on a single host.

1. Extract the OpenBLAS files from the file. These commands extract the files from the OpenBLAS tar file and simplify the directory name that contains the OpenBLAS files.

   ```sh
   tar -xzvf packages/v0.2.8 -C /home/gpadmin/packages
   mv /home/gpadmin/packages/xianyi-OpenBLAS-9c51cdf /home/gpadmin/packages/OpenBLAS
   ```

2. Compile OpenBLAS. These commands set the `LIBRARY_PATH` environment variable and run the `make` command to build OpenBLAS libraries.

   ```sh
   cd /home/gpadmin/packages/OpenBLAS
   export LIBRARY_PATH=$LD_LIBRARY_PATH
   make FC=gfortran USE_THREAD=0
   ```

3. These commands install the OpenBLAS libraries in `/usr/local` as root and change the owner of the files to `gpadmin`.

   ```sh
   cd /home/gpadmin/packages/OpenBLAS/
   sudo make PREFIX=/usr/local install
   sudo ldconfig
   sudo chown -R gpadmin /usr/local/lib
   ```

   These are the libraries that are installed and symbolic links that are created:

   ```
   libopenblas.a -> libopenblas_sandybridge-r0.2.8.a
   libopenblas_sandybridge-r0.2.8.a
   libopenblas_sandybridge-r0.2.8.so
   libopenblas.so -> libopenblas_sandybridge-r0.2.8.so
   libopenblas.so.0 -> libopenblas_sandybridge-r0.2.8.so
   ```

   You can use the `gpssh` utility to build and install the OpenBLAS libraries on multiple hosts.

   All the Greenplum Database hosts (master and segment hosts) have identical configurations. You can copy the OpenBLAS libraries from the system where they were built instead of building the OpenBlas libraries on all the hosts. For example, these `gpssh` and `gpscp` commands copy and install the OpenBLAS libraries on the hosts listed in the `gpdb_remotes` file.

   ```sh
   gpssh -f gpdb_remotes -e 'sudo yum -y install gcc gcc-gfortran gcc-c++'
gpssh -f gpdb_remotes -e 'ln -s /usr/bin/g++ /usr/bin/gxx'
gpssh -f gpdb_remotes -e sudo chown gpadmin /usr/local/lib
   ```

   ```sh
gpsscp -f gpdb_remotes /usr/local/lib/libopen* sandy* =:/usr/local/lib
   ```

   ```sh
gpssh -f gpdb_remotes
   >>> cd /usr/local/lib
   >>> ln -s libopenblas_sandybridge-r0.2.8.a libopenblas.a
   >>> ln -s libopenblas_sandybridge-r0.2.8.so libopenblas.so
   >>> ln -s libopenblas_sandybridge-r0.2.8.so libopenblas.so.0
   >>> sudo ldconfig
   ```
Build and Install NumPy

After you have installed the OpenBLAS libraries, you can build and install NumPy module. These steps install the NumPy module on a single host. You can use the `gpssh` utility to build and install the NumPy module on multiple hosts.

1. Go to the `packages` subdirectory and get the NumPy module source and extract the files.

   ```
   cd /home/gpadmin/packages
   tar -xzf numpy-1.8.0.tar.gz
   ```

2. Set up the environment for building and installing NumPy.

   ```
   export BLAS=/usr/local/lib/libopenblas.a
   export LAPACK=/usr/local/lib/libopenblas.a
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib/
   export LIBRARY_PATH=$LD_LIBRARY_PATH
   ```

3. Go to the NumPy directory and build and install NumPy. Building the NumPy package might take some time.

   ```
   cd numpy-1.8.0
   python setup.py build
   python setup.py install
   ```

   **Note:** If the NumPy module did not successfully build, the NumPy build process might need a `site.cfg` that specifies the location of the OpenBLAS libraries. Create the file `site.cfg` in the NumPy package directory:

   ```
   cd ~/packages/numpy-1.8.0
   touch site.cfg
   ```

   Add the following to the `site.cfg` file and run the NumPy build command again:

   ```
   [default]
   library_dirs = /usr/local/lib

   [atlas]
   atlas_libs = openblas
   library_dirs = /usr/local/lib

   [lapack]
   lapack_libs = openblas
   library_dirs = /usr/local/lib

   # added for scikit-learn
   [openblas]
   libraries = openblas
   library_dirs = /usr/local/lib
   include_dirs = /usr/local/include
   ```

4. The following Python command ensures that the module is available for import by Python on a host system.

   ```
   python -c "import numpy"
   ```

As in the simple module installation, you can use the `gpssh` utility to build, install, and test the module on Greenplum Database segment hosts.

The environment variables that are require to build the NumPy module are also required in the `gpadmin` user environment when running Python NumPy functions. You can use the `gpssh` utility with the `echo`
command to add the environment variables to the `.bashrc` file. For example, these `echo` commands add the environment variables to the `.bashrc` file in the user home directory.

```
echo -e \n\n#Needed for NumPy' >> ~/.bashrc
echo -e 'export BLAS=/usr/local/lib/libopenblas.a' >> ~/.bashrc
echo -e 'export LAPACK=/usr/local/lib/libopenblas.a' >> ~/.bashrc
echo -e 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib' >> ~/.bashrc
echo -e 'export LIBRARY_PATH=$LD_LIBRARY_PATH' >> ~/.bashrc
```

**Testing Installed Python Modules**

You can create a simple PL/Python user-defined function (UDF) to validate that Python a module is available in the Greenplum Database. This example tests the NumPy module.

This PL/Python UDF imports the NumPy module. The function returns `SUCCESS` if the module is imported, and `FAILURE` if an import error occurs.

```
CREATE OR REPLACE FUNCTION plpy_test(x int)
returns text
as $$
try:
    from numpy import *
    return 'SUCCESS'
except ImportError, e:
    return 'FAILURE'
$$ language plpythonu;
```

Create a table that contains data on each Greenplum Database segment instance. Depending on the size of your Greenplum Database installation, you might need to generate more data to ensure data is distributed to all segment instances.

```
CREATE TABLE DIST AS (SELECT x FROM generate_series(1,50) x ) DISTRIBUTED RANDOMLY ;
```

This `SELECT` command runs the UDF on the segment hosts where data is stored in the primary segment instances.

```
SELECT gp_segment_id, plpy_test(x) AS status
FROM dist
GROUP BY gp_segment_id, status
ORDER BY gp_segment_id, status;
```

The `SELECT` command returns `SUCCESS` if the UDF imported the Python module on the Greenplum Database segment instance. If the `SELECT` command returns `FAILURE`, you can find the segment host of the segment instance host. The Greenplum Database system table `gp_segment_configuration` contains information about mirroring and segment configuration. This command returns the host name for a segment ID.

```
SELECT hostname, content AS seg_ID FROM gp_segment_configuration
WHERE content = seg_id ;
```

If `FAILURE` is returned, these are some possible causes:

- A problem accessing required libraries. For the NumPy example, a Greenplum Database might have a problem accessing the OpenBLAS libraries or the Python libraries on a segment host.
Make sure you get no errors when running command on the segment host as the `gpadmin` user. This `gpssh` command tests importing the `numpy` module on the segment host `mdw1`.

```bash
  gpssh -h mdw1 python -c "import numpy"
```

- If the Python `import` command does not return an error, environment variables might not be configured in the Greenplum Database environment. For example, the variables are not in the `.bashrc` file, or Greenplum Database might not have been restarted after adding the environment variables to the `.bashrc` file.

Ensure sure that the environment variables are properly set and then restart the Greenplum Database. For the NumPy example, ensure the environment variables listed at the end of the section _Build and Install NumPy_ are defined in the `.bashrc` file for the `gpadmin` user on the master and segment hosts.

**Note:** On the Greenplum Database master and segment hosts, the `.bashrc` file for the `gpadmin` user must source the file `$GPHOME/greenplum_path.sh`.

### Examples

This PL/Python UDF returns the maximum of two integers:

```sql
CREATE FUNCTION pymax (a integer, b integer) RETURNS integer AS $$
  if (a is None) or (b is None):
    return None
  if a > b:
    return a
  return b
$$ LANGUAGE plpythonu;
```

You can use the `STRICT` property to perform the null handling instead of using the two conditional statements.

```sql
CREATE FUNCTION pymax (a integer, b integer) RETURNS integer AS $$
  return max(a, b)
$$ LANGUAGE plpythonu STRICT;
```

You can run the user-defined function `pymax` with `SELECT` command. This example runs the UDF and shows the output.

```sql
SELECT ( pymax(123, 43));
```

```sql
<table>
<thead>
<tr>
<th>column1</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
</tr>
</tbody>
</table>
```

This example that returns data from an SQL query that is run against a table. These two commands create a simple table and add data to the table.

```sql
CREATE TABLE sales (id int, year int, qtr int, day int, region text) DISTRIBUTED BY (id);
```

```sql
INSERT INTO sales VALUES
(1, 2014, 1,1, 'usa'),
(2, 2002, 2,2, 'europe'),
(3, 2014, 3,3, 'asia'),
(4, 2014, 4,4, 'usa'),
(5, 2014, 1,5, 'europe'),
```
This PL/Python UDF executes a `SELECT` command that returns 5 rows from the table. The Python function returns the `REGION` value from the row specified by the input value. In the Python function, the row numbering starts from 0. Valid input for the function is an integer between 0 and 4.

```sql
CREATE OR REPLACE FUNCTION mypytest(a integer)
    RETURNS text
AS $$
    rv = plpy.execute("SELECT * FROM sales ORDER BY id", 5)
    region = rv[a]["region"]
    return region
$$ language plpythonu;
```

Running this `SELECT` statement returns the `REGION` column value from the third row of the result set.

```sql
SELECT mypytest(2) ;
```

This command deletes the UDF from the database.

```sql
DROP FUNCTION mypytest(integer) ;
```

This example executes the PL/Python function in the previous example as an anonymous block with the `DO` command. In the example, the anonymous block retrieves the input value from a temporary table.

```sql
CREATE TEMP TABLE mytemp AS VALUES (2) DISTRIBUTED RANDOMLY;
DO $$
    temprow = plpy.execute("SELECT * FROM mytemp", 1)
    myval = temprow[0]["column1"]
    rv = plpy.execute("SELECT * FROM sales ORDER BY id", 5)
    region = rv[myval]["region"]
    plpy.notice("region is %s" % region)
$$ language plpythonu;
```

### References

#### Technical References

For information about the Python language, see [https://www.python.org/](https://www.python.org/).

For information about PL/Python see the PostgreSQL documentation at [https://www.postgresql.org/docs/8.3/static/plpython.html](https://www.postgresql.org/docs/8.3/static/plpython.html).

For information about Python Package Index (PyPI), see [https://pypi.python.org/pypi](https://pypi.python.org/pypi).

These are some Python modules that can be downloaded:

- SciPy library provides user-friendly and efficient numerical routines such as routines for numerical integration and optimization [http://www.scipy.org/scipylib/index.html](http://www.scipy.org/scipylib/index.html). This `wget` command downloads the SciPy package tar file.

```bash
wget http://sourceforge.net/projects/scipy/files/scipy/0.10.1/scipy-0.10.1.tar.gz/download
```

• Natural Language Toolkit (nltk) is a platform for building Python programs to work with human language data. [http://www.nltk.org/](http://www.nltk.org/). This `wget` command downloads the nltk package tar file.

```bash
wget http://pypi.python.org/packages/source/n/nltk/nltk-2.0.2.tar.gz#md5=6e714ff74c3398e88be084748df4e657
```

**Note:** The Python package Distribute [https://pypi.python.org/pypi/distribute](https://pypi.python.org/pypi/distribute) is required for nltk. The Distribute module should be installed the nltk package. This `wget` command downloads the Distribute package tar file.

```bash
wget http://pypi.python.org/packages/source/d/distribute/distribute-0.6.21.tar.gz
```
Greenplum PL/Java Language Extension

This section contains an overview of the Greenplum Database PL/Java language.

• About PL/Java
• About Greenplum Database PL/Java
• Installing PL/Java
• Uninstalling PL/Java
• Enabling PL/Java and Installing JAR Files
• Writing PL/Java functions
• Using JDBC
• Exception Handling
• Savepoints
• Logging
• Security
• Some PL/Java Issues and Solutions
• Example
• References

About PL/Java

With the Greenplum Database PL/Java extension, you can write Java methods using your favorite Java IDE and install the JAR files that contain those methods into Greenplum Database.

Greenplum Database PL/Java package is based on the open source PL/Java 1.4.0. Greenplum Database PL/Java provides the following features.

• Ability to execute PL/Java functions with Java 1.7 or higher.
• Ability to specify Java runtime.
• Standardized utilities (modeled after the SQL 2003 proposal) to install and maintain Java code in the database.
• Standardized mappings of parameters and result. Complex types as well as sets are supported.
• An embedded, high performance, JDBC driver utilizing the internal Greenplum Database SPI routines.
• Metadata support for the JDBC driver. Both DatabaseMetaData and ResultSetMetaData are included.
• The ability to return a ResultSet from a query as an alternative to building a ResultSet row by row.
• Full support for savepoints and exception handling.
• The ability to use IN, INOUT, and OUT parameters.
• Two separate Greenplum Database languages:
  • pljava, TRUSTED PL/Java language
  • pljavau, UNTRUSTED PL/Java language
• Transaction and Savepoint listeners enabling code execution when a transaction or savepoint is committed or rolled back.
• Integration with GNU GCJ on selected platforms.

A function in SQL will appoint a static method in a Java class. In order for the function to execute, the appointed class must available on the class path specified by the Greenplum Database sever configuration parameter pljava_classpath. The PL/Java extension adds a set of functions that helps installing and maintaining the java classes. Classes are stored in normal Java archives, JAR files. A JAR file can optionally contain a deployment descriptor that in turn contains SQL commands to be executed when the JAR is deployed or undeployed. The functions are modeled after the standards proposed for SQL 2003.
PL/Java implements a standardized way of passing parameters and return values. Complex types and sets are passed using the standard JDBC ResultSet class.

A JDBC driver is included in PL/Java. This driver calls Greenplum Database internal SPI routines. The driver is essential since it is common for functions to make calls back to the database to fetch data. When PL/Java functions fetch data, they must use the same transactional boundaries that are used by the main function that entered PL/Java execution context.

PL/Java is optimized for performance. The Java virtual machine executes within the same process as the backend to minimize call overhead. PL/Java is designed with the objective to enable the power of Java to the database itself so that database intensive business logic can execute as close to the actual data as possible.

The standard Java Native Interface (JNI) is used when bridging calls between the backend and the Java VM.

**About Greenplum Database PL/Java**

There are a few key differences between the implementation of PL/Java in standard PostgreSQL and Greenplum Database.

**Functions**

The following functions are not supported in Greenplum Database. The classpath is handled differently in a distributed Greenplum Database environment than in the PostgreSQL environment.

- sqlj.install_jar
- sqlj.replace_jar
- sqlj.remove_jar
- sqlj.get_classpath
- sqlj.set_classpath

Greenplum Database uses the `pljava_classpath` server configuration parameter in place of the `sqlj.set_classpath` function.

**Server Configuration Parameters**

The following server configuration parameters are used by PL/Java in Greenplum Database. These parameters replace the `pljava.*` parameters that are used in the standard PostgreSQL PL/Java implementation:

- `pljava_classpath`

  A colon (:) separated list of the jar files containing the Java classes used in any PL/Java functions. The jar files must be installed in the same locations on all Greenplum Database hosts. With the trusted PL/Java language handler, jar file paths must be relative to the `$GPHOME/lib/postgresql/java/` directory. With the untrusted language handler (javaU language tag), paths may be relative to `$GPHOME/lib/postgresql/java/` or absolute.

  The server configuration parameter `pljava_classpath_insecure` controls whether the server configuration parameter `pljava_classpath` can be set by a user without Greenplum Database superuser privileges. When `pljava_classpath_insecure` is enabled, Greenplum Database developers who are working on PL/Java functions do not have to be database superusers to change `pljava_classpath`.

  **Warning:** Enabling `pljava_classpath_insecure` exposes a security risk by giving non-administrator database users the ability to run unauthorized Java methods.

- `pljava_statement_cache_size`

  Sets the size in KB of the Most Recently Used (MRU) cache for prepared statements.
• **pljava_release_lingering_savepoints**

  If **TRUE**, lingering savepoints will be released on function exit. If **FALSE**, they will be rolled back.

• **pljava_vmoptions**

  Defines the start up options for the Greenplum Database Java VM.

  See the *Greenplum Database Reference Guide* for information about the Greenplum Database server configuration parameters.

### Installing PL/Java

For Greenplum Database, the PL/Java extension is available as a package. Download the package from the Greenplum Database page on *Pivotal Network* and then install it with the Greenplum Package Manager (gppkg).

The *gppkg* utility installs Greenplum Database extensions, along with any dependencies, on all hosts across a cluster. It also automatically installs extensions on new hosts in the case of system expansion and segment recovery.

For information about *gppkg*, see the *Greenplum Database Utility Guide*.

**Important:** PL/Java requires a Java environment on each Greenplum Database host. Ensure that the same Java environment is at the same location on all hosts: masters and segments.

To install and use PL/Java:

1. Specify the Java version used by PL/Java. Set the environment variables **JAVA_HOME** and **LD_LIBRARY_PATH** in the *greenplum_path.sh*.
2. Install the Greenplum Database PL/Java extension.
3. Enable the language for each database where you intend to use PL/Java.
4. Install user-created JAR files containing Java methods into the same directory on all Greenplum Database hosts.
5. Add the name of the JAR file to the Greenplum Database server configuration parameter **pljava_classpath**. The parameter lists the installed JAR files. For information about the parameter, see the *Greenplum Database Reference Guide*.

### Installing the Greenplum PL/Java Extension

Before you install the PL/Java extension, make sure that your Greenplum database is running, you have sourced *greenplum_path.sh*, and that the **$MASTER_DATA_DIRECTORY** and **$GPHOME** variables are set.

1. Download the PL/Java extension package from the Greenplum Database page on *Pivotal Network* then copy it to the master host.
2. Install the software extension package by running the *gppkg* command. This example installs the PL/Java extension package on a Linux system:

   ```
   $ gppkg -i pljava-ossv1.5.0_pv1.4.0-GPDB-version-orca-rhel6-x86_64.gppkg
   ```

3. Ensure the environment variables **JAVA_HOME** and **LD_LIBRARY_PATH** are set properly in **$GPHOME/greenplum_path.sh** on all Greenplum Database hosts.

   - Set the **JAVA_HOME** variable to the directory where your Java Runtime is installed. For example, for Oracle JRE this directory would be `/usr/java/latest`. For OpenJDK, the directory is `/usr/lib/jvm/jre`. This example changes the environment variable to use `/usr/java/latest`.

   ```
   JAVA_HOME=/usr/java/latest
   ```

   - Set the **LD_LIBRARY_PATH** to include the directory with Java server runtime libraries. PL/Java depends on `libjvm.so` and the shared object should be in your **LD_LIBRARY_PATH**. By default,
libjvm.so is available in $JAVA_HOME/lib/amd64/server. This example adds the directory to the environment variable.

LD_LIBRARY_PATH=$GPHOME/lib:$GPHOME/ext/python/lib:$JAVA_HOME/lib/amd64/server:$LD_LIBRARY_PATH

This example gpscp command copies the file to all hosts specified in the file ghosts_file.

$ gpscp -f ghosts_file $GPHOME/greenplum_path.sh
  =:$GPHOME/greenplum_path.sh

4. Reload greenplum_path.sh.

$ source $GPHOME/greenplum_path.sh

5. Restart Greenplum Database.

$ gpstop -r

Uninstalling PL/Java

- Remove PL/Java Support for a Database
- Uninstall the Java JAR files and Software Package

Remove PL/Java Support for a Database

For a database that no long requires the PL/Java language, remove support for PL/Java. Run the uninstall.sql file as the gpadmin user. For example, this command disables the PL/Java language in the specified database.

$ psql -d mydatabase
  -f $GPHOME/share/postgresql/pljava/uninstall.sql

Uninstall the Java JAR files and Software Package

If no databases have PL/Java as a registered language, remove the Java JAR files and uninstall the Greenplum PL/Java extension with the gppkg utility.

1. Remove the pljava_classpath server configuration parameter from the postgresql.conf file on all Greenplum Database hosts. For example:

   $ gpconfig -r pljava_classpath

2. Remove the JAR files from the directories where they were installed on all Greenplum Database hosts. For information about JAR file installation directories, see Enabling PL/Java and Installing JAR Files.

3. Use the Greenplum gppkg utility with the -r option to uninstall the PL/Java extension. This example uninstalls the PL/Java extension on a Linux system:

   $ gppkg -r pljava-ossvl.4.0_pvl.3_gpdbversion

   You can run the gppkg utility with the options -q --all to list the installed extensions and their versions.

4. Reload greenplum_path.sh.

   $ source $GPHOME/greenplum_path.sh
5. Restart the database.

   $ gpstop -r

**Enabling PL/Java and Installing JAR Files**

Perform the following steps as the Greenplum Database administrator `gpadmin`.

1. Enable PL/Java by running the SQL script `$GPHOME/share/postgresql/pljava/install.sql` in the databases that will use PL/Java. For example, this example enables PL/Java on the database `mytestdb`:

   ```
   $ psql -d mytestdb \
   -f $GPHOME/share/postgresql/pljava/install.sql
   ```

   The script `install.sql` registers both the trusted and untrusted PL/Java language.

2. Copy your Java archives (JAR files) to the same directory on all Greenplum Database hosts. This example uses the Greenplum Database `gpscp` utility to copy the file `myclasses.jar` to the directory `$GPHOME/lib/postgresql/java/`:

   ```
   $ gpscp -f gphosts_file myclasses.jar \
   =:/usr/local/greenplum-db/lib/postgresql/java/
   ```

   The file `gphosts_file` contains a list of the Greenplum Database hosts.

3. Set the `pljava_classpath` server configuration parameter in the master `postgresql.conf` file. For this example, the parameter value is a colon (:) separated list of the JAR files. For example:

   ```
   $ gpconfig -c pljava_classpath -v 'examples.jar:myclasses.jar'
   ```

   The file `examples.jar` is installed when you install the PL/Java extension package with the `gppkg` utility.

   **Note:** If you install JAR files in a directory other than `$GPHOME/lib/postgresql/java/`, you must specify the absolute path to the JAR file. Each JAR file must be in the same location on all Greenplum Database hosts. For more information about specifying the location of JAR files, see the information about the `pljava_classpath` server configuration parameter in the Greenplum Database Reference Guide.

4. Reload the `postgresql.conf` file.

   ```
   $ gpstop -u
   ```

5. (optional) Greenplum provides an `examples.sql` file containing sample PL/Java functions that you can use for testing. Run the commands in this file to create the test functions (which use the Java classes in `examples.jar`).

   ```
   $ psql -f $GPHOME/share/postgresql/pljava/examples.sql
   ```

**Writing PL/Java functions**

Information about writing functions with PL/Java.

- SQL Declaration
- Type Mapping
- NULL Handling
- Complex Types
- Returning Complex Types
- Returning Complex Types
- **Functions That Return Sets**
- **Returning a SETOF < scalar type >**
- **Returning a SETOF < complex type >**

### SQL Declaration

A Java function is declared with the name of a class and a static method on that class. The class will be resolved using the classpath that has been defined for the schema where the function is declared. If no classpath has been defined for that schema, the public schema is used. If no classpath is found there either, the class is resolved using the system classloader.

The following function can be declared to access the static method `getProperty` on the `java.lang.System` class:

```sql
CREATE FUNCTION getsysprop(VARCHAR)
RETURNS VARCHAR
AS 'java.lang.System.getProperty'
LANGUAGE java;
```

Run the following command to return the Java `user.home` property:

```sql
SELECT getsysprop('user.home');
```

### Type Mapping

Scalar types are mapped in a straight forward way. This table lists the current mappings.

**Table 234: PL/Java data type mapping**

<table>
<thead>
<tr>
<th>PostgreSQL</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>boolean</td>
</tr>
<tr>
<td>char</td>
<td>byte</td>
</tr>
<tr>
<td>int2</td>
<td>short</td>
</tr>
<tr>
<td>int4</td>
<td>int</td>
</tr>
<tr>
<td>int8</td>
<td>long</td>
</tr>
<tr>
<td>varchar</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>text</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>bytea</td>
<td>byte[ ]</td>
</tr>
<tr>
<td>date</td>
<td>java.sql.Date</td>
</tr>
<tr>
<td>time</td>
<td>java.sql.Time (stored value treated as local time)</td>
</tr>
<tr>
<td>timetz</td>
<td>java.sql.Time</td>
</tr>
<tr>
<td>timestamp</td>
<td>java.sql.Timestamp (stored value treated as local time)</td>
</tr>
<tr>
<td>timestamptz</td>
<td>java.sql.Timestamp</td>
</tr>
<tr>
<td>complex</td>
<td>java.sql.ResultSet</td>
</tr>
<tr>
<td>setof complex</td>
<td>java.sql.ResultSet</td>
</tr>
</tbody>
</table>
All other types are mapped to java.lang.String and will utilize the standard textin/textout routines registered for respective type.

NULL Handling

The scalar types that map to Java primitives can not be passed as NULL values. To pass NULL values, those types can have an alternative mapping. You enable this mapping by explicitly denoting it in the method reference.

```sql
CREATE FUNCTION trueIfEvenOrNull(integer)
    RETURNS bool
    AS 'foo.fee.Fum.trueIfEvenOrNull(java.lang.Integer)'
    LANGUAGE java;
```

The Java code would be similar to this:

```java
package foo.fee;
public class Fum {
    static boolean trueIfEvenOrNull(Integer value) {
        return (value == null) ? true : (value.intValue() % 2) == 0;
    }
}
```

The following two statements both yield true:

```sql
SELECT trueIfEvenOrNull(NULL);
SELECT trueIfEvenOrNull(4);
```

In order to return NULL values from a Java method, you use the object type that corresponds to the primitive (for example, you return java.lang.Integer instead of int). The PL/Java resolve mechanism finds the method regardless. Since Java cannot have different return types for methods with the same name, this does not introduce any ambiguity.

Complex Types

A complex type will always be passed as a read-only java.sql.ResultSet with exactly one row. The ResultSet is positioned on its row so a call to next() should not be made. The values of the complex type are retrieved using the standard getter methods of the ResultSet.

Example:

```sql
CREATE TYPE complexTest
    AS(base integer, incbase integer, ctime timestamptz);
CREATE FUNCTION useComplexTest(complexTest)
    RETURNS VARCHAR
    AS 'foo.fee.Fum.useComplexTest'
    IMMUTABLE LANGUAGE java;
```

In the Java class Fum, we add the following static method:

```java
public static String useComplexTest(ResultSet complexTest) throws SQLException {
    int base = complexTest.getInt(1);
    int incbase = complexTest.getInt(2);
    Timestamp ctime = complexTest.getTimestamp(3);
```
Returning Complex Types

Java does not stipulate any way to create a ResultSet. Hence, returning a ResultSet is not an option. The SQL-2003 draft suggests that a complex return value should be handled as an IN/OUT parameter. PL/Java implements a ResultSet that way. If you declare a function that returns a complex type, you will need to use a Java method with boolean return type with a last parameter of type java.sql.ResultSet. The parameter will be initialized to an empty updateable ResultSet that contains exactly one row.

Assume that the complexTest type in previous section has been created.

```sql
CREATE FUNCTION createComplexTest(int, int)
RETURNS complexTest
AS 'foo.fee.Fum.createComplexTest'
IMMUTABLE LANGUAGE java;
```

The PL/Java method resolve will now find the following method in the Fum class:

```java
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;
}
```

The return value denotes if the receiver should be considered as a valid tuple (true) or NULL (false).

Functions That Return Sets

When returning result set, you should not build a result set before returning it, because building a large result set would consume a large amount of resources. It is better to produce one row at a time. Incidentally, that is what the Greenplum Database backend expects a function with SETOF return to do. You can return a SETOF a scalar type such as an int, float or varchar, or you can return a SETOF a complex type.

Returning a SETOF <scalar type>

In order to return a set of a scalar type, you need create a Java method that returns something that implements the java.util.Iterator interface. Here is an example of a method that returns a SETOF varchar:

```sql
CREATE FUNCTION javatest.getSystemProperties()
RETURNS SETOF varchar
AS 'foo.fee.Bar.getNames'
IMMUTABLE LANGUAGE java;
```

This simple Java method returns an iterator:

```java
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();
    }
}

Returning a SETOF <complex type>

A method returning a SETOF <complex type> must use either the interface
org.postgresql.pljava.ResultSetProvider or org.postgresql.pljava.ResultSetHandle. The reason for having two interfaces is that they cater for optimal handling of two distinct use cases. The former is for cases when you want to dynamically create each row that is to be returned from the SETOF function. The latter makes is in cases where you want to return the result of an executed query.

Using the ResultSetProvider Interface

This interface has two methods. The boolean assignRowValues(java.sql.ResultSet tupleBuilder, int rowNumber) and the void close() method. The Greenplum Database query evaluator will call the assignRowValues repeatedly until it returns false or until the evaluator decides that it does not need any more rows. Then it calls close.

You can use this interface the following way:

```java
CREATE FUNCTION javatest.listComplexTests(int, int)
    RETURNS SETOF complexTest
    AS 'foo.fee.Fum.listComplexTest'
    IMMUTABLE LANGUAGE java;
```

The function maps to a static java method that returns an instance that implements the ResultSetProvider interface.

```java
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;
    }
}
The `listComplexTests` method is called once. It may return NULL if no results are available or an instance of the `ResultSetProvider`. Here the Java class `Fum` implements this interface so it returns an instance of itself. The method `assignRowValues` will then be called repeatedly until it returns false. At that time, `close` will be called.

**Using the `ResultSetHandle` Interface**

This interface is similar to the `ResultSetProvider` interface in that it has a `close()` method that will be called at the end. But instead of having the evaluator call a method that builds one row at a time, this method has a method that returns a ResultSet. The query evaluator will iterate over this set and deliver the ResultSet contents, one tuple at a time, to the caller until a call to `next()` returns false or the evaluator decides that no more rows are needed.

Here is an example that executes a query using a statement that it obtained using the default connection. The SQL suitable for the deployment descriptor looks like this:

```sql
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;
```

And in the Java package `org.postgresql.pljava.example` a class `Users` is added:

```java
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");
}

Using JDBC

PL/Java contains a JDBC driver that maps to the PostgreSQL SPI functions. A connection that maps to the current transaction can be obtained using the following statement:

```java
Connection conn =
    DriverManager.getConnection("jdbc:default:connection");
```

After obtaining a connection, you can prepare and execute statements similar to other JDBC connections. These are limitations for the PL/Java JDBC driver:

- The transaction cannot be managed in any way. Thus, you cannot use methods on the connection such as:
  - commit()
  - rollback()
  - setAutoCommit()
  - setTransactionIsolation()

- Savepoints are available with some restrictions. A savepoint cannot outlive the function in which it was set and it must be rolled back or released by that same function.
- A ResultSet returned from `executeQuery()` are always `FETCH_FORWARD` and `CONCUR_READ_ONLY`.
- Meta-data is only available in PL/Java 1.1 or higher.
- `CallableStatement` (for stored procedures) is not implemented.
- The types `Clob` or `Blob` are not completely implemented, they need more work. The types `byte[]` and `String` can be used for `bytea` and `text` respectively.

Exception Handling

You can catch and handle an exception in the Greenplum Database backend just like any other exception. The backend `ErrorData` structure is exposed as a property in a class called `org.postgresql.pljava.ServerException` (derived from `java.sql.SQLException`) and the Java try/catch mechanism is synchronized with the backend mechanism.

**Important:** You will not be able to continue executing backend functions until your function has returned and the error has been propagated when the backend has generated an exception unless you have used a savepoint. When a savepoint is rolled back, the exceptional condition is reset and you can continue your execution.

Savepoints

Greenplum Database savepoints are exposed using the `java.sql.Connection` interface. Two restrictions apply.

- A savepoint must be rolled back or released in the function where it was set.
- A savepoint must not outlive the function where it was set.
**Logging**

PL/Java uses the standard Java Logger. Hence, you can write things like:

```java
Logger.getAnonymousLogger().info("Time is " + new Date(System.currentTimeMillis()));
```

At present, the logger uses a handler that maps the current state of the Greenplum Database configuration setting `log_min_messages` to a valid Logger level and that outputs all messages using the Greenplum Database backend function `elog()`.

**Note:** The `log_min_messages` setting is read from the database the first time a PL/Java function in a session is executed. On the Java side, the setting does not change after the first PL/Java function execution in a specific session until the Greenplum Database session that is working with PL/Java is restarted.

The following mapping apply between the Logger levels and the Greenplum Database backend levels.

<table>
<thead>
<tr>
<th>java.util.logging.Level</th>
<th>Greenplum Database Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEVERE ERROR</td>
<td>ERROR</td>
</tr>
<tr>
<td>WARNING</td>
<td>WARNING</td>
</tr>
<tr>
<td>CONFIG</td>
<td>LOG</td>
</tr>
<tr>
<td>INFO</td>
<td>INFO</td>
</tr>
<tr>
<td>FINE</td>
<td>DEBUG1</td>
</tr>
<tr>
<td>FINER</td>
<td>DEBUG2</td>
</tr>
<tr>
<td>FINEST</td>
<td>DEBUG3</td>
</tr>
</tbody>
</table>

**Security**

- **Installation**
- **Trusted Language**

**Installation**

Only a database superuser can install PL/Java. The PL/Java utility functions are installed using SECURITY DEFINER so that they execute with the access permissions that where granted to the creator of the functions.

**Trusted Language**

PL/Java is a trusted language. The trusted PL/Java language has no access to the file system as stipulated by PostgreSQL definition of a trusted language. Any database user can create and access functions in a trusted language.

PL/Java also installs a language handler for the language `javau`. This version is not trusted and only a superuser can create new functions that use it. Any user can call the functions.
Some PL/Java Issues and Solutions

When writing the PL/Java, mapping the JVM into the same process-space as the Greenplum Database backend code, some concerns have been raised regarding multiple threads, exception handling, and memory management. Here are brief descriptions explaining how these issues were resolved.

- Multi-threading
- Exception Handling
- Java Garbage Collector Versus `palloc()` and Stack Allocation

Multi-threading
Java is inherently multi-threaded. The Greenplum Database backend is not. There is nothing stopping a developer from utilizing multiple Threads class in the Java code. Finalizers that call out to the backend might have been spawned from a background Garbage Collection thread. Several third party Java-packages that are likely to be used make use of multiple threads. How can this model coexist with the Greenplum Database backend in the same process?

Solution
The solution is simple. PL/Java defines a special object called the Backend.THREADLOCK. When PL/Java is initialized, the backend immediately grabs this objects monitor (i.e. it will synchronize on this object). When the backend calls a Java function, the monitor is released and then immediately regained when the call returns. All calls from Java out to backend code are synchronized on the same lock. This ensures that only one thread at a time can call the backend from Java, and only at a time when the backend is awaiting the return of a Java function call.

Exception Handling
Java makes frequent use of try/catch/finally blocks. Greenplum Database sometimes use an exception mechanism that calls `longjmp` to transfer control to a known state. Such a jump would normally effectively bypass the JVM.

Solution
The backend now allows errors to be caught using the macros `PG_TRY/PG_CATCH/PG_END_TRY` and in the catch block, the error can be examined using the ErrorData structure. PL/Java implements a `java.sql.SQLException` subclass called `org.postgresql.pljava.ServerException`. The ErrorData can be retrieved and examined from that exception. A catch handler is allowed to issue a rollback to a savepoint. After a successful rollback, execution can continue.

Java Garbage Collector Versus `palloc()` and Stack Allocation

Primitive types are always passed by value. This includes the `String` type (this is a must since Java uses double byte characters). Complex types are often wrapped in Java objects and passed by reference. For example, a Java object can contain a pointer to a `palloc`ed or stack allocated memory and use native JNI calls to extract and manipulate data. Such data will become stale once a call has ended. Further attempts to access such data will at best give very unpredictable results but more likely cause a memory fault and a crash.

Solution
The PL/Java contains code that ensures that stale pointers are cleared when the MemoryContext or stack where they where allocated goes out of scope. The Java wrapper objects might live on but any attempt to use them will result in a stale native handle exception.
**Example**

The following simple Java example creates a JAR file that contains a single method and runs the method.

**Note:** The example requires Java SDK to compile the Java file.

The following method returns a substring.

```java
public static String substring(String text, int beginIndex, int endIndex)
{
    return text.substring(beginIndex, endIndex);
}
```

Enter the java code in a text file `example.class`.

Contents of the file `manifest.txt`:

```ini
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
```

Compile the java code:

```
javac *.java
```

Create a JAR archive named `analytics.jar` that contains the class file and the manifest file MANIFEST file in the JAR.

```
jar cfm analytics.jar manifest.txt *.class
```

Upload the jar file to the Greenplum master host.

Run the `gpscp` utility to copy the jar file to the Greenplum Java directory. Use the `-f` option to specify the file that contains a list of the master and segment hosts.

```
gpscp -f gphosts_file analytics.jar
=:/usr/local/greenplum-db/lib/postgresql/java/
```

Use the `gpconfig` utility to set the Greenplum `pljava_classpath` server configuration parameter. The parameter lists the installed jar files.

```
gpconfig -c pljava_classpath -v 'analytics.jar'
```

Run the `gpstop` utility with the `-u` option to reload the configuration files.

```
gpstop -u
```

From the `psql` command line, run the following command to show the installed jar files.

```
show pljava_classpath
```
The following SQL commands create a table and define a Java function to test the method in the jar file:

```sql
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;
```

You can place the contents in a file, `mysample.sql` and run the command from a `psql` command line:

```bash
> \i mysample.sql
```

The output is similar to this:

```
java_substring
-----------
y st
(1 row)
```

**References**

The PL/Java Github wiki page - [https://github.com/tada/pljava/wiki](https://github.com/tada/pljava/wiki).

PL/Java 1.4.0 release - [https://github.com/tada/pljava/tree/B1_4](https://github.com/tada/pljava/tree/B1_4).
Greenplum PL/Perl Language Extension

This chapter includes the following information:

• About Greenplum PL/Perl
• Greenplum Database PL/Perl Limitations
• Trusted/Untrusted Language
• Developing Functions with PL/Perl

About Greenplum PL/Perl

With the Greenplum Database PL/Perl extension, you can write user-defined functions in Perl that take advantage of its advanced string manipulation operators and functions. PL/Perl provides both trusted and untrusted variants of the language.

PL/Perl is embedded in your Greenplum Database distribution. Greenplum Database PL/Perl requires Perl to be installed on the system of each database host.

Refer to the PostgreSQL PL/Perl documentation for additional information.

Greenplum Database PL/Perl Limitations

Limitations of the Greenplum Database PL/Perl language include:

• Greenplum Database does not support PL/Perl triggers.
• PL/Perl functions cannot call each other directly.
• SPI is not yet fully implemented.
• If you fetch very large data sets using spi_exec_query(), you should be aware that these will all go into memory. You can avoid this problem by using spi_query()/spi_fetchrow(). A similar problem occurs if a set-returning function passes a large set of rows back to Greenplum Database via a return statement. Use return_next for each row returned to avoid this problem.
• When a session ends normally, not due to a fatal error, PL/Perl executes any END blocks that you have defined. No other actions are currently performed. (File handles are not automatically flushed and objects are not automatically destroyed.)

Trusted/Untrusted Language

PL/Perl includes trusted and untrusted language variants.

The PL/Perl trusted language is named plperl. The trusted PL/Perl language restricts file system operations, as well as require, use, and other statements that could potentially interact with the operating system or database server process. With these restrictions in place, any Greenplum Database user can create and execute functions in the trusted plperl language.

The PL/Perl untrusted language is named plperlu. You cannot restrict the operation of functions you create with the plperlu untrusted language. Only database superusers have privileges to create untrusted PL/Perl user-defined functions. And only database superusers and other database users that are explicitly granted the permissions can execute untrusted PL/Perl user-defined functions.

PL/Perl has limitations with respect to communication between interpreters and the number of interpreters running in a single process. Refer to the PostgreSQL Trusted and Untrusted PL/Perl documentation for additional information.
Enabling and Removing PL/Perl Support

You must register the PL/Perl language with a database before you can create and execute a PL/Perl user-defined function within that database. To remove PL/Perl support, you must explicitly remove the extension from each database in which it was registered. You must be a database superuser or owner to register or remove trusted languages in Greenplum databases.

Note: Only database superusers may register or remove support for the untrusted PL/Perl language plperlu.

Before you enable or remove PL/Perl support in a database, ensure that:

- Your Greenplum Database is running.
- You have sourced greenplum_path.sh.
- You have set the $MASTER_DATA_DIRECTORY and $GPHOME environment variables.

Enabling PL/Perl Support

For each database in which you want to enable PL/Perl, register the language using the SQL CREATE LANGUAGE command or the Greenplum Database createlang utility. For example, run the following command as the gpadmin user to register the trusted PL/Perl language for the database named testdb:

```
$ createlang plperl -d testdb
```

Removing PL/Perl Support

To remove support for PL/Perl from a database, run the SQL DROP LANGUAGE command or the Greenplum Database droplang utility. For example, run the following command as the gpadmin user to remove support for the trusted PL/Perl language from the database named testdb:

```
$ psql -d testdb
psql (8.3.23)
Type "help" for help.

  testdb=# DROP LANGUAGE plperl;
```

The default DROP LANGUAGE behavior is RESTRICT; the command fails if any existing objects (such as functions) depend on the language. Specify the CASCADE clause to also drop all dependent objects, including functions that you created with PL/Perl.

Developing Functions with PL/Perl

You define a PL/Perl function using the standard SQL CREATE FUNCTION syntax. The body of a PL/Perl user-defined function is ordinary Perl code. The PL/Perl interpreter wraps this code inside a Perl subroutine.

You can also create an anonymous code block with PL/Perl. An anonymous code block, called with the SQL DO command, receives no arguments, and whatever value it might return is discarded. Otherwise, a PL/Perl anonymous code block behaves just like a function. Only database superusers create an anonymous code block with the untrusted plperlu language.

The syntax of the CREATE FUNCTION command requires that you write the PL/Perl function body as a string constant. While it is more convenient to use dollar-quoting, you can choose to use escape string syntax (E' ' ) provided that you double any single quote marks and backslashes used in the body of the function.
PL/Perl arguments and results are handled as they are in Perl. Arguments you pass in to a PL/Perl function are accessed via the @_ array. You return a result value with the return statement, or as the last expression evaluated in the function. A PL/Perl function cannot directly return a non-scalar type because you call it in a scalar context. You can return non-scalar types such as arrays, records, and sets in a PL/Perl function by returning a reference.

PL/Perl treats null argument values as "undefined". Adding the STRICT keyword to the LANGUAGE subclause instructs Greenplum Database to immediately return null when any of the input arguments are null. When created as STRICT, the function itself need not perform null checks.

The following PL/Perl function utilizes the STRICT keyword to return the greater of two integers, or null if any of the inputs are 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 considers anything in a function argument that is not a reference to be a string, the standard Greenplum Database external text representation. The argument values supplied to a PL/Perl function are simply the input arguments converted to text form (just as if they had been displayed by a SELECT statement). In cases where the function argument is not an ordinary numeric or text type, you must convert the Greenplum Database type to a form that is more usable by Perl. Conversely, the return and return_next statements accept any string that is an acceptable input format for the function’s declared return type.

Refer to the PostgreSQL PL/Perl Functions and Arguments documentation for additional information, including composite type and result set manipulation.

### Built-in PL/Perl Functions

PL/Perl includes built-in functions to access the database, including those to prepare and perform queries and manipulate query results. The language also includes utility functions for error logging and string manipulation.

The following example creates a simple table with an integer and a text column. It creates a PL/Perl user-defined function that takes an input string argument and invokes the spi_exec_query() built-in function to select all columns and rows of the table. The function returns all rows in the query results where the v column includes the function input string.

```
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;

SELECT return_match( 'iff' );
  return_match
     --------------
      (4,different)
       (1 row)

Refer to the PostgreSQL PL/Perl Built-in Functions documentation for a detailed discussion of available functions.

Global Values in PL/Perl

You can use the global hash map %_SHARED to share data, including code references, between PL/Perl function calls for the lifetime of the current session.

The following example uses %_SHARED to share data between the user-defined set_var() and get_var() PL/Perl functions:

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)

For security reasons, PL/Perl creates a separate Perl interpreter for each role. This prevents accidental or malicious interference by one user with the behavior of another user's PL/Perl functions. Each such interpreter retains its own value of the %_SHARED variable and other global state. Two PL/Perl functions share the same value of %_SHARED if and only if they are executed by the same SQL role.

There are situations where you must take explicit steps to ensure that PL/Perl functions can share data in %_SHARED. For example, if an application executes under multiple SQL roles (via SECURITY DEFINER functions, use of SET ROLE, etc.) in a single session, make sure that functions that need to communicate are owned by the same user, and mark these functions as SECURITY DEFINER.

Notes

Additional considerations when developing PL/Perl functions:

- PL/Perl internally utilizes the UTF-8 encoding. It converts any arguments provided in other encodings to UTF-8, and converts return values from UTF-8 back to the original encoding.
- Nesting named PL/Perl subroutines retains the same dangers as in Perl.
- Only the untrusted PL/Perl language variant supports module import. Use plperlu with care.
- Any module that you use in a plperlu function must be available from the same location on all Greenplum Database hosts.
Greenplum MADlib Extension for Analytics

This chapter includes the following information:

- About MADlib
- Installing MADlib
- Upgrading MADlib
- Uninstalling MADlib
- Examples
- References

About MADlib

MADlib is an open-source library for scalable in-database analytics. With the Greenplum Database MADlib extension, you can use MADlib functionality in a Greenplum Database.

MADlib provides data-parallel implementations of mathematical, statistical and machine-learning methods for structured and unstructured data. It provides an suite of SQL-based algorithms for machine learning, data mining and statistics that run at scale within a database engine, with no need for transferring data between Greenplum Database and other tools.

MADlib requires the m4 macro processor version 1.4.13 or later.

MADlib can be used with PivotalR, an R package that enables users to interact with data resident in Greenplum Database using the R client. See About MADlib, R, and PivotalR.

Note: When using MADlib 1.11 and earlier, set the server configuration parameter optimizer_control to on (the default). If the parameter is set to off, these MADlib functions do not work: decision tree, random forest, LDA, k-Means, PMML export for decision tree, PMML export for random forest. The installation checks for these functions also fail.

The parameter optimizer_control controls whether the server configuration parameter optimizer can be changed. The parameter optimizer controls whether the GPORCA optimizer is enabled when running SQL queries. Some MADlib installation checks and functions change the value of optimizer to improve performance. If optimizer_control is set to off, the value of optimizer cannot be changed and the function fails.

Installing MADlib

To install MADlib on Greenplum Database, you first install a compatible Greenplum MADlib package and then install the MADlib function libraries on all databases that will use MADlib.

The gppkg utility installs Greenplum Database extensions, along with any dependencies, on all hosts across a cluster. It also automatically installs extensions on new hosts in the case of system expansion segment recovery.

Installing the Greenplum Database MADlib Package

Before you install the MADlib package, make sure that your Greenplum database is running, you have sourced greenplum_path.sh, and that the $MASTER_DATA_DIRECTORY and $GPHOME variables are set.

1. Download the MADlib extension package from Pivotal Network.
2. Copy the MADlib package to the Greenplum Database master host.
3. Install the software package by running the `gppkg` command. This example installs the MADlib 1.11 package on a Linux system:

```
$ gppkg -i madlib-ossv1.11_pvl.9.8_gpdb<version>-rhel6-x86_64.gppkg
```

### Adding MADlib Functions to a Database

After installing the MADlib package, run the `madpack` command to add MADlib functions to Greenplum Database. `madpack` is in `$GPHOME/madlib/bin`.

```
$ madpack [-s schema_name] -p greenplum -c user@host:port/database install
```

For example, this command creates MADlib functions in the Greenplum database `testdb` running on server `mdw` on port `5432`. The `madpack` command logs in as the user `gpadmin` and prompts for password. The target schema is `madlib`.

```
$ madpack -s madlib -p greenplum -c gpadmin@mdw:5432/testdb install
```

After installing the functions, The Greenplum Database `gpadmin` superuser role should grant all privileges on the target schema (in the example `madlib`) to users who will be accessing MADlib functions. Users without access to the functions will get the error `ERROR: permission denied for schema MADlib`. The `madpack install-check` option runs test using Madlib modules to check the MADlib installation:

```
$ madpack -s madlib -p greenplum -c gpadmin@mdw:5432/testdb install-check
```

**Note:** The command `madpack -h` displays information for the utility.

### Upgrading MADlib

You upgrade an installed MADlib package with the Greenplum Database `gppkg` utility and the MADlib `madpack` command.

#### Upgrading a MADlib Package

These steps upgrade an installed MADlib package to MADlib 1.11.

1. Run the `gppkg` utility with the `-u` option. This command upgrades an installed MADlib package to MADlib 1.11.

```
gppkg -u madlib-ossv1.11_pvl.9.8_gpdb<version>-rhel6-x86_64.gppkg
```

#### Upgrading MADlib Functions

After you upgrade the MADlib package, you run the `madpack` command to upgrade the MADlib functions in Greenplum Database. For this example command, the MADlib functions are installed in the schema `madlib` of the Greenplum Database `test`. This command upgrades the MADlib functions in the database `schema`:

```
madpack -s madlib -p greenplum -c gpadmin@mdw:5432/testdb upgrade
```

### Uninstalling MADlib

- Remove MADlib objects from the database
- Uninstall the Greenplum Database MADlib Package
When you remove MADlib support from a database, routines that you created in the database that use MADlib functionality will no longer work.

**Remove MADlib objects from the database**

Use the `madpack uninstall` command to remove MADlib objects from a Greenplum database. For example, this command removes MADlib objects from the database `testdb`.

```bash
$ madpack -s madlib -p greenplum -c gpadmin@mdw:5432/testdb uninstall
```

**Uninstall the Greenplum Database MADlib Package**

If no databases use the MADlib functions, use the Greenplum `gppkg` utility with the `-r` option to uninstall the MADlib package. When removing the package you must specify the package and version. This example uninstalls MADlib package version 1.9.

```bash
$ gppkg -r madlib-ossvl.11_pvl.9.8_gpdb<version>-rhel6-x86_64.gppkg
```

You can run the `gppkg` utility with the options `-q --all` to list the installed extensions and their versions. After you uninstall the package, restart the database.

```bash
$ gpstop -r
```

**Examples**

Following are examples using the Greenplum MADlib extension:

- **Linear Regression**
- **Association Rules**
- **Naive Bayes Classification**

See the MADlib documentation for additional examples.

**Linear Regression**

This example runs a linear regression on the table `regr_example`. The dependent variable data are in the `y` column and the independent variable data are in the `x1` and `x2` columns.

The following statements create the `regr_example` table and load some sample data:

```sql
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);
```

The MADlib `linregr_train()` function produces a regression model from an input table containing training data. The following `SELECT` statement runs a simple multivariate regression on the `regr_example` table and saves the model in the `reg_example_model` table.

```sql
SELECT madlib.linregr_train (  
```
The `madlib.linregr_train()` function can have additional arguments to set grouping columns and to calculate the heteroskedasticity of the model.

**Note:** The intercept is computed by setting one of the independent variables to a constant 1, as shown in the preceding example.

Running this query against the `regr_example` table creates the `regr_example_model` table with one row of data:

```sql
SELECT * FROM regr_example_model;
```

<table>
<thead>
<tr>
<th>RECORD 1</th>
<th>------------------------+------------------------</th>
</tr>
</thead>
<tbody>
<tr>
<td>coef</td>
<td>{0.111111111111127,1.14814814814815,1.01851851851852}</td>
</tr>
<tr>
<td>r2</td>
<td>0.968612680477111</td>
</tr>
<tr>
<td>std_err</td>
<td>{1.49587911309236,0.20704331249903,0.346449758034495}</td>
</tr>
<tr>
<td>t_stats</td>
<td>{0.0742781352708591,5.54544858420156,2.93987366103776}</td>
</tr>
<tr>
<td>p_values</td>
<td>{0.952799748147436,0.113579771006374,0.208730790695278}</td>
</tr>
<tr>
<td>condition_no</td>
<td>22.650203241881</td>
</tr>
<tr>
<td>num_rows_processed</td>
<td>4</td>
</tr>
<tr>
<td>num_missing_rows_skipped</td>
<td>0</td>
</tr>
<tr>
<td>variance_covariance</td>
<td>{(2.23765432098598,-0.257201646090342,-0.437242798353582), (-0.257201646090342,0.042866941015057,0.0342935528120456), (-0.437242798353582,0.0342935528120456,0.12002743484216)}</td>
</tr>
</tbody>
</table>

The model saved in the `regr_example_model` table can be used with the MADlib linear regression prediction function, `madlib.linregr_predict()`, to view the residuals:

```sql
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;
```

|----+----+----+------------------+------------------|
| i  | y  | x1 | x2 | predict          | residual         |
|----+----+----+------------------+------------------|
| 1  | 5  | 2  | 3  | 5.462962962962971 | -0.462962962962971|
| 3  | 6  | 4  | 1  | 5.722222222222222 | 0.2777777777777762|
| 2  | 10 | 7  | 2  | 10.1851851851852 | -0.185185185185201|
| 4  | 8  | 3  | 4  | 7.62962962962964 | 0.370370370370364|

(4 rows)

**Association Rules**

This example demonstrates the association rules data mining technique on a transactional data set. Association rule mining is a technique for discovering relationships between variables in a large data set. This example considers items in a store that are commonly purchased together. In addition to market basket analysis, association rules are also used in bioinformatics, web analytics, and other fields.

The example analyzes purchase information for seven transactions that are stored in a table with the MADlib function `MADlib.assoc_rules`. The function assumes that the data is stored in two columns with a single item and transaction ID per row. Transactions with multiple items consist of multiple rows with one row per item.
These commands create the table.

```
DROP TABLE IF EXISTS test_data;
CREATE TABLE test_data (  
    trans_id INT,  
    product text
);
```

This INSERT command adds the data to the table.

```
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');
```

The MADlib function `madlib.assoc_rules()` analyzes the data and determines association rules with the following characteristics.

- A support value of at least .40. Support is the ratio of transactions that contain X to all transactions.
- A confidence value of at least .75. Confidence is the ratio of transactions that contain X to transactions that contain Y. One could view this metric as the conditional probability of X given Y.

This SELECT command determines association rules, creates the table `assoc_rules`, and adds the statistics to the table.

```
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
```

This is the output of the SELECT command. There are two rules that fit the characteristics.

```
output_schema | output_table | total_rules | total_time
--------------+--------------+-------------+-----------------
public        | assoc_rules  |           2 | 00:00:01.153283
(1 row)
```

To view the association rules, you can run this SELECT command.

```
SELECT pre, post, support FROM assoc_rules
    ORDER BY support DESC;
```
This is the output. The pre and post columns are the itemsets of left and right hand sides of the association rule respectively.

<table>
<thead>
<tr>
<th>pre</th>
<th>post</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{diapers}</td>
<td>{beer}</td>
<td>0.714285714285714</td>
</tr>
<tr>
<td>{chips}</td>
<td>{beer}</td>
<td>0.428571428571429</td>
</tr>
</tbody>
</table>

(2 rows)

Based on the data, beer and diapers are often purchased together. To increase sales, you might consider placing beer and diapers closer together on the shelves.

**Naive Bayes Classification**

Naive Bayes analysis predicts the likelihood of an outcome of a class variable, or category, based on one or more independent variables, or attributes. The class variable is a non-numeric categorial variable, a variable that can have one of a limited number of values or categories. The class variable is represented with integers, each integer representing a category. For example, if the category can be one of "true", "false", or "unknown," the values can be represented with the integers 1, 2, or 3.

The attributes can be of numeric types and non-numeric, categorical, types. The training function has two signatures – one for the case where all attributes are numeric and another for mixed numeric and categorical types. Additional arguments for the latter identify the attributes that should be handled as numeric values. The attributes are submitted to the training function in an array.

The MADlib Naive Bayes training functions produce a features probabilities table and a class priors table, which can be used with the prediction function to provide the probability of a class for the set of attributes.

**Naive Bayes Example 1 - Simple All-numeric Attributes**

In the first example, the class variable is either 1 or 2 and there are three integer attributes.

1. The following commands create the input table and load sample data.

```sql
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}');
```

Actual data in production scenarios is more extensive than this example data and yields better results. Accuracy of classification improves significantly with larger training data sets.

2. Train the model with the create_nb_prepared_data_tables() function.

```sql
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. Create a table with data to classify using the model.

```sql
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. Create a classification view using the feature probabilities, class priors, and class_example_topredict tables.

```sql
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. Display the classification results.

```sql
SELECT * FROM example_classified;
```

<table>
<thead>
<tr>
<th>key</th>
<th>class</th>
<th>nb_prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0.6</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.75</td>
</tr>
</tbody>
</table>

(6 rows)

**Naive Bayes Example 2 – Weather and Outdoor Sports**

This example calculates the probability that the user will play an outdoor sport, such as golf or tennis, based on weather conditions.

The table `weather_example` contains the example values.

The identification column for the table is `day`, an integer type.

The `play` column holds the dependent variable and has two classifications:
- 0 - No
- 1 - Yes

There are four attributes: outlook, temperature, humidity, and wind. These are categorical variables. The `MADlib create_nb_classify_view()` function expects the attributes to be provided as an array of `INTEGER`, `NUMERIC`, or `FLOAT8` values, so the attributes for this example are encoded with integers as follows:
- `outlook` may be sunny (1), overcast (2), or rain (3).
- `temperature` may be hot (1), mild (2), or cool (3).
- `humidity` may be high (1) or normal (2).
- `wind` may be strong (1) or weak (2).

The following table shows the training data, before encoding the variables.

<table>
<thead>
<tr>
<th>day</th>
<th>play</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>No</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
</tr>
</tbody>
</table>
1. Create the training table.

```sql
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, 1, '{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, '{1,2,1,1}'),  
(13, 1, '{2,1,2,2}');
```

2. Create the model from the training table.

```sql
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. View the feature probabilities:

```sql
SELECT * FROM weather_probs;
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
```
4. To classify a group of records with a model, first load the data into a table. In this example, the table `t1` has four rows to classify.

    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}');

5. Use the MADlib `create_nb_classify_view()` function to classify the rows in the table.

    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  
    );

    The result is four rows, one for each record in the `t1` table.

    SELECT * FROM t1_out ORDER BY key;
    key | nb_classification
    ----+-------------------
    1 | {0}
    2 | {1}
    3 | {1}
    4 | {0}
    (4 rows)
PivotalR allows R users to leverage the scalability and performance of in-database analytics without leaving the R command line. The computational work is executed in-database, while the end user benefits from the familiar R interface. Compared with respective native R functions, there is an increase in scalability and a decrease in running time. Furthermore, data movement, which can take hours for very large data sets, is eliminated with PivotalR.

Key features of the PivotalR package:

- Explore and manipulate data in the database with R syntax. SQL translation is performed by PivotalR.
- Use the familiar R syntax for predictive analytics algorithms, for example linear and logistic regression. PivotalR accesses the MADlib in-database analytics function calls.
- Comprehensive documentation package with examples in standard R format accessible from an R client.
- The PivotalR package also supports access to the MADlib functionality.

For information about PivotalR, including supported MADlib functionality, see [https://cwiki.apache.org/confluence/display/MADLIB/PivotalR](https://cwiki.apache.org/confluence/display/MADLIB/PivotalR).

The R package for PivotalR can be found at [https://cran.r-project.org/web/packages/PivotalR/index.html](https://cran.r-project.org/web/packages/PivotalR/index.html).
Greenplum Fuzzy String Match Extension

The Greenplum Database Fuzzy String Match extension provides functions to determine similarities and distance between strings based on various algorithms.

- Soundex Functions
- Levenshtein Functions
- Metaphone Functions
- Double Metaphone Functions
- Installing and Uninstalling the Fuzzy String Match Functions

The Greenplum Database installation contains the files required for the functions in this extension module and SQL scripts to define the extension functions in a database and remove the functions from a database.

Warning: The functions soundex, metaphone, dmetaphone, and dmetaphone_alt do not work well with multibyte encodings (such as UTF-8).

The Greenplum Database Fuzzy String Match extension is based on the PostgreSQL fuzzystrmatch module.

Soundex Functions

The Soundex system is a method of matching similar-sounding (similar phonemes) names by converting them to the same code.

Note: Soundex is most useful for English names.

These functions work with Soundex codes:

- `soundex(text string1)` returns text
- `difference(text string1, text string2)` returns int

The `soundex` function converts a string to its Soundex code. Soundex codes consist of four characters.

The `difference` function converts two strings to their Soundex codes and then reports the number of matching code positions. The result ranges from zero to four, zero being no match and four being an exact match. These are some examples:

```sql
SELECT soundex('hello world!');
SELECT soundex('Anne'), soundex('Ann'), difference('Anne', 'Ann');
SELECT soundex('Anne'), soundex('Andrew'), difference('Anne', 'Andrew');
SELECT soundex('Anne'), soundex('Margaret'), difference('Anne', 'Margaret');

CREATE TABLE s (nm text);
INSERT INTO s VALUES ('john');
INSERT INTO s VALUES ('joan');
INSERT INTO s VALUES ('wobbly');
INSERT INTO s VALUES ('jack');

SELECT * FROM s WHERE soundex(nm) = soundex('john');
SELECT * FROM s WHERE difference(s.nm, 'john') > 2;
```

For information about the Soundex indexing system see https://www.archives.gov/research/census/soundex.html.
**Levenshtein Functions**

These functions calculate the Levenshtein distance between two strings:

```
levenshtein(text source, text target, int ins_cost, int del_cost, int sub_cost) returns int
levenshtein(text source, text target) returns int
levenshtein_less_equal(text source, text target, int ins_cost, int del_cost, int sub_cost, int max_d) returns int
levenshtein_less_equal(text source, text target, int max_d) returns int
```

Both the `source` and `target` parameters can be any non-null string, with a maximum of 255 bytes. The cost parameters `ins_cost`, `del_cost`, and `sub_cost` specify cost of a character insertion, deletion, or substitution, respectively. You can omit the cost parameters, as in the second version of the function; in that case the cost parameters default to 1.

`levenshtein_less_equal` is an accelerated version of `levenshtein` function for low values of distance. If actual distance is less or equal then `max_d`, then `levenshtein_less_equal` returns an accurate value of the distance. Otherwise, this function returns value which is greater than `max_d`. Examples:

```
test=# SELECT levenshtein('GUMBO', 'GAMBOL');
levenshtein
---------
  2
(1 row)
test=# SELECT levenshtein('GUMBO', 'GAMBOL', 2,1,1);
levenshtein
---------
  3
(1 row)
test=# SELECT levenshtein_less_equal('extensive', 'exhaustive',2);
levenshtein_less_equal
-----------------------
  3
(1 row)
test=# SELECT levenshtein_less_equal('extensive', 'exhaustive',4);
levenshtein_less_equal
-----------------------
  4
(1 row)
```

For information about the Levenshtein algorithm, see [http://www.levenshtein.net/](http://www.levenshtein.net/).

**Metaphone Functions**

Metaphone, like Soundex, is based on the idea of constructing a representative code for an input string. Two strings are then deemed similar if they have the same codes. This function calculates the metaphone code of an input string:

```
metaphone(text source, int max_output_length) returns text
```

The `source` parameter must be a non-null string with a maximum of 255 characters. The `max_output_length` parameter sets the maximum length of the output metaphone code; if longer, the output is truncated to this length. Example:

```
test=# SELECT metaphone('GUMBO', 4);
```

metaphone
---------
KM
(1 row)

For information about the Metaphone algorithm, see https://en.wikipedia.org/wiki/Metaphone.

**Double Metaphone Functions**

The Double Metaphone system computes two "sounds like" strings for a given input string - a "primary" and an "alternate". In most cases they are the same, but for non-English names especially they can be a bit different, depending on pronunciation. These functions compute the primary and alternate codes:

```
dmetaphone(text source) returns text
```

```
dmetaphone_alt(text source) returns text
```

There is no length limit on the input strings. Example:

```
test=# select dmetaphone('gumbo');
dmetaphone
---------
KMP
(1 row)
```

For information about the Double Metaphone algorithm, see https://en.wikipedia.org/wiki/Metaphone#Double_Metaphone.

**Installing and Uninstalling the Fuzzy String Match Functions**

Greenplum Database supplies SQL scripts to install and uninstall the Fuzzy String Match extension functions.

To install the functions in a database, run the following SQL script:

```
psql -f $GPHOME/share/postgresql/contrib/fuzzystrmatch.sql
```

To uninstall the functions, run the following SQL script:

```
psql -f $GPHOME/share/postgresql/contrib/uninstall_fuzzystrmatch.sql
```

*Note:* When you uninstall the Fuzzy String Match functions from a database, routines that you created in the database that use the functions will no longer work.
Summary of Greenplum Features

This section provides a high-level overview of the system requirements and feature set of Greenplum Database. It contains the following topics:

- Greenplum SQL Standard Conformance
- Greenplum and PostgreSQL Compatibility

Greenplum SQL Standard Conformance

The SQL language was first formally standardized in 1986 by the American National Standards Institute (ANSI) as SQL 1986. Subsequent versions of the SQL standard have been released by ANSI and as International Organization for Standardization (ISO) standards: SQL 1989, SQL 1992, SQL 1999, SQL 2003, SQL 2006, and finally SQL 2008, which is the current SQL standard. The official name of the standard is ISO/IEC 9075-1:2008. In general, each new version adds more features, although occasionally features are deprecated or removed.

It is important to note that there are no commercial database systems that are fully compliant with the SQL standard. Greenplum Database is almost fully compliant with the SQL 1992 standard, with most of the features from SQL 1999. Several features from SQL 2003 have also been implemented (most notably the SQL OLAP features).

This section addresses the important conformance issues of Greenplum Database as they relate to the SQL standards. For a feature-by-feature list of Greenplum’s support of the latest SQL standard, see SQL 2008 Optional Feature Compliance.

Core SQL Conformance

In the process of building a parallel, shared-nothing database system and query optimizer, certain common SQL constructs are not currently implemented in Greenplum Database. The following SQL constructs are not supported:

1. Some set returning subqueries in EXISTS or NOT EXISTS clauses that Greenplum’s parallel optimizer cannot rewrite into joins.
2. Backwards scrolling cursors, including the use of FETCH PRIOR, FETCH FIRST, FETCH ABSOLUTE, and FETCH RELATIVE.
3. In CREATE TABLE statements (on hash-distributed tables): a UNIQUE or PRIMARY KEY clause must include all of (or a superset of) the distribution key columns. Because of this restriction, only one UNIQUE clause or PRIMARY KEY clause is allowed in a CREATE TABLE statement. UNIQUE or PRIMARY KEY clauses are not allowed on randomly-distributed tables.
4. CREATE UNIQUE INDEX statements that do not contain all of (or a superset of) the distribution key columns. CREATE UNIQUE INDEX is not allowed on randomly-distributed tables.

   Note that UNIQUE INDEXES (but not UNIQUE CONSTRAINTS) are enforced on a part basis within a partitioned table. They guarantee the uniqueness of the key within each part or sub-part.
5. VOLATILE or STABLE functions cannot execute on the segments, and so are generally limited to being passed literal values as the arguments to their parameters.
6. Triggers are not supported since they typically rely on the use of VOLATILE functions.
7. Referential integrity constraints (foreign keys) are not enforced in Greenplum Database. Users can declare foreign keys and this information is kept in the system catalog, however.
8. Sequence manipulation functions CURRVAL and LASTVAL.

SQL 1992 Conformance

The following features of SQL 1992 are not supported in Greenplum Database:
1. NATIONAL CHARACTER (NCHAR) and NATIONAL CHARACTER VARYING (NVARCHAR). Users can declare the NCHAR and NVARCHAR types, however they are just synonyms for CHAR and VARCHAR in Greenplum Database.
2. CREATE ASSERTION statement.
3. INTERVAL literals are supported in Greenplum Database, but do not conform to the standard.
4. GET DIAGNOSTICS statement.
5. GRANT INSERT or UPDATE privileges on columns. Privileges can only be granted on tables in Greenplum Database.
6. GLOBAL TEMPORARY TABLES and LOCAL TEMPORARY TABLES. Greenplum TEMPORARY TABLES do not conform to the SQL standard, but many commercial database systems have implemented temporary tables in the same way. Greenplum temporary tables are the same as VOLATILE TABLES in Teradata.
7. UNIQUE predicate.
8. MATCH PARTIAL for referential integrity checks (most likely will not be implemented in Greenplum Database).

SQL 1999 Conformance

The following features of SQL 1999 are not supported in Greenplum Database:

1. Large Object data types: BLOB, CLOB, NCLOB. However, the BYTEA and TEXT columns can store very large amounts of data in Greenplum Database (hundreds of megabytes).
2. MODULE (SQL client modules).
3. CREATE PROCEDURE (SQL/PSM). This can be worked around in Greenplum Database by creating a FUNCTION that returns void, and invoking the function as follows:

   ```sql
   SELECT myfunc(args);
   ```

4. The PostgreSQL/Greenplum function definition language (PL/P Gulf) is a subset of Oracle's PL/SQL, rather than being compatible with the SQL/PSM function definition language. Greenplum Database also supports function definitions written in Python, Perl, Java, and R.
5. BIT and BIT VARYING data types (intentionally omitted). These were deprecated in SQL 2003, and replaced in SQL 2008.
6. Greenplum supports identifiers up to 63 characters long. The SQL standard requires support for identifiers up to 128 characters long.
7. Prepared transactions (PREPARE TRANSACTION, COMMIT PREPARED, ROLLBACK PREPARED). This also means Greenplum does not support XA Transactions (2 phase commit coordination of database transactions with external transactions).
8. CHARACTER SET option on the definition of CHAR() or VARCHAR() columns.
9. Specification of CHARACTERS or OCTETS (BYTES) on the length of a CHAR() or VARCHAR() column. For example, VARCHAR(15 CHARACTERS) or VARCHAR(15 OCTETS) or VARCHAR(15 BYTES).
10. CURRENT_SCHEMA function.
11. CREATE DISTINCT TYPE statement. CREATE DOMAIN can be used as a work-around in Greenplum.
12. The explicit table construct.

SQL 2003 Conformance

The following features of SQL 2003 are not supported in Greenplum Database:

1. MERGE statements.
2. IDENTITY columns and the associated GENERATED ALWAYS/GENERATED BY DEFAULT clause. The SERIAL or BIGSERIAL data types are very similar to INT or BIGINT GENERATED BY DEFAULT AS IDENTITY.
3. MULTISET modifiers on data types.
4. ROW data type.
5. Greenplum Database syntax for using sequences is non-standard. For example, `nextval('seq')` is used in Greenplum instead of the standard `NEXT VALUE FOR seq`.

6. `GENERATED ALWAYS AS` columns. Views can be used as a work-around.

7. The sample clause (TABLESAMPLE) on SELECT statements. The `random()` function can be used as a work-around to get random samples from tables.

8. The partitioned join tables construct (PARTITION BY in a join).

9. `GRANT` SELECT privileges on columns. Privileges can only be granted on tables in Greenplum Database. Views can be used as a work-around.

10. For `CREATE TABLE x (LIKE(y))` statements, Greenplum does not support the `[INCLUDING|EXCLUDING] [DEFAULTS|CONSTRAINTS|INDEXES]` clauses.

11. Greenplum array data types are almost SQL standard compliant with some exceptions. Generally customers should not encounter any problems using them.

### SQL 2008 Conformance

The following features of SQL 2008 are not supported in Greenplum Database:

1. `BINARY` and `VARBINARY` data types. `BYTEA` can be used in place of `VARBINARY` in Greenplum Database.

2. `FETCH FIRST` or `FETCH NEXT` clause for `SELECT`, for example:

   ```sql
   SELECT id, name FROM tab1 ORDER BY id OFFSET 20 ROWS FETCH NEXT 10 ROWS ONLY;
   ```

   Greenplum has `LIMIT` and `LIMIT OFFSET` clauses instead.

3. The `ORDER BY` clause is ignored in views and subqueries unless a `LIMIT` clause is also used. This is intentional, as the Greenplum optimizer cannot determine when it is safe to avoid the sort, causing an unexpected performance impact for such `ORDER BY` clauses. To work around, you can specify a really large `LIMIT`. For example: `SELECT * FROM mytable ORDER BY 1 LIMIT 9999999999`

4. The `row subquery` construct is not supported.

5. `TRUNCATE TABLE` does not accept the `CONTINUE IDENTITY` and `RESTART IDENTITY` clauses.

### Greenplum and PostgreSQL Compatibility

Greenplum Database is based on PostgreSQL 8.3 with additional features from newer PostgreSQL releases. To support the distributed nature and typical workload of a Greenplum Database system, some SQL commands have been added or modified, and there are a few PostgreSQL features that are not supported. Greenplum has also added features not found in PostgreSQL, such as physical data distribution, parallel query optimization, external tables, resource queues, and enhanced table partitioning.

For full SQL syntax and references, see the [SQL Command Reference](#).

### Table 236: SQL Support in Greenplum Database

<table>
<thead>
<tr>
<th>SQL Command</th>
<th>Supported in Greenplum</th>
<th>Modifications, Limitations, Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER AGGREGATE</td>
<td>YES</td>
<td>Changes the definition of a Greenplum Database extension - based on PostgreSQL 9.6.</td>
</tr>
<tr>
<td>ALTER CONVERSION</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ALTER DATABASE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ALTER DOMAIN</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ALTER EXTENSION</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>SQL Command</td>
<td>Supported in Greenplum</td>
<td>Modifications, Limitations, Exceptions</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>ALTER FILESPACE</td>
<td>YES</td>
<td>Greenplum Database parallel tablespace feature - not in PostgreSQL 8.3.</td>
</tr>
<tr>
<td>ALTER FUNCTION</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ALTER GROUP</td>
<td>YES</td>
<td>An alias for <strong>ALTER ROLE</strong></td>
</tr>
<tr>
<td>ALTER INDEX</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ALTER LANGUAGE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ALTER OPERATOR</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ALTER OPERATOR CLASS</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ALTER OPERATOR FAMILY</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ALTER PROTOCOL</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ALTER RESOURCE QUEUE</td>
<td>YES</td>
<td>Greenplum Database resource management feature - not in PostgreSQL.</td>
</tr>
<tr>
<td>ALTER ROLE</td>
<td>YES</td>
<td>Greenplum Database Clauses: RESOURCE QUEUE queue_name</td>
</tr>
<tr>
<td>ALTER SCHEMA</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ALTER SEQUENCE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ALTER TABLE</td>
<td>YES</td>
<td><strong>Unsupported Clauses / Options:</strong> CLUSTER ON ENABLE/DISABLE TRIGGER Greenplum Database Clauses: ADD</td>
</tr>
<tr>
<td>ALTER TABLESPACE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ALTER TRIGGER</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ALTER TYPE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ALTER USER</td>
<td>YES</td>
<td>An alias for <strong>ALTER ROLE</strong></td>
</tr>
<tr>
<td>ALTER VIEW</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ANALYZE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>BEGIN</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>SQL Command</td>
<td>Supported in Greenplum</td>
<td>Modifications, Limitations, Exceptions</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>CHECKPOINT</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>CLOSE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>CLUSTER</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>COMMENT</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>COMMIT PREPARED</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>COPY</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Modified Clauses:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESCAPE [ AS ] 'escape'</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Greenplum Database Clauses:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[LOG ERRORS] SEGMENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REJECT LIMIT count [ROWS</td>
</tr>
<tr>
<td>CREATE AGGREGATE</td>
<td>YES</td>
<td><strong>Unsupported Clauses / Options:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[ , SORTOP = sort_operator ]</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Greenplum Database Clauses:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[ , PREFUNC = prefunc ]</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Limitations:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The functions used to implement the aggregate must be IMMUTABLE functions.</td>
</tr>
<tr>
<td>CREATE CAST</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>CREATE CONSTRAINT TRIGGER</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>CREATE CONVERSION</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>CREATE DATABASE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>CREATE DOMAIN</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>CREATE EXTENSION</td>
<td>YES</td>
<td><strong>Loads a new extension into Greenplum Database - based on PostgreSQL 9.6.</strong></td>
</tr>
<tr>
<td>CREATE EXTERNAL TABLE</td>
<td>YES</td>
<td><strong>Greenplum Database parallel ETL feature - not in PostgreSQL 8.3.</strong></td>
</tr>
<tr>
<td>SQL Command</td>
<td>Supported in Greenplum</td>
<td>Modifications, Limitations, Exceptions</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>CREATE FUNCTION</td>
<td>YES</td>
<td>Limitations: Functions defined as <code>STABLE</code> or <code>VOLATILE</code> can be executed in Greenplum Database provided that they are executed on the master only. <code>STABLE</code> and <code>VOLATILE</code> functions cannot be used in statements that execute at the segment level.</td>
</tr>
<tr>
<td>CREATE GROUP</td>
<td>YES</td>
<td>An alias for <code>CREATE ROLE</code></td>
</tr>
<tr>
<td>CREATE INDEX</td>
<td>YES</td>
<td><strong>Greenplum Database Clauses:</strong> USING bitmap (bitmap indexes) Limitations: UNIQUE indexes are allowed only if they contain all of (or a superset of) the Greenplum distribution key columns. On partitioned tables, a unique index is only supported within an individual partition - not across all partitions. <code>CONCURRENTLY</code> keyword not supported in Greenplum.</td>
</tr>
<tr>
<td>CREATE LANGUAGE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>CREATE OPERATOR</td>
<td>YES</td>
<td>Limitations: The function used to implement the operator must be an <code>IMMUTABLE</code> function.</td>
</tr>
<tr>
<td>CREATE OPERATOR CLASS</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>CREATE OPERATOR FAMILY</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>CREATE PROTOCOL</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>CREATE RESOURCE QUEUE</td>
<td>YES</td>
<td>Greenplum Database resource management feature - not in PostgreSQL 8.3.</td>
</tr>
<tr>
<td>CREATE ROLE</td>
<td>YES</td>
<td><strong>Greenplum Database Clauses:</strong> <code>RESOURCE QUEUE queue_name</code></td>
</tr>
<tr>
<td>CREATE RULE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>CREATE SCHEMA</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>SQL Command</td>
<td>Supported in Greenplum</td>
<td>Modifications, Limitations, Exceptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>CREATE SEQUENCE</td>
<td>YES</td>
<td><strong>Limitations:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The \texttt{lastval()} and \texttt{currval()} functions are not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The \texttt{setval()} function is only allowed in queries that do not operate on distributed data.</td>
</tr>
<tr>
<td>CREATE TABLE</td>
<td>YES</td>
<td><strong>Unsupported Clauses / Options:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>\texttt{[GLOBAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Limited Clauses:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>\texttt{UNIQUE} or \texttt{PRIMARY KEY} constraints are only allowed on hash-distributed tables (\texttt{DISTRIBUTED BY}), and the constraint columns must be the same as or a superset of the distribution key columns of the table and must include all the distribution key columns of the partitioning key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Greenplum Database Clauses:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>\texttt{DISTRIBUTED BY (column, [ ... ] )}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\texttt{DISTRIBUTED RANDOMLY}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\texttt{PARTITION BY type (column [, ...]) (partition_specification, [...] )}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\texttt{WITH (appendonly=true [,compresslevel=value,blocksize=value] )}</td>
</tr>
<tr>
<td>CREATE TABLE AS</td>
<td>YES</td>
<td>See \texttt{CREATE TABLE}</td>
</tr>
<tr>
<td>CREATE TABLESPACE</td>
<td>NO</td>
<td><strong>Greenplum Database Clauses:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>\texttt{FILESPACE filespace_name}</td>
</tr>
<tr>
<td>CREATE TRIGGER</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>SQL Command</td>
<td>Supported in Greenplum</td>
<td>Modifications, Limitations, Exceptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>CREATE TYPE</td>
<td>YES</td>
<td>Limitations: The functions used to implement a new base type must be IMMUTABLE functions.</td>
</tr>
<tr>
<td>CREATE USER</td>
<td>YES</td>
<td>An alias for CREATE ROLE</td>
</tr>
<tr>
<td>CREATE VIEW</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DEALLOCATE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DECLARE</td>
<td>YES</td>
<td>Unsupported Clauses / Options: SCROLL FOR UPDATE [ OF column [, ... ] ] Limitations: Cursors cannot be backward-scrolled. Forward scrolling is supported. PL/pgSQL does not have support for updatable cursors.</td>
</tr>
<tr>
<td>DELETE</td>
<td>YES</td>
<td>Unsupported Clauses / Options: RETURNING</td>
</tr>
<tr>
<td>DISCARD</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DO</td>
<td>YES</td>
<td>PostgreSQL 9.0 feature</td>
</tr>
<tr>
<td>DROP AGGREGATE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DROP CAST</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DROP CONVERSION</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DROP DATABASE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DROP DOMAIN</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DROP EXTENSION</td>
<td>YES</td>
<td>Removes an extension from Greenplum Database – based on PostgreSQL 9.6.</td>
</tr>
<tr>
<td>DROP EXTERNAL TABLE</td>
<td>YES</td>
<td>Greenplum Database parallel ETL feature - not in PostgreSQL 8.3.</td>
</tr>
<tr>
<td>DROP FILESPACE</td>
<td>YES</td>
<td>Greenplum Database parallel tablespace feature - not in PostgreSQL 8.3.</td>
</tr>
<tr>
<td>DROP FUNCTION</td>
<td>YES</td>
<td>An alias for DROP ROLE</td>
</tr>
<tr>
<td>DROP GROUP</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>SQL Command</td>
<td>Supported in Greenplum</td>
<td>Modifications, Limitations, Exceptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>DROP INDEX</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DROP LANGUAGE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DROP OPERATOR</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DROP OPERATOR CLASS</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DROP OPERATOR FAMILY</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DROP OWNED</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>DROP PROTOCOL</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DROP RESOURCE QUEUE</td>
<td>YES</td>
<td>Greenplum Database resource management feature - not in PostgreSQL 8.3.</td>
</tr>
<tr>
<td>DROP ROLE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DROP RULE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DROP SCHEMA</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DROP SEQUENCE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DROP TABLE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DROP TABLESPACE</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>DROP TRIGGER</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>DROP TYPE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DROP USER</td>
<td>YES</td>
<td>An alias for <code>DROP ROLE</code></td>
</tr>
<tr>
<td>DROP VIEW</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>EXECUTE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>EXPLAIN</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>FETCH</td>
<td>YES</td>
<td><strong>Unsupported Clauses / Options:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LAST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PRIOR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BACKWARD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BACKWARD ALL</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Limitations:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cannot fetch rows in a nonsequential fashion; backward scan is not supported.</td>
</tr>
<tr>
<td>GRANT</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>SQL Command</td>
<td>Supported in Greenplum</td>
<td>Modifications, Limitations, Exceptions</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>INSERT</td>
<td>YES</td>
<td>Unsupported Clauses / Options:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RETURNING</td>
</tr>
<tr>
<td>LISTEN</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>LOAD</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>LOCK</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>MOVE</td>
<td>YES</td>
<td>See FETCH</td>
</tr>
<tr>
<td>NOTIFY</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>PREPARE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>PREPARE TRANSACTION</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>REASSIGN OWNED</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>REINDEX</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>RELEASE SAVEPOINT</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>RESET</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>REVOKE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ROLLBACK</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>ROLLBACK PREPARED</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ROLLBACK TO SAVEPOINT</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>SAVEPOINT</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>SELECT</td>
<td>YES</td>
<td><strong>Limitations:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Limited use of VOLATILE and STABLE functions in FROM or WHERE clauses</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Text search (Tsearch2) is not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FETCH FIRST or FETCH NEXT clauses not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Greenplum Database Clauses (OLAP):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[GROUP BY grouping_element [, ...]]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[WINDOW window_name AS (window_specification)]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[FILTER (WHERE condition)] applied to an aggregate function in the SELECT list</td>
</tr>
<tr>
<td>SELECT INTO</td>
<td>YES</td>
<td>See SELECT</td>
</tr>
<tr>
<td>SQL Command</td>
<td>Supported in Greenplum</td>
<td>Modifications, Limitations, Exceptions</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>SET</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>SET CONSTRAINTS</td>
<td>NO</td>
<td>In PostgreSQL, this only applies to foreign key constraints, which are currently not enforced in Greenplum Database.</td>
</tr>
<tr>
<td>SET ROLE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>SET SESSION AUTHORIZATION</td>
<td>YES</td>
<td>Deprecated as of PostgreSQL 8.1 - see SET ROLE</td>
</tr>
<tr>
<td>SET TRANSACTION</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>SHOW</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>START TRANSACTION</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>TRUNCATE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>UNLISTEN</td>
<td>NO</td>
<td></td>
</tr>
</tbody>
</table>
| UPDATE                       | YES                    | **Unsupported Clauses:** RETURNING  
**Limitations:** SET not allowed for Greenplum distribution key columns. |
| VACUUM                       | YES                    | **Limitations:** VACUUM FULL is not recommended in Greenplum Database. |
| VALUES                       | YES                    |                                        |
Chapter 17

Greenplum Database UNIX Client Documentation

Information and instructions for installing and using client and load tools on UNIX systems.
Greenplum Database Client Tools for UNIX

Utility programs that connect to Greenplum Database from UNIX systems.

Installing the Greenplum Client Tools

This section contains information for installing the client programs on your client machine and for enabling Greenplum Database to accept remote client connections:

- Running the Client Tools Installer
- Configuring the Command-Line Tools
- Enabling Greenplum Database for Remote Client Connections
- Configuring a Client System for Kerberos Authentication
- Next Steps

See the Greenplum Database Release Notes for the list of currently supported platforms for the Client Tools.

Running the Client Tools Installer

The Greenplum Database client tools installer installs the following client tools:

- PostgreSQL Interactive Terminal (`psql`)
- Greenplum MapReduce Client Program (`gpmapreduce`)

To install the Greenplum Database client tools

1. Download the appropriate `greenplum-clients-<version>-PLATFORM.bin` installer package for your platform from Pivotal Network.
2. Unzip the installer:
   ```bash
   unzip greenplum-clients-<version>-PLATFORM.bin.zip
   ```
3. Run the installer:
   ```bash
   /bin/bash greenplum-clients-<version>-PLATFORM.bin
   ```
4. The installer will prompt you to accept the license agreement and to provide an installation path. For the installation path, be sure to enter an absolute path if you choose not to accept the default location (for example, `/mydir/gp-client-tools`). The client tools are installed into `greenplum-db-<version>` by default.

About Your Installation

Your Greenplum Database client tools installation contains the following files and directories:

- `bin` — client command-line tools (`psql` and `gpmapreduce`)
- `greenplum_clients_path.sh` — environment variables
- `include` — libpq C header files
- `lib` — client tools library files

Configuring the Command-Line Tools

As a convenience, a `greenplum_clients_path.sh` file is provided in your client tools installation directory following installation. It has the following environment variable settings:

`GPHOME_CLIENTS` — The installation directory of the Greenplum Database client tools.
**PATH** — The paths to the command-line utilities.

**LD_LIBRARY_PATH** — The path to the library files.

You can source this file in your user's startup shell profile (such as `.bashrc` or `.bash_profile`).

For example, you could add a line similar to the following to your chosen profile files (making sure the right install path is used):

```
source greenplum-db-<version>/greenplum_clients_path.sh
```

After editing the chosen profile file, source it as the correct user to make the changes active. For example:

```
source ~/.bashrc
```

### Additional Connection Environment Variables

The PostgreSQL command-line tools require several connection parameters in order to be able to connect to a Greenplum Database database. In order to save some typing on the command-line, you can create the following environment variables in your preferred profile file (such as `.bashrc`).

- **PGDATABASE** — The name of the default Greenplum database to connect to.
- **PGHOST** — The Greenplum master host name or IP address.
- **PGPORT** — The port number that the Greenplum master instance (postmaster process) is running on.
- **PGUSER** — The default database role name to use for login.

### Enabling Greenplum Database for Remote Client Connections

In order for Greenplum Database to be able to accept remote client connections, you must configure your Greenplum Database master so that connections are allowed from the client hosts and database users that will be connecting to Greenplum Database.

**To enable remote client connections**

1. Make sure that the `pg_hba.conf` file of the Greenplum Database master is correctly configured to allow connections from the users to the database(s) using the authentication method you want. For details, see "Editing the pg_hba.conf File" in the *Greenplum Database Administration Guide*, and also see the *Greenplum Database Security Configuration Guide*.

   Make sure the authentication method you choose is supported by the client tool you are using.

2. If you edited `pg_hba.conf` file, the change requires a server reload (using the `gpstop -u` command) to take effect.

3. Make sure that the databases and roles you are using to connect exist in the system and that the roles have the correct privileges to the database objects.

### Configuring a Client System for Kerberos Authentication

If you use Kerberos authentication to connect to a Greenplum Database with the `psql` utility, your client system must be configured to use Kerberos authentication. If you are not using Kerberos authentication to connect to a Greenplum Database, Kerberos is not needed on your client system.

- **Requirements**
- **Setting Up a Client System with Kerberos Authentication**
- **Accessing Greenplum Database with psql**

For information about enabling Kerberos authentication with Greenplum Database, see the "Kerberos Authentication" chapter in the *Greenplum Database Administrator Guide*. 
Requirements

The following are requirements to connect to a Greenplum Database that is enabled with Kerberos authentication from a client system with Greenplum Database client software.

- Prerequisites
- Required Software on the Client Machine

Prerequisites

- Kerberos must be installed and configured on the Greenplum Database master host. See Enabling Greenplum Database for Remote Client Connections.
- The client systems require the Kerberos configuration file krb5.conf from the Greenplum Database master.
- The client systems require a Kerberos keytab file that contains the authentication credentials for the Greenplum Database user that is used to log into the database.
- The client machines must be able to connect to Greenplum Database master host.

If necessary, add the Greenplum Database master host name and IP address to the system hosts file. On Linux systems, the hosts file is located in /etc.

Required Software on the Client Machine

The Kerberos kinit utility is required on the client machine. The kinit utility is available when you install the Kerberos packages:

- krb5-libs
- krb5-workstation

Note: When you install the Kerberos packages, you can use other Kerberos utilities such as klist to display Kerberos ticket information.

Setting Up a Client System with Kerberos Authentication

To connect to Greenplum Database with Kerberos authentication requires a Kerberos ticket. On client systems, tickets are generated from Kerberos keytab files with the kinit utility and are stored in a cache file.

1. Install a copy of the Kerberos configuration file krb5.conf from the Greenplum Database master. The file is used by the Greenplum Database client software and the Kerberos utilities.

Install krb5.conf in the /etc directory.

If needed, add the parameter default_ccache_name to the [libdefaults] section of the krb5.ini file and specify the location of the Kerberos ticket cache file on the client system.

2. Obtain a Kerberos keytab file that contains the authentication credentials for the Greenplum Database user.

3. Run kinit specifying the keytab file to create a ticket on the client machine. For this example, the keytab file gpdb-kerberos.keytab is in the current directory. The ticket cache file is in the gpadmin user home directory. For example:

```
$ kinit -k -t gpdb-kerberos.keytab
   -c /home/gpadmin/cache.txt
gpadmin/kerberos-gpdb@KRB.EXAMPLE.COM
```

Accessing Greenplum Database with psql

From a remote system, you can access a Greenplum Database that has Kerberos authentication enabled.
To connect to Greenplum Database with `psql`

1. As the `gpadmin` user, open a command window.
2. Start `psql` from the command window and specify a connection to the Greenplum Database specifying the user that is configured with Kerberos authentication.

   The following example logs into the Greenplum Database on the machine `kerberos-gpdb` as the `gpadmin` user with the Kerberos credentials `gpadmin/kerberos-gpdb`:

   ```
   $ psql -U "gpadmin/kerberos-gpdb" -h kerberos-gpdb postgres
   ```

Next Steps

Refer to the client tool reference documentation for further instructions:

- **Greenplum Command Line Tools** — See *Client Tools Reference*.
- **Greenplum Database SQL Syntax** — See *SQL Syntax Summary*.

**Client Tools Reference**

This chapter references the client tools provided with this release. They all require certain connection information such as the Greenplum Database master host name, port, database name, and role name. These can be configured using environment variables. See *Configuring the Command-Line Tools*.

The following tools are provided:

- **gpmapreduce** (submits Greenplum MapReduce specifications for execution).
- **psql** (PostgreSQL interactive terminal)

See also the *SQL Syntax Summary* for a quick reference to available SQL commands.
Greenplum Database Load Tools for UNIX

Programs that load data into Greenplum Database from UNIX systems.

Installing the Greenplum Load Tools

This section contains information for installing the Greenplum data loading programs on your client machine and for enabling Greenplum Database to accept remote client connections:

- Running the Load Tools Installer
- Configuring the Command-Line Load Tools
- Enabling Greenplum Database for Remote Client Connections
- Next Steps

See the Greenplum Database Release Notes for the list of currently supported platforms for the Load Tools.

Running the Load Tools Installer

The Greenplum Database load tools installer installs the following data loading tools:

- Greenplum parallel file distribution program (gpfdist)
- Greenplum data loading utility (gpload)

To install the Greenplum Database load tools

1. Download the appropriate greenplum-loaders-<version>-PLATFORM.bin.zip installer package for your platform from Pivotal Network.
2. Unzip the installer:
   ```bash
   unzip greenplum-loaders-<version>-PLATFORM.bin.zip
   ```
3. Run the installer:
   ```bash
   /bin/bash greenplum-loaders-<version>-PLATFORM.bin
   ```
4. The installer will prompt you to accept the license agreement and to provide an installation path. For the installation path, be sure to enter an absolute path if you choose not to accept the default location (for example, /mydir/gp-loader-tools). The load tools are installed into greenplum-db-<version> by default.

About Your Installation

Your Greenplum Database load tools installation contains the following files and directories:

- **bin** — data loading command-line tools and library files (gpfdist and gpload)
- **docs** — documentation files
- **greenplum_loaders_path.sh** — environment variables

Configuring the Command-Line Load Tools

As a convenience, a greenplum_loaders_path.sh file is provided in your load tools installation directory following installation. It has the following environment variable settings:

- **GREENPLUM_LOADERS_HOME** — The installation directory of the Greenplum Database load tools.
- **PATH** — The path to the data loading command-line utilities.
- **LD_LIBRARY_PATH** — The path to additional Python library files needed for gpload.
You can source this file in your user's startup shell profile (such as `.bashrc` or `.bash_profile`). For example, you could add a line similar to the following to your chosen profile files (making sure the right install path is used):

```
source greenplum-db-<version>/greenplum_loaders_path.sh
```

After editing the chosen profile file, source it as the correct user to make the changes active. For example:

```
source ~/.bashrc
```

### Additional Connection Environment Variables

The Greenplum load tools require several connection parameters in order to be able to connect to a Greenplum Database instance. In order to save some typing on the command-line, you can create the following environment variables in your preferred profile file (such as `.bashrc`).

- **PGDATABASE** — The name of the default Greenplum database to connect to.
- **PGHOST** — The Greenplum master host name or IP address.
- **PGPORT** — The port number that the Greenplum master instance is running on.
- **PGUSER** — The default database role name to use for login.

### Enabling Greenplum Database for Remote Client Connections

In order for Greenplum Database to be able to accept remote client connections, you must configure your Greenplum Database master so that connections are allowed from the client hosts and database users that will be connecting to Greenplum Database.

**To enable remote client connections**

1. Make sure that the `pg_hba.conf` file of the Greenplum Database master is correctly configured to allow connections from the users to the database(s) using the authentication method you want. For details, see "Editing the pg_hba.conf File" in the Greenplum Database Administration Guide, and also see the Greenplum Database Security Configuration Guide. Make sure the authentication method you choose is supported by the client tool you are using.

   2. If you edited `pg_hba.conf` file, the change requires a server reload (using the `gpstop -u` command) to take effect.

3. Make sure that the databases and roles you are using to connect exist in the system and that the roles have the correct privileges to the database objects.

### Next Steps

Refer to the load tool reference documentation for further instructions:

- **Greenplum Load Tools** — See Load Tools Reference.
- **Greenplum Database SQL Syntax** — See SQL Syntax Summary.

### Load Tools Reference

This chapter describes the data loading tools provided with this release. They all require certain connection information such as the Greenplum Database master host name, port, database name, and role name. These can be configured using environment variables. For more information, see Configuring the Command-Line Load Tools.

The following tools are provided:

- Greenplum parallel file distribution program (`gpfdist`)
- Greenplum data loading utility (`gpload`)
See also the *SQL Syntax Summary* for a quick reference to available SQL commands.
Chapter 18

Greenplum Database Windows Client Documentation

Information and instructions for installing and using client and load tools on Microsoft Windows systems.
Greenplum Database Client Tools for Windows

Utility programs that connect to Greenplum Database from Microsoft Windows systems.

Installing the Greenplum Client Tools

This section contains information for installing the various client programs on your Windows machine and for enabling Greenplum Database to accept remote client connections:

• Running the Client Tools Installer
• Configuring the Client Tools
• Enabling Greenplum Database for Remote Client Connections
• Configuring a Client System for Kerberos Authentication

See the Greenplum Database Release Notes for the list of currently supported platforms for the Client Tools.

Running the Client Tools Installer

The Greenplum Database client tools installer installs psql.exe, the interactive command-line client interface to Greenplum Database.

To install the Greenplum Database client tools

1. Download the greenplum-clients-<version>-WinXP-x86_32.msi package from Pivotal Network.
2. Double-click on the greenplum-clients-<version>-WinXP-x86_32.msi package to launch the installer.
3. Click Next on the Welcome screen.
4. Click I Agree on the License Agreement screen.
5. By default, the Greenplum Database client tools will be installed into greenplum-db-<version>. Click Browse if you want to choose another location.
6. Click Next when you have chosen the install path you want.
7. Click Install to begin the installation.
8. Click Finish to exit the installer.

About Your Installation

Your Greenplum Database client tools installation contains the following files and directories:

• bin — client tools programs
• greenplum_clients_path.bat — sets environment variables
• lib — client tools library files

Configuring the Client Tools

Greenplum provides a batch program (greenplum_clients_path.bat) to set the required environment settings for Greenplum loader (located in greenplum-db-<version> by default).

To set the required environment settings

1. Open a Windows command prompt (Start > Run and type cmd).
2. At the command prompt, go to the directory where you installed Greenplum loader. For example:

   cd "Program Files"\Greenplum\greenplum-clients-<version>
3. Execute the `greenplum_loaders_path.bat` program:

```
greenplum_clients_path.bat
```

The PostgreSQL command-line tools also require several connection parameters in order to be able to
clock to a Greenplum database. In order to save some typing on the command-line, you can optionally
create the following environment variables in your Windows Control Panel.

- `PGDATABASE` — The name of the default Greenplum database to connect to.
- `PGHOST` — The Greenplum Database master host name or IP address.
- `PGPORT` — The port number that the Greenplum master instance (postmaster process) is running on.
- `PGUSER` — The default database role name to use for login.

**To add a new user environment variable on Windows XP**

1. In Windows Explorer, go to `C:\\Control Panel`.
2. Double-click the `System` icon.
3. On the `Advanced` tab, click `Environment Variables` (bottom).
4. Click `New`.
5. Define the new environment variable. For example:

6. Click `OK`.

### Enabling Greenplum Database for Remote Client Connections

In order for Greenplum Database to be able to accept remote client connections, you must configure your
Greenplum Database master so that connections are allowed from the client hosts and database users that
will be connecting to Greenplum Database.

**To enable remote client connections**

1. Make sure that the `pg_hba.conf` file of the Greenplum Database master is correctly configured to
allow connections from the users to the database(s) using the authentication method you want. For
details, see "Editing the pg_hba.conf File" in the *Greenplum Database Administration Guide*, and also
see the *Greenplum Database Security Configuration Guide*.

Make sure the authentication method you choose is supported by the client tool you are using.
2. If you edited `pg_hba.conf` file, the change requires a server reload (using the `gpstop -u` command)
to take effect.
3. Make sure that the databases and roles you are using to connect exist in the system and that the roles
have the correct privileges to the database objects.

### Configuring a Client System for Kerberos Authentication

If you use Kerberos authentication to connect to your Greenplum Database with the `psql` utility, your client
system must be configured to use Kerberos authentication. If you are not using Kerberos authentication to
connect to a Greenplum Database, Kerberos is not needed on your client system.

- **Requirements**
- **Setting Up a Client System with Kerberos Authentication**
- **Accessing Greenplum Database with psql**

For information about enabling Kerberos authentication with Greenplum Database, see the chapter
"Kerberos Authentication" in the *Greenplum Database Administrator Guide*. 
Requirements

The following are requirements to connect to a Greenplum Database that is enabled with Kerberos authentication from a client system with Greenplum Database client software.

- Prerequisites
- Required Software on the Client Machine

Prerequisites

- Kerberos must be installed and configured on the Greenplum Database master host. See "Enabling Greenplum Database for Remote Client Connections."
- The client systems require the Kerberos configuration file krb5.conf from the Greenplum Database master.
- The client systems require a Kerberos keytab file that contains the authentication credentials for the Greenplum Database user that is used to log into the database.
- The client machines must be able to connect to Greenplum Database master host.

If necessary, add the Greenplum Database master host name and IP address to the system hosts file. On Windows 7 systems, the hosts file is located in C:\Windows\System32\drivers\etc\.

Required Software on the Client Machine


  Note: When you install the Kerberos software, you can use other Kerberos utilities such as klist to display Kerberos ticket information.

Setting Up a Client System with Kerberos Authentication

To connect to Greenplum Database with Kerberos authentication requires a Kerberos ticket. On client systems, tickets are generated from Kerberos keytab files with the kinit utility and are stored in a cache file.

1. Install a copy of the Kerberos configuration file krb5.conf from the Greenplum Database master. The file is used by the Greenplum Database client software and the Kerberos utilities.

   Rename krb5.conf to krb5.ini and move it to the Windows directory. On Windows 7, the Windows directory is C:\Windows.

   If needed, add the parameter default_ccache_name to the [libdefaults] section of the krb5.ini file and specify the location of the Kerberos ticket cache file on the client system.

2. Obtain a Kerberos keytab file that contains the authentication credentials for the Greenplum Database user.

3. Run kinit specifying the keytab file to create a ticket on the client machine. For the following example, the keytab file gpdb-kerberos.keytab is in the current directory. The ticket cache file is in the gpadmin user home directory.

   > kinit -k -t gpdb-kerberos.keytab
   -c C:\Users\gpadmin\cache.txt
   gpadmin/kerberos-gpdb@KRB.EXAMPLE.COM

Accessing Greenplum Database with psql

From a remote system, you can access a Greenplum Database that has Kerberos authentication enabled.

1. As the gpadmin user, open a command window.
2. Run the client tool batch file from command window `greenplum_clients_path.bat`  
   Change the current directory to the `bin` directory of the Kerberos for Windows installation. For example:
   ```
   > cd C:\test\MIT\Kerberos\bin
   ```

3. Start `psql` from the command window and specify a connection to the Greenplum Database specifying the user that is configured with Kerberos authentication.
   The following example logs into the Greenplum Database on the machine `keberos-gpdb` as the `gpadmin` user with the Kerberos credentials `gpadmin/kerberos-gpdb`:
   ```
   > psql -U "gpadmin/kerberos-gpdb" -h kerberos-gpdb template1
   ```

For information about running `psql`, see "Running the Greenplum Client Tools."

**Running the Greenplum Client Tools**

This section contains information for connecting to Greenplum Database using the `psql.exe` command-line client tool.

**Connecting to Greenplum Database**

Users and administrators always connect to Greenplum Database through the master host. In order to establish a connection to the Greenplum Database master, you will need to know the following connection information and configure your client program accordingly. See Configuring the Client Tools for more information.

**Table 237: Connection Parameters**

<table>
<thead>
<tr>
<th>Connection Parameter</th>
<th>Description</th>
<th>Environment Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database name</td>
<td>The name of the database to which you want to connect. For a newly initialized system, use the <code>postgres</code> database to connect for the first time.</td>
<td>$PGDATABASE</td>
</tr>
<tr>
<td>Host name</td>
<td>The host name of the Greenplum Database master. The default host is the local host.</td>
<td>$PGHOST</td>
</tr>
<tr>
<td>Port</td>
<td>The port number that the Greenplum Database master instance is running on. The default is 5432.</td>
<td>$PGPORT</td>
</tr>
</tbody>
</table>
### Connection Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Environment Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>User name</td>
<td>The database user (role) name to connect as. This is not necessarily the same as your OS user name. Check with your Greenplum administrator if you are not sure what you database user name is. Note that every Greenplum Database system has one superuser account that is created automatically at initialization time. This account has the same name as the OS name of the user who initialized the Greenplum system (typically gpadmin).</td>
<td>$PGUSER</td>
</tr>
</tbody>
</table>

### Running `psql.exe`

The `psql.exe` program is invoked from a Windows command-line session. For complete command syntax and options for `psql.exe`, see `psql.exe`.

Depending on the default values used or the environment variables you have set, the following examples show how to access a database in Greenplum Database via `psql`:

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

If a user-defined database has not yet been created, you can access the system by connecting to the `postgres` database. For example:

```bash
$ psql.exe postgres
```

After connecting to a database, `psql` provides a prompt with the name of the database to which `psql` is currently connected, followed by the string `=>` (or `=#` if you are the database superuser). For example:

```
gpdatabase=>
```

At the prompt, you may type in SQL commands. A SQL command must end with a `;` (semicolon) in order to be sent to the server and executed. For example:

```
=> SELECT count(*) FROM mytable;
```

For more information on using the `psql.exe` client application, see `psql.exe`. For more information on SQL commands and syntax, see SQL Syntax Summary.

### Client Tools Reference

This is a reference of the command-line client tools provided with this release. All of these tools can be run from a Windows console session (`cmd`) or a command-line utility such as Cygwin. They all require certain connection information such as the Greenplum master host name, port, database name, and role name. These can be configured using environment variables. For more information, see Configuring the Client Tools.
The following tools are provided:

- `psql.exe` (PostgreSQL interactive terminal)

See also the *SQL Syntax Summary* for a quick reference to available SQL commands.
Greenplum Database Load Tools for Windows

Programs that load data into Greenplum Database from Microsoft Windows systems.

Installing Greenplum Loader

This section contains information for installing the Greenplum data loading programs on your Windows machine and for enabling Greenplum Database to accept remote client connections:

- Installing Python
- Running the Loader Installer
- Configuring Greenplum Loader
- Enabling Greenplum Database for Remote Client Connections

See the Greenplum Database Release Notes for the list of currently supported platforms for the Load Tools.

Installing Python

The Greenplum loader program (gpload.py) for Windows requires Python 2.5.4 (32-bit version) to also be installed on your machine. If you do not have an installation of Python, you can get one from https://www.python.org.

Note: The 64-bit version of Python is not compatible with the Greenplum loader program for Windows.

To install Python

1. Download the Python 2.5.4 installer for Windows from https://www.python.org/downloads/.
2. Locate, then double-click on the python Load Tools for Windows-2.5.4.x.msi package to launch the installer.
3. Select Install for all users and click Next.
4. By default, Python will be installed into C:\PythonXX. Click Up or New if you want to choose another location. Click Next.
5. Click Next to install the selected Python components.
6. Click Finish to complete your installation.

Running the Loader Installer

The Greenplum loader installer installs the following loader programs:

- gpload.py (loader program)
- gpfdist.exe (parallel file distribution program used by gpload.py)

To install Greenplum loader

1. Download the greenplum-clients-<version>-x86_32.msi package from Pivotal Network.
2. Double-click on the greenplum-clients-<version>-x86_32.msi package to launch the installer.
3. Click Next on the Welcome screen.
4. Click I Agree on the License Agreement screen.
5. By default, Greenplum loader will be installed into greenplum-db-<version>. Click Browse to choose another location.
6. Click Next.
7. Click **Install** to begin the installation.
8. Click **Finish** to exit the installer.

**About Your Installation**

Your Greenplum loader installation contains the following files and directories:

- **bin** — loader command-line utilities (gpload.py and gpfdist.exe)
- **bin/lib** — additional Python libraries needed by gpload.py
- **greenplum_loaders_path.bat** — sets the required environment variables

**Configuring Greenplum Loader**

Greenplum provides a batch program (greenplum_loaders_path.bat) to set the required environment settings for Greenplum loader (located in greenplum-db-<version> by default).

**To set the required environment settings**

1. Open a Windows command prompt (Start > Run and type cmd).
2. At the command prompt, go to the directory where you installed Greenplum loader. For example:
   ```
   cd "Program Files"\Greenplum\greenplum-loaders-<version>
   dir
   ```
3. Execute the `greenplum_loaders_path.bat` program:
   ```
   greenplum_loaders_path.bat
   ```
4. Verify that you can execute the `gpload.py` program:
   ```
   gpload.py -?
   ```

The loader command-line tools also require several connection parameters in order to be able to connect to a Greenplum database. In order to save some typing on the command-line, you can optionally create the following environment variables in your Windows Control Panel.

- **PGDATABASE** — The name of the default Greenplum database to connect to.
- **PGHOST** — The Greenplum Database master host name or IP address.
- **PGPORT** — The port number that the Greenplum master instance (postmaster process) is running on.
- **PGUSER** — The default database role name to use for login.

**To add a new user environment variable on Windows XP**

1. In Windows Explorer, go to C:\Control Panel.
2. Double-click the **System** icon.
3. On the **Advanced** tab, click **Environment Variables** (bottom).
4. Click **New**.
5. Define the new environment variable. For example:

6. Click **OK**.

**Enabling Greenplum Database for Remote Client Connections**

In order for Greenplum Database to be able to accept remote client connections, you must configure your Greenplum Database master so that connections are allowed from the client hosts and database users that will be connecting to Greenplum Database.
To enable remote client connections

1. Make sure that the pg_hba.conf file of the Greenplum Database master is correctly configured to allow connections from the users to the database(s) using the authentication method you want. For details, see "Editing the pg_hba.conf File" in the Greenplum Database Administration Guide, and also see the Greenplum Database Security Configuration Guide.

   Make sure the authentication method you choose is supported by the client tool you are using.

2. If you edited pg_hba.conf file, the change requires a server reload (using the gpstop -u command) to take effect.

3. Make sure that the databases and roles you are using to connect exist in the system and that the roles have the correct privileges to the database objects.

Running Greenplum Loader

This section contains information for defining a load job and running the Greenplum loader program (gpload.py).

- Before You Begin
- Creating the Load Control File
- Formatting the Input Data
- Running Greenplum Loader
- Greenplum Loader Log Files
- Updating Database Statistics After Data Loads
- Vacuuming the Database After Load Errors

Before You Begin

Before you can run Greenplum loader:

1. Make sure you have installed and configured Python and the Greenplum loader programs. See Installing Greenplum Loader.

2. Make sure that you have network access to and from all hosts in your Greenplum Database array (master and segments), and to and from the hosts where the data to be loaded resides (if not on the local machine).

3. Make sure that the ports you declare in your load control file are unblocked by your Windows firewall.

4. Make sure your Greenplum Database system is up and running and that you know all of the connection information (host name, port, role name, database name, etc.).

5. Create your database, schema, and table structures in Greenplum Database prior to loading data.

6. Prepare your data so that it is in a format acceptable by Greenplum loader. See Formatting the Input Data.

7. Write your control file. The control file specifies the source data, load rules, and target table for loading the data. See Creating the Load Control File.

Creating the Load Control File

Before you can run Greenplum loader (gpload.py), you must create a text file that contains the load specification information. This file must be in valid YAML 1.1 document format and use the Greenplum schema for defining the various steps of a load operation. See gpload.py for details on the correct control file format and schema.

Here is an example of a load control file:

```yaml
---
VERSION: 1.0.0.1
DATABASE: ops
USER: gpadmin
```
Formatting the Input Data

When you use Greenplum loader, you need to specify how your data is formatted. Data can be in either delimited text (TEXT) or comma separated values (CSV) format. External data must be formatted correctly in order to be read by Greenplum Database. This section explains the format of data files expected by Greenplum Database.

- Formatting Rows
- Formatting Columns
- Representing NULL Values
- Escaping
- Character Encoding

Formatting Rows

Greenplum Database expects rows of data to be separated by the LF character (Line feed, 0x0A), CR (Carriage return, 0x0D), or CR followed by LF (CR+LF, 0x0D 0x0A). LF is the standard newline representation on UNIX or UNIX-like operating systems. Other operating systems (such as Windows or Mac OS 9) may use CR individually, or CR+LF. All of these representations of a newline are supported by Greenplum Database as a row delimiter.

Formatting Columns

The default column or field delimiter is the horizontal TAB character (0x09) for text files and the comma character (0x2C) for CSV files. However, it is possible to declare another single character delimiter using the DELIMITER attribute in the load configuration file. The delimiter character must only appear between
any two data value fields. Do not place a delimiter at the beginning or end of a row. For example, if using the pipe character (|) as your delimiter:

data value 1 | data value 2 | data value 3

**Representing NULL Values**

NULL is the value used to represent an unknown piece of data in a column or field. Within your data files you can designate a string to represent null values. The default string is \N (backslash-N) in TEXT mode, or an empty value with no quotations in CSV mode. You can also declare a different string using the NULL attribute in the load configuration file. For example, you might prefer an empty string for cases where you do not want to distinguish nulls from empty strings. When using the Greenplum Database loading tools, any data item that matches the designated null string will be considered a null value.

**Escaping**

The data file has two reserved characters that have special meaning to Greenplum Database:

- The designated delimiter character, which is used to separate columns or fields in the data file.
- The newline character used to designate a new row in the data file.

If your data contains either of these characters, you must escape the character so Greenplum treats it as data and not as a field separator or new row. By default, the escape character is a backslash (\) for text-formatted files and a double quote (") for csv-formatted files.

**Escaping in Text Formatted Files**

By default, the escape character is a backslash (\) for text-formatted files. If you want to use a different escape character, use the ESCAPE attribute in the load configuration file. In cases where your selected escape character is present in your data, you can use it to escape itself.

For example, suppose you have a table with three columns and you want to load the following three fields:

- \ = \backslash
- vertical bar = |
- exclamation point = !

Your designated delimiter character is pipe (|), and your designated escape character is backslash (\). The formatted row in your data file would look like this:

```
backslash = \\
data value 1 | data value 2 | data value 3
```

Notice how the backslash character that is part of the data is escaped with another backslash character, and the pipe character that is part of the data is escaped with a backslash character.

The escape character can also be used to escape octal and hexadecimal sequences. When used in this way, the escaped value will get converted to the equivalent character when loaded into Greenplum Database. For example, to load the ampersand character (&), you could use the escape character to escape its equivalent hexadecimal (\0x26) or octal (\046) representation.

If there is no need to escape the data in text-formatted files, you can disable escaping using the ESCAPE clause of the COPY and CREATE EXTERNALTABLE commands or the ESCAPE attribute of the load control file for `gpload.py` as follows:

```
ESCAPE 'OFF'
```

This is useful for input data that contains a lot of backslash characters within the data itself (such as web log data).
**Escaping in CSV Formatted Files**

By default, the escape character is a double quote (") for CSV-formatted files. If you want to use a different escape character, use the `ESCAPE` clause of `COPY` and `CREATE EXTERNAL TABLE` commands or the `ESCAPE` attribute of the load control file for `gpload.py` to declare a different escape character. In cases where your selected escape character is present in your data, you can use it to escape itself.

For example, suppose you have a table with three columns and you want to load the following three fields:

- Free trip to A,B
- 5.89
- Special rate "1.79"

Your designated delimiter character is comma (,), and your designated escape character is double quote ("),. The formatted row in your data file would look like this:

```
"Free trip to A,B", "5.89", "Special rate "1.79""
```

Notice how that for the comma character that is part of the data, the entire data value is enclosed in double quotes. Also notice how the double quotes that are part of the data are also escaped with a double quote even though the field value is enclosed in double quotes.

Embedding the entire field inside a set of double quotes also guarantees preservation of leading and trailing whitespace characters:

```
"Free trip to A,B " , "5.89 " , "Special rate ""1.79" "
```

**Character Encoding**

A character encoding system consists of a code that pairs each character from a given repertoire with something else, such as a sequence of numbers or octets, in order to facilitate the transmission and storage of data. The character set support in Greenplum Database allows you to store text in a variety of character sets, including single-byte character sets such as the ISO 8859 series and multiple-byte character sets such as EUC (Extended UNIX Code), UTF-8, and Mule internal code. All supported character sets can be used transparently by clients, but a few are not supported for use within the server (that is, as a server-side encoding).

Data files must be in a character encoding recognized by Greenplum Database. See the [Greenplum Database Reference Guide](#) for the supported character sets. Data files that contain invalid or unsupported encoding sequences will encounter errors when loading into Greenplum Database.

**Note:** On data files generated on a Microsoft Windows operating system, try running the `dos2unix` system command to remove any Windows-only characters prior to loading into Greenplum Database.

**Running Greenplum Loader**

Greenplum loader is invoked by running the `gpload.py` program from a Windows command-line session. For complete command syntax and options, see `gpload.py`.

**Greenplum Loader Log Files**

By default, `gpload.py` creates a directory called `gpAdminLogs` in the same location from where you execute the program and writes its log files there. Alternatively, you can use the `-l` option when executing `gpload.py` to direct the log output to a different location. See `gpload.py` for the format of these log files.
Updating Database Statistics After Data Loads

After loading data, always run the `ANALYZE` SQL command to update the database statistics used by the query planner. `ANALYZE` collects statistics about the contents of tables in the database, and stores the results in the system table `pg_statistic`. The query planner uses these statistics to help determine the most efficient execution plans for queries. For example, to collect statistics on a newly loaded table, run the following on the Greenplum master host:

```
psql dbname -c 'ANALYZE mytable;'
```

Vacuuming the Database After Load Errors

The Greenplum loader will stop a load operation if it encounters an error. When this happens, the target table may already have received earlier rows in the load operation. Although these rows will not be visible or accessible, they still occupy disk space. This may amount to a considerable amount of wasted disk space if the failure happened well into a large load operation. You may wish to invoke the `VACUUM` command to recover the wasted space. For example, run the following command on the master host after a load error:

```
vacuumdb dbname [table_name]
```

`VACUUM` reclaims storage occupied by deleted tuples. In normal operation, tuples that are deleted or obsoleted by an update are not physically removed from their table; they remain present until a `VACUUM` is done. Therefore it’s recommended to do `VACUUM` periodically, especially on frequently-updated tables.

Running gpfdist as a Windows Service

Greenplum Database Loaders allow `gpfdist` to run as a Windows Service.

Follow the instructions below to download, register and activate `gpfdist` as a service:

1. Update your Greenplum Database Loader package to the latest version. This package is available from Pivotal Network.
2. Register `gpfdist` as a Windows service:
   a. Open a Windows command window
   b. Run the following command:

   ```
scc create gpfdist binpath= "path_to_gpfdist.exe -p 8081 -d External\load\files\path -l Log\file\path"
   ```

   You can create multiple instances of `gpfdist` by running the same command again, with a unique name and port number for each instance:
   ```
scc create gpfdistN binpath= "path_to_gpfdist.exe -p 8082 -d External\load\files\path -l Log\file\path"
   ```

3. Activate the `gpfdist` service:
   a. Open the Windows Control Panel and select Administrative Tools > Services.
   b. Highlight then right-click on the `gpfdist` service in the list of services.
   c. Select Properties from the right-click menu, the Service Properties window opens.
   d. Optional: Change the Startup Type to Automatic (after a system restart, this service will be running), then under Service status, click Start.
   e. Click OK.
Repeat the above steps for each instance of `gpfdist` that you created.

**Loader Program Reference**

This is a reference of the command-line loader programs. These programs can be run from a Windows console session (cmd) or a command-line utility such as Cygwin. They all require certain connection information such as the Greenplum master host name, port, database name, and role name. These can be configured using environment variables. For more information, see *Configuring Greenplum Loader*.

The following loader programs are provided:

- `gpload.py` (loader program)
- `gpfdist.exe` (parallel file distribution program used by `gpload.py`)

See also the *SQL Syntax Summary* for a quick reference to available SQL commands.
Chapter 19

DataDirect ODBC Drivers for Pivotal Greenplum

ODBC drivers enable third party applications to connect via a common interface to the Pivotal Greenplum Database system. This document describes how to install DataDirect Connect XE for ODBC drivers for Pivotal Greenplum on either a Linux or Windows system. Unless specified otherwise, references to DataDirect Connect XE for ODBC refer to DataDirect Connect XE for ODBC and DataDirect Connect64 XE for ODBC.

The DataDirect ODBC Drivers for Pivotal Greenplum are available for download from Pivotal Network.
Prerequisites

- Install KornShell (ksh) on your system if it is not available.
- Note the appropriate serial number and license key (use the same number for both the serial number and license key during the installation):

<table>
<thead>
<tr>
<th>Driver</th>
<th>Serial Number / License Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataDirect Connect XE for ODBC 7.1 drivers (32-bit drivers)</td>
<td>1076681728</td>
</tr>
<tr>
<td>DataDirect Connect64 XE for ODBC 7.1 drivers (64-bit drivers)</td>
<td>1076681984</td>
</tr>
</tbody>
</table>
Supported Client Platforms

DataDirect Connect64 XE for ODBC drivers for Greenplum support the following 64-bit client platforms:

- AIX 64: 7.1, 7.0, 6.1, 5.3 Fixpack 5 or higher
- HP-UX IPF: 11i v3 (B.11.3X), 11i v2 (B.11.23)
- Linux Itanium: Red Hat Enterprise Linux (RHEL) 6.x, RHEL 5.x, RHEL 4.x
- Linux x64: RHEL 6.x, RHEL 5.x, RHEL 4.x, SUSE Linux Enterprise Server (SLES) 11, SLES 10
- Solaris on SPARC: 11 and 11 Express (Solaris 5.11), 10 (Solaris 5.10), 9 (Solaris 5.9), 8 (Solaris 5.8)
- Solaris x64: 11 (Solaris 5.11), 10 (Solaris 5.10)

DataDirect Connect XE for ODBC drivers for Greenplum support the following 32-bit client platforms:

- AIX 32: 7.1, 7.0, 6.1, 5.3 Fixpack 5 or higher
- HP-UX IPF: 11i v3 (B.11.3X), 11i v2 (B.11.23)
- HP-UX PA-RISC: 11i v3 (B.11.3X), 11i v2 (B.11.23) 11i v1 (B.11.11), 11
- Linux x86: Red Hat Enterprise Linux (RHEL) 6.x, RHEL 5.x, RHEL 4.x, SUSE Linux Enterprise Server (SLES) 11, SLES 10
- Solaris on SPARC: 11 and 11 Express (Solaris 5.11), 10 (Solaris 5.10), 9 (Solaris 5.9), 8 (Solaris 5.8)
Installing on Linux Systems

To install ODBC drivers on your client:

1. Log into Pivotal Network and download the correct ODBC driver for your operating system. The following Linux and UNIX files are available:
   - PROGRESS_DATADIRECT_CONNECT64_ODBC_7.1.6.HOTFIX_LINUX_64.TAR.Z
   - PROGRESS_DATADIRECT_CONNECT64_ODBC_7.1.6.HOTFIX_LINUX_32.TAR.Z
   - PROGRESS_DATADIRECT_CONNECT64_ODBC_7.1.6.HOTFIX_AIX_64.TAR.Z
   - PROGRESS_DATADIRECT_CONNECT64_ODBC_7.1.6.HOTFIX_AIX_32.TAR.Z

2. Unpack the files. For example:
   
   ```
   $ tar -zxvf PROGRESS_DATADIRECT_CONNECT64_ODBC_7.1.6.HOTFIX_LINUX_64.TAR.Z
   ```

3. Move to the new driver directory and execute the installer:

   ```
   $ cd PROGRESS_DATADIRECT_CONNECT64_ODBC_7.1.6.HOTFIX_LINUX_64
   $ ksh unixmi.ksh
   ```

   Progress DataDirect Connect for ODBC Setup is preparing....

   English has been set as the installation language.

   Log file : /tmp/logfile.492.1

   The following operating system has been detected:

   LinuxX64

   Is this the current operating system on your machine (Y/N) ?

4. Press **Y** to confirm your operating system. The installer displays the license agreement.

5. Enter **YES** to accept the End User License Agreement. The installer prompts you for registration information:

   ```
   Enter YES to accept the above agreement : YES
   Please enter the following information for proper registration.
   In the Key field, enter either EVAL or the Key provided.
   Name 
   Company:
   Serial Number:
   - 1076681984 for 64-bit driver, or
   - 1076681728 for 32-bit driver.
   Key:
   - 1076681984 for 64-bit driver, or
   - 1076681728 for 32-bit driver.
   ```

6. Enter the required registration information at each prompt:
The installation program displays the registered driver information. For example:

You have chosen the Greenplum Wire Protocol driver.
Server Unlimited
Unlimited Connections

To change this information, enter C. Otherwise, press Enter to continue. :

7. Press Enter to continue with the installation. The installer prompts you for a temporary directory:

DataDirect Connect for ODBC Setup is preparing the installation.
Choose a temporary directory.

Enter the full path to the temporary install directory. [/tmp]:

8. Press Enter to accept the default /tmp directory or enter a custom directory to store temporary files.
The installer extracts temporary files and prompts you for an installation directory:

Checking for available space...
There is enough space.
Extracting files...

Choose a destination directory.
Enter the full path to the install directory. [/opt/Progress/DataDirect/Connect64_for_ODBC_71]:

9. Press Enter to accept the default directory or enter a custom destination directory. The installer checks
for available space and installs the software:

Checking for available space...
There is enough space.
Extracting files...

Creating license file......

DataDirect Connect for ODBC Setup successfully removed all of the
temporary files.

Thank you for using Progress DataDirect products under OEM license to
Greenplum Inc.

Would you like to install another product (Y/N) ? [Y]

10. Enter N to exit the installer.

**Configuring the Driver on Linux**

After you install the driver software, perform these steps to configure the driver.

1. Change to the installation directory for your driver. For example:

    $ cd /opt/Progress/DataDirect/Connect64_for_ODBC_71/

2. Set the LD_LIBRARY_PATH, ODBCINI and ODBCINST environment variables with the command:

    $ source odbc.sh
3. Open the odbc.ini file and create a new DSN entry. You can use the existing “Greenplum Wire Protocol” entry as a template.

```bash
$ vi $ODBCINI
```

You must edit the following entries to add values that match your system:

<table>
<thead>
<tr>
<th>Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Pivotal Greenplum database name.</td>
</tr>
<tr>
<td>HostName</td>
<td>Master host name.</td>
</tr>
<tr>
<td>PortNumber</td>
<td>Master host port number.</td>
</tr>
<tr>
<td>LogonID</td>
<td>Greenplum Database user.</td>
</tr>
<tr>
<td>Password</td>
<td>Password.</td>
</tr>
</tbody>
</table>

4. Verify the driver version:

```bash
$ cd /opt/Progress/DataDirect/Connect64_for_ODBC_71/bin
$ ./ddtestlib ddgplm27.so
Load of /usr/local/Progress/DataDirect/Connect64_for_ODBC_71/lib/ddgplm27.so successful, qehandle is 0xEE5060
File version: 07.10.0015 (B0020, U0014)
```

**Testing the Driver Connection on Linux**

To test the DSN connection:

1. Execute the example utility to test the DSN connection, entering the Greenplum Wire Protocol data source name and the credentials of a Pivotal Greenplum user. For example:

```bash
$ cd /opt/Progress/DataDirect/Connect64_for_ODBC_71/samples/example
$ ./example
./example DataDirect Technologies, Inc. ODBC Example Application.
Enter the data source name: Greenplum Wire Protocol
Enter the user name: gpadmin
Enter the password: gpadmin

Enter SQL statements (Press ENTER to QUIT)
SQL>
```

2. Enter the following select statement to confirm database connectivity:

```sql
Enter SQL statements (Press ENTER to QUIT)
SQL> select version();
version
PostgreSQL 8.3.23 (Greenplum Database 5.0.0 build commit:8c709516061c5f5476c03d6e2da99aae42722ae1) on x86_64-pc-linux-gnu, compiled by GCC gcc (GCC) 6.2.0 compiled on Sep 1 2017 22:39:53
```

3. Press the ENTER key to exit the example application.
Installing on Windows Systems

To install ODBC drivers on your client:

1. Log into Pivotal Network and download the correct ODBC driver for your operating system (32-bit or 64-bit). The following Windows files are available:
   - PROGRESS_DATADIRECT_CONNECT64_ODBC_7.1.6.HOTFIX_WIN_64.zip
   - PROGRESS_DATADIRECT_CONNECT64_ODBC_7.1.6.HOTFIX_WIN_32.zip
2. Uncompress the installer.
3. Double-click setup.exe to launch the install wizard.
4. If necessary, permit the InstallAnywhere installer to run.
5. Click Next at the Introduction screen to begin the installation.
6. Accept the End User License Agreement and click Next.
7. Select OEM or Licensed Installation as the installation type and click Next.
8. Enter your licensing information: Division name, Company Name, and serial number/license key found in Prerequisites.
9. Select Add. You should see this driver in the License dialog box: ODBC Greenplum Wire Protocol Third Party All Platform Server Unlimited Cores
10. Select Next.
11. Choose options appropriate for your installation. For example, select to replace the existing drivers and/or to create the default data sources. Click Next.
12. Accept the default installation directory or choose a custom directory. Click Next.
13. Verify the selected installation options, and click Install to begin installation. The installation process may take several minutes.
14. Select Done to complete installing the driver package.

Verifying the Version on Windows

To verify your driver version:

1. Select Start > All Programs > DataDirect > ODBC Administrator to open the Windows ODBC Administrator.
2. Click the Drivers tab, and scroll down to DataDirect <version> Greenplum Wire Protocol. Ensure that you see the expected version number.

Configuring and Testing the Driver on Windows

To configure and test a DSN connection to a Greenplum Database:

1. Open the ODBC Administrator.
2. Select the System DSN tab.
3. Select Add.
5. Enter the details for your chosen Greenplum Database instance. For example:

   Recommended: Set the Max Long Varchar size.
   - Select the Advanced tab.
   - In Max Long Varchar Size, enter 8192 then select Apply.
6. Select **Test Connect**.
7. Enter your user name and password, then select **OK**.
8. You should see the confirmation message **Connection Established!**

If your connection fails, check the following for accuracy:

- Host Name
- Port Number
- Database Name
- User Name
- Password
- Greenplum instance is active
DataDirect Driver Documentation

For more information on working with Data Direct, see documentation that is installed with the driver. By default, you can access the installed documentation by using a Web browser to open the file /opt/Progress/DataDirect/Connect64_for_ODBC_71/help/index.html.

Documentation is also available online at https://www.progress.com/documentation/datadirect-connectors. Titles include:

- User’s Guide
- Reference
- Troubleshooting Guide
- Installation Help
- Windows Readme
- UNIX/Linux Readme
Chapter 20

DataDirect JDBC Driver for Pivotal Greenplum

DataDirect JDBC drivers are compliant with the Type 4 architecture, but provide advanced features that define them as Type 5 drivers. Additionally, the drivers consistently support the latest database features and are fully compliant with Java™ SE 8 and JDBC 4.0 functionality.

The DataDirect JDBC Driver for Pivotal Greenplum is available for download from Pivotal Network.
Prerequisites

- The DataDirect JDBC Driver requires Java SE 5 or higher. See System and Product Requirements in the DataDirect documentation for information and requirements associated with specific features of the JDBC driver.
- The license key is embedded in the greenplum.jar file itself. You do not need to apply a specific license key to the driver to activate it.
Downloading the DataDirect JDBC Driver

To install the JDBC driver on your client:

1. Log into Pivotal Network and download the DataDirect JDBC driver file: greenplum.jar
2. Add the full path to the greenplum.jar to your Java CLASSPATH environment variable, or add it to your classpath with the -classpath option when executing a Java application.
Obtaining Version Details for the Driver

To view the JDBC driver version information:

1. Change to the directory that contains the downloaded `greenplum.jar` driver file. For example:

   ```bash
   $ cd /opt/Progress/DataDirect/Connect_for_JDBC_51/lib
   ```

2. Execute the data source class to display the version information.

   For Linux/Unix systems:

   ```bash
   $ java -classpath greenplum.jar com.pivotal.jdbc.GreenplumDriver
   [Pivotal][Greenplum JDBC Driver]Driver Version: 5.1.4.000143
   (F000337.U000158)
   ```

   For Windows systems:

   ```bash
   java -classpath .;\greenplum.jar com.pivotal.jdbc.GreenplumDriver
   [Pivotal][Greenplum JDBC Driver]Driver Version: 5.1.4.000143
   (F000337.U000158)
   ```
## Usage Information

The JDBC driver is provided in the `greenplum.jar` file. Use the following data source class and connection URL information with the driver.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver File Name</td>
<td>greenplum.jar</td>
</tr>
<tr>
<td>Data Source Class</td>
<td>com.pivotal.jdbc.GreenplumDriver</td>
</tr>
<tr>
<td>Connection URL</td>
<td>jdbc:pivotal:greenplum://host:port;DatabaseName=&lt;name&gt;</td>
</tr>
<tr>
<td>Driver Defaults</td>
<td>ResultSetMetadataOptions=1</td>
</tr>
<tr>
<td></td>
<td>FetchTWFSasTime=true</td>
</tr>
<tr>
<td></td>
<td>SupportsCatalogs=true</td>
</tr>
<tr>
<td></td>
<td>MaxLongVarcharSize=8190</td>
</tr>
<tr>
<td></td>
<td>MaxNumericPrecision=28</td>
</tr>
<tr>
<td></td>
<td>MaxNumericScale=6</td>
</tr>
</tbody>
</table>
DataDirect Driver Documentation

For more information on working with the Data Direct JDBC driver, see documentation available online at https://www.progress.com/documentation/datadirect-connectors. Titles include:

- User's Guide
- Reference
- Installation Help
- Readme
- Quick Start