VMware Tanzu Greenplum v5.26 Documentation

VMware Tanzu Greenplum 5

You can find the most up-to-date technical documentation on the VMware website at:

https://docs.vmware.com/

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Pivotal Greenplum 5.26.0 Release Notes

Updated: April, 2020

Welcome to Pivotal Greenplum 5.26.0

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.

This document contains pertinent release information about Pivotal Greenplum Database 5.26.0. 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 the Pivotal Support Lifecycle Policy.

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

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

Important: The Greenplum `gpbackup` and `gprestore` utilities are now distributed separately from Pivotal Greenplum Database, and are updated independently of the core Greenplum Database server. These utilities will not be updated in future Greenplum Database 5.x releases. You can upgrade to the latest `gpbackup` and `gprestore` versions by downloading and installing the latest Pivotal Greenplum Backup and Restore release from Pivotal Network.
Important: Pivotal Support does **not** provide support for open source versions of Greenplum

Database. Only Pivotal Greenplum Database is supported by Pivotal Support.

Pivotal Greenplum 5.26.0 is a minor release that changes features and resolves issues.

New Features

Greenplum Database 5.26.0 includes these new features:

- For the `CREATE EXTERNAL TABLE` command, the `LOG ERRORS` clause supports the new `PERSISTENTLY` keyword. The `LOG ERRORS` clause logs information about external table data rows with formatting errors. The error log data is stored internally. When you specify `LOG ERRORS PERSISTENTLY`, the log data persists after the external table is dropped.

  If you use the `PERSISTENTLY` keyword, you must install the functions that manage the persistent error log information.

  For information about the error log information and functions for viewing and managing error log information, see `CREATE EXTERNAL TABLE`
- The PL/Container version has been updated to 1.6.0. This version supports Docker images with Python 3 installed. These new PL/Container features enable support for Python 3:
  - A Docker image that is installed with Python 3 - `plcontainer-python3-images-1.6.0.tar.gz`

    The Docker image can be downloaded from Pivotal Network.
  - The new value `python3` for the `--language` option of `plcontainer runtime-add` command.

    You specify this value when you add a Docker image that has Python 3 installed on to the Greenplum Database hosts with the `plcontainer runtime-add` command.

  Note: PL/Container 1.5.x and earlier do not support Python 3.

  For information about PL/Container, see PL/Container Language.
- Greenplum Database 5.26 includes MADlib version 1.17, which introduces new Deep Learning features, k-Means clustering, and other improvements and bug fixes. See the MADlib 1.17 Release Notes for a complete list of changes.
- PXF version 5.11.2 is included, which introduces new and changed features and bug fixes. See PXF Version 5.11.2 below.

PXF Version 5.11.2

PXF includes the following new and changed features:

- PXF provides a `restart` command to stop, and then restart, all PXF server instances in the cluster. See Restarting PXF.
- The `pxf [cluster] sync` command now recognizes a `[-d | --delete]` option. When specified, PXF deletes files on the remote host(s) that are not present in the PXF user configuration on the Greenplum Database master host. Refer to pxf and pxf cluster.
- PXF supports filter predicate pushdown for Parquet data that you access with the Hadoop and Object Store Connectors. Parquet Data Type Mapping describes filter pushdown support for Parquet data types in PXF.
- PXF no longer validates the JDBC `BATCH_SIZE` write option during a read operation.
- PXF includes improvements to error handling and error surfacing.
- PXF bundles newer `guava`, `jackson-databind`, and Google Cloud Storage `hadoop2` libraries.
- The PXF `pxf-log4j.properties` template file updates a log filter and changes the level from `INFO` to `WARN`.

- PXF removes references to the unused `pxf-public.classpath` file. This in turn removes spurious `WARNING: Failed to read classpath file ...` log messages.
- PXF bundles Tomcat 7.0.100 and removes unused and default Tomcat applications and files, hardening its default Tomcat security.
- PXF no longer requires a `$JAVA_HOME` setting in `gpadmin`'s `.bashrc` file on the master, standby master, and segment hosts. You can now specify `JAVA_HOME` before or during PXF initialization. Refer to the Initialization Overview in the PXF initialization documentation.

Resolved Issues

The listed issues are resolved in Pivotal Greenplum Database 5.26.0.

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.

307 - PXF
: PXF did not correctly handle an external table that was created with the `ESCAPE 'OFF'` or `DELIMITER 'OFF'` formatting options. This issue is resolved. PXF now correctly neither escapes nor adds delimiters when reading external data with an external table created with these options.

30382 - VACUUM,TRUNCATE
: In some cases, performing a `VACUUM FULL` operation on the `pg_class` catalog table and concurrently performing a `TRUNCATE` operation on a user created heap table returned the error updated tuple is already `HEAP_MOVED_OFF` and caused the database to become unavailable. The `TRUNCATE` command did not properly manage the heap table entry in `pg_class` during the `TRUNCATE` operation. This issue is resolved.

30391, 168828451, 8677 - Planner
: Some queries returned incorrect results when the queries contain subqueries that perform a join and also contain one or more equality predicates and optionally an `IS NULL` predicate. Incorrect results were returned when either a merge join or a nested loop join did not correctly process the predicates. This issue is resolved.

30441 - analyzedb
: The `analyzedb` utility could fail with an error similar to `ERROR: relation "pg_aoseg.pg_aocsseg_xxxxxx" does not exist` if a table was dropped during the `analyzedb` operation. This problem was resolved by ensuring that `analyzedb` skips any dropped tables when determining the list of tables to analyze.

30450 - PXF
: PXF initialization and reset failed when the default system Java version differed from that specified in PXF's `$JAVA_HOME`. This issue is resolved; PXF has added flexibility to the specification of the `$JAVA_HOME` setting.

30452 - Dispatch
: If the server configuration parameter `check_function_bodies` was set in a session on the master, the parameter setting did not persist when a related segment instance session was reset. This caused some functions to fail. Now the parameter setting persists when a segment instance session is reset.

30464 - Query Optimizer
: GPORCA incorrectly determined that the plan for a query with a filter on a window function over the distribution key column was direct dispatchable. Now direct dispatch requires the filter to be on a table scan.

30483 - Query Optimizer
: A query that specified multiple constants in an `IN` clause generated a large number of spill files and returned the error `workfile per query size limit exceeded` when GPORCA incorrectly normalized a histogram that was not well-defined. This issue is resolved.

30489 - Server
: In some cases, concurrently running a user-defined function that truncated, inserted, and analyzed data in different target partitions of a partitioned table in a single transaction resulted

in a local deadlock. This issue is resolved.

30493 - analyzedb

When invoked with the `--config-file` option, `analyzedb` did not enumerate the leaf partitions of a partitioned table and processed the root partition as a non-partitioned table. For heap tables this produced an error. For append-optimized tables, no error was raised, but DML changes to leaf partitions were not tracked properly. This issue is resolved; using the `--config-file` option correctly analyzes partitioned tables.

30518 - Query Optimizer

A query that specified an aggregate function such as `min()` or `count()` that was invoked on a `citext`-type column failed with the error `cache lookup failed for function 0` because GPORCA incorrectly generated a multi-stage aggregate for the query. This issue is resolved.

30844 - gpreload

`gpreload` returned the error "more than one row returned" when attempting to reload a table and a view with the same name exists in a different schema in the database. This issue is resolved.

30525 - Logging

In some cases, Greenplum Database encountered a segmentation fault and rotated the log file early when the logging level was set to `WARNING` or less severe and Greenplum attempted to write to the alert log file after it failed to open the file. This issue is resolved.

169030090 - Server

Superusers were limited to 3 connections by default, causing "too many clients" errors when users run maintenance scripts. The maximum number of superuser connections is set with the `superuser_default_connections` server configuration parameter. This issue is resolved. The default value for this parameter has changed from 3 to 10.

170861600 - Server

Using `ALTER TABLE` *`tablename`* `SPLIT PARTITION` could cause rows to be assigned to the wrong partition, or could cause a crash, if one or more columns before the partition key were dropped. This issue is resolved.

Beta Features

Because Pivotal Greenplum Database is based on the open source Greenplum Database project code, it includes several Beta 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: Beta features are not supported for production deployments.
Greenplum Database 5.26.0 includes these Beta features:

- GPORCA cost model for bitmap indexes. The Beta cost model is designed to choose faster, bitmap nested loop joins instead of hash joins. The new costing model is implemented as a Beta feature, and it is used as a default only if you enable it by setting the configuration parameter:

  ```
  set optimizer_cost_model = experimental
  ```

  The `optimizer_cost_model` parameter is required only during the Beta test period for this cost model. After further testing and validation, the new cost model will be enabled by default.

- GPORCA algorithm for calculating the scale factor for join queries. The Beta algorithm increases performance by reducing the impact of multiple join predicates against the same two tables. This new algorithm is implemented as a Beta feature, and it is used as a default only if you set the configuration parameter:

  ```
  set optimizer_damping_factor_join = 0
  ```

  The `optimizer_damping_factor_join` parameter is provided only during the Beta test

period for this algorithm. After further testing and validation, the new algorithm will be used by default, and `optimizer_damping_factor_join` will be removed.

- Storage plugin API for `gpbackup` and `gprestore`. Partners, customers, and OSS developers can develop plugins to use in conjunction with `gpbackup` and `gprestore`.

  For information about the storage plugin API, see Backup/Restore Storage Plugin API.

- Recursive `WITH` Queries (Common Table Expressions). See WITH Queries (Common Table Expressions).

- Resource groups remain a Beta feature only on the SuSE 11 platform, due to limited cgroups functionality in the kernel.

  SuSE 12 resolves the Linux cgroup issues that caused the performance degradation when Greenplum Database resource groups are enabled.

Deprecated Features

Deprecated features will be removed in a future major release of Greenplum Database. Pivotal Greenplum 5.x deprecates:

- The following PXF configuration properties (deprecated since 5.24):
  - The `PXF_USER_IMPERSONATION`, `PXF_PRINCIPAL`, and `PXF_KEYTAB` settings in the `pxf-env.sh` file. You can use the `pxf-site.xml` file to configure Kerberos and impersonation settings for your new Hadoop server configurations.
  - The `pxf.impersonation.jdbc` property setting in the `jdbc-site.xml` file. You can use the `pxf.service.user.impersonation` property to configure user impersonation for a new JDBC server configuration.
- The `--skip_root_stats` option to `analyzedb` (deprecated since 5.18).

  If the option is specified, a warning is issued stating that the option will be ignored.

- The `gptransfer` utility (deprecated since 5.17).

  The utility copies objects between Greenplum Database systems. The `gpcopy` utility provides `gptransfer` functionality.

- The `gphdfs` external table protocol (deprecated since 5.17).

  Consider using the Greenplum Platform Extension Framework (PXF) `pxf` external table protocol to access data stored in an external Hadoop file system. Refer to Accessing External Data with PXF for more information.

- The Greenplum Platform Extension Framework (PXF) HDFS profile names for the Text, Avro, JSON, Parquet, and SequenceFile data formats (deprecated since 5.16).

  Refer to Connectors, Data Formats, and Profiles in the PXF Hadoop documentation for more information.

- The server configuration parameter `gp_max_csv_line_length` (deprecated since 5.11).

  For data in a CSV formatted file, the parameter controls the maximum allowed line length that can be imported into the system).

- The server configuration parameter `gp_unix_socket_directory` (deprecated since 5.9). Note: Do not change the value of this parameter. The default location is required for Greenplum Database utilities.

- Support for Data Domain Boost 3.0.0.3 (deprecated since 5.2).

  The DELL EMC end of Primary Support date is December 31, 2017.

- These unused catalog tables (deprecated since 5.1):
  - `gp_configuration`

    - `gp_db_interfaces`
    - `gp_interfaces`
- The `gpcrondump` and `gpdbrestore` utilities (deprecated since 5.0).
- The `gpcheck` utility (deprecated since 5.0).

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.
- Some features are works-in-progress and are considered to be Beta features. Pivotal does not support using Beta features in a production environment. See Beta Features.
- 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 1. Key Known Issues in Pivotal Greenplum 5.x

Issue	Category	Description
29703	Loading Data from External Tables	Due to limitations in the Greenplum Database external table framework, Greenplum Database cannot log the following types of errors that it encounters while loading data: • data type parsing errors • unexpected value type errors • data type conversion errors • errors returned by native and user-defined functions `LOG ERRORS` returns error information for data exceptions only. When it encounters a parsing error, Greenplum terminates the load job, but it cannot log and propagate the error back to the user via `gp_read_error_log()`. **Workaround**: Clean the input data before loading it into Greenplum Database.
30537	Postgres Planner	The Postgres Planner generates a very large query plan that causes out of memory issues for the following type of CTE (common table expression) query: the `WITH` clause of the CTE contains a partitioned table with a large number partitions, and the `WITH` reference is used in a subquery that joins another partitioned table. **Workaround:** If possible, use the GPORCA query optimizer. With the server configuration parameter `optimizer=on`, Greenplum Database attempts to use GPORCA for query planning and optimization when possible and falls back to the Postgres Planner when GPORCA cannot be used. Also, the specified type of query might require a long time to complete.
171883625	PXF	`pxf [cluster] init` may fail to recognize a new `JAVA_HOME` setting when the value is provided via the shell environment. **Workaround:** Edit `$PXF_CONF/conf/pxf-env.sh` and manually set `JAVA_HOME` to the new value, run `pxf cluster sync` to synchronize this configuration change across the Greenplum cluster, and then re-run `pxf [cluster] init`.

Table 1. Key Known Issues in Pivotal Greenplum 5.x

Issue	Category	Description
30420	Postgres Planner	Greenplun Database 5 generates a PANIC For some queries that use the aggregate function `percentile_cont()`. **Workaround:** Setting the server configuration parameter `gp_idf_deduplicate` to `force` eliminates the PANIC. Note: The issue does not occur in Greenplum Database 6. Also, the server configuration parameter `gp_idf_deduplicate` has been removed in Greenplum Database 6.
N/A	Greenplum Stream Server	The Pivotal Greenplum Stream Server (GPSS does not support loading data from multiple Kafka topics to the same Greenplum Database table. All jobs will hang if GPSS encounters this situation.
N/A	PXF	PXF is available only for supported Red Hat and CentOS platforms. PXF is not available for supported SuSE platforms.
9460	CREATE UNIQUE INDEX	When you create a unique index on a partitioned table, Greenplum Database does not check to ensure that the index contains the table partition keys, which are required to enforce uniqueness. **Workaround:** To avoid duplicate rows, manually specify partition keys when executing `CREATE UNIQUE INDEX` on a partitioned table.

3290	JSON	The `to_json()` function is not implemented as a callable function. Attempting to call the function results in an error. For example:
		`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.

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`.
29139	DML	In some cases for an append-optimized partitioned table, Greenplum Database acquires a `ROW EXCLUSIVE` lock on all leaf partitions of the table when inserting data directly into one of the leaf partitions of the table. The locks are acquired the first time Greenplum Database performs validation on the leaf partitions. When inserting data into one leaf partition, the locks are not acquired on the other leaf partitions as long as the validation information remains in memory. The issue does not occur for heap-storage partitioned tables.

Table 1. Key Known Issues in Pivotal Greenplum 5.x

Issue	Category	Description
29246	gpconfig	When querying the `gp_enable_gpperfmon` server configuration parameter with `gpconfig -s gp_enable_gpperfmon`, `gpconfig` always reports `off`, even when the parameter has been set to `on` correctly, and the `gpmmon` and `gpsmon` agent processes are running. **Workaround:** To see if `gpperfmon` is enabled, check the value of the `gp_enable_gpperfmon` configuration parameter in the `postgresql.conf` file or use the `ps` command to look for the `gpmmon` (master) or `gpsmon` (segment) processes.
29351	gptransfer	The `gptransfer` utility can copy a data row with a maximum length of 256 MB.
29395	DDL	The `gpdbrestore` or `gprestore` utility fails when the utility attempts to restore a table from a backup and the table is incorrectly defined with duplicate columns as distribution keys. The issue is caused when the `gpcrondump` or `gpbackup` utility backed up a table that is incorrectly defined. The `CREATE TABLE AS` command could create a table that is incorrectly defined with a distribution policy that contains duplicate columns as distribution keys. The `CREATE TABLE ISSUE` has been resolved. Now the `CREATE TABLE AS` command does not create the specified type of table. The command returns an error. However, restore operations will continue to fail if you try to restore the incorrectly defined tables from a backup.
29485	Catalog and Metadata	When a session creates temporary objects in a database, Greenplum Database might not the drop temporary objects when the session ends if the session terminates abnormally or is terminated from an administrator command.
29496	gpconfig	For a small number of server configuration parameters such as `log_min_messages`, the command `gpconfig -s <config_param>` does not display the correct value of the parameter for the segment hosts when the value of the parameter on master is different than the value on the segments. For parameters with the set classification `master`, the utility displays the value set on the master for both master and segments (for information about set classifications, see Setting Parameters). For those parameters, the value on the master is passed as part of queries to segment instances. The SQL query that `gpconfig` runs to display the master and segment parameter values returns the master host value that is passed to the segment as part of the query. For a few parameters such as `log_min_messages`, segment instances use the segment host value specified in the `postgresql.conf` file at start up. The segment value can overridden for the scope of a query. **Workaround:** To display the parameter value specified in the `postgresql.conf` file on the master host and segment hosts, you can specify the `gpconfig` option `--file`.

Table 1. Key Known Issues in Pivotal Greenplum 5.x

Issue	Category	Description
29523	gptoolkit	An upgrade between minor releases does not update the `template0` database, and in some cases, using these views in the `gp_toolkit` schema might cause issues if you create a database using `template0` as the template database after you upgrade to Greenplum Database 5.11.0 or later. • The `gp_toolkit.gp_bloat_expected_pages` view might incorrectly report that a root partition table is bloated even though root partition tables do not contain data if you upgrade from Greenplum Database 5.10.x or earlier. • The `gp_toolkit.gp_bloat_diag` view might return an integer out of range error in some cases if you upgrade from Greenplum Database 5.3.0 or earlier. For example, the issues might occur if you upgrade a Greenplum Database system from 5.3.0 or an earlier 5.x release and then run a `gprestore` operation with the `--redirect-db` option to create a new database. The utility creates a new database using `template0` as the template database. **Workaround:** You can update the views in the `gp_toolkit` schema in the new database. For information about checking and updating `gp_toolkit`, see Update for gp_toolkit.gp_bloat_expected_pages Issue and Update for gp_toolkit.gp_bloat_diag Issue.
29674	VACUUM	Performing parallel `VACUUM` operations on a catalog table such as `pg_class`, `gp_relation_node`, or `pg_type` and another table causes a deadlock and blocks connections to the database. **Workaround:** Avoid performing parallel `VACUUM` operations on catalog tables and user tables.
29699	ANALYZE	In Greenplum Database 5.15.1 and earlier 5.x releases, an `ANALYZE` command might return a error that states `target lists can have at most 1664 entries` when performing an `ANALYZE` operation on a table with a large number of columns (more than 800 columns). The error occurs because the in-memory sample table created by `ANALYZE` requires an additional column to indicate whether a column is `NULL` or is a truncated column for each variable length column being analyzed (such as `varchar`, `text`, and `bpchar`, numeric, arrays, and geometric datatype columns). The error is returned when `ANALYZE` attempts to create a sample table and the number of columns (table columns and indicator columns) exceeds the maximum number of columns allowed. In Greenplum Database 5.16.0 and later 5.x releases, the `ANALYZE` sample table does not require an additional column for variable length text columns such as `varchar`, `text`, and `bpchar` columns. **WORKAROUND:** To collect statistics on the table, perform `ANALYZE` operations on 2 or more sets of table columns.
29766	VACUUM	A long-running catalog query can block `VACUUM` operations on the system catalog until the query completes or is canceled. This type of blocking cannot be observed using `pg_locks`, and the `VACUUM` operation itself cannot be canceled until the long-running query completes or is canceled.

Table 1. Key Known Issues in Pivotal Greenplum 5.x

Issue	Category	Description
29917	Segment Mirroring	A "read beyond eof" error has been observed with certain persistent tables during full recovery. The root cause of this problem has not yet been determined. Greenplum Database version 5.21 contains additional debug logging to help in determining the cause of this problem. The additional logging is enabled by default, and adds approximately 646 bytes to each persistent table entry file. If you want to disable the additional debug logging, set the `debug_filerep_config_print` configuration parameter "false."
30180	Locking, Signals, Processes	The `pg_cancel_backend()` and `pg_terminate_backend()` functions might leave some orphan processes when they are used to cancel a running `VACUUM` command. **Workaround**: You can stop the orphan processes by restarting the Greenplum Database system.
30207	Catalog and Metadata	Defining a unique index on an empty table that is defined with a `DISTRIBUTED BY` clause might change the table's distribution policy. Greenplum Database adds any columns specified in the index that are not in the table's distribution policy to the distribution policy. In Greenplum Database, the columns specified in a unique index must be a left-subset of the table's distribution policy. If the table is not empty, an error is returned.

Table 1. Key Known Issues in Pivotal Greenplum 5.x

Issue	Category	Description

148119917	Resource Groups	Testing of the 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:
		
		`[81375.325947] BUG: unable to handle kernel NULL pointer dereference at 0000000000000010[81375.325986]`
		`IP: [<ffffffff812f94b1>] rb_next+0x1/0x50 [81375.326014]`
		`PGD 0 [81375.326025] Oops: 0000 [#1] SMP [81375.326041]`
		`Modules linked in: veth ipt_MASQUERADE nf_nat_masquerade_ipv4`
		`iptable_nat nf_conntrack_ipv4 nf_defrag_ipv4 nf_nat_ipv4`
		`xt_addrtype iptable_filter xt_conntrack nf_nat nf_conntrack`
		`bridge stp llc 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_ssif pcspkr lpc_ich sg sb_edac mfd_core`
		`edac_core mei_me ipmi_si mei wmi ipmi_msghandler shpchp`
		`acpi_power_meter acpi_pad ip_tables xfs libcrc32c sd_mod`
		`crc_t10dif crct10dif_generic mgag200 syscopyarea sysfillrect`
		`crct10dif_pclmul sysimgblt crct10dif_common crc32c_intel`
		`drm_kms_helper ixgbe ttm mdio ahci igb libahci drm ptp`
		`pps_core libata 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/17 Not tainted`
		`3.10.0-327.el7.x86_64 #1 [81375.326422] Hardware name: Cisco`
		`Systems Inc UCSC-C240-M4L/UCSC-C240-M4L, BIOS C240M4.2.0.8b.0.080620151546 08/06/2015 [81375.326459]`
		
		Workaround: Upgrade to the latest-available kernel for your Red Hat or CentOS release to avoid the above system panic.

Table 1. Key Known Issues in Pivotal Greenplum 5.x

Issue	Category	Description
149789783	Resource Groups	Significant Pivotal Greenplum performance degradation has been observed when enabling resource group-based workload management on Red Hat 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. When resource groups are enabled on systems with an affected kernel, there can be a delay of 1 second or longer when starting a transaction or a query. The delay is caused by a Linux cgroup 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 and 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:** This bug does not affect CentOS 7.x and Red Hat 7.x systems. If you use Red Hat 6 and the performance with resource groups is acceptable for your use case, upgrade your kernel to version 2.6.32-696 or higher to benefit from other fixes to the cgroups implementation. SuSE 11 does not have a kernel version that resolves this issue; resource groups are still considered to be a Beta feature on this platform. Resource groups are not supported on SuSE 11 for production use.

150906510	Backup and Restore	Greenplum Database 4.3.15.0 and later backups contain the following line in the backup files:
		
		`SET gp_strict_xml_parse = false;`
		
		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:
		
		`[WARNING]:-gpdbrestore finished but ERRORS were found, please check the restore report file for details`
		
		Also, the report file may contain the error:
		
		`ERROR: unrecognized configuration parameter "gp_strict_xml_parse"`
		
		These warnings and errors do not affect the restoration procedure, and can be ignored.

Table 1. Key Known Issues in Pivotal Greenplum 5.x

Issue	Category	Description
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`

158011506	Catalog and Metadata	In some cases, the timezone used by Greenplum Database might be different than the host system timezone, or the Greenplum Database timezone set by a user. In some rare cases, times used and displayed by Greenplum Database might be slightly different than the host system time.
		The timezone used by Greenplum Database is selected from a set of internally stored PostgreSQL timezones. Greenplum Database selects the timezone by matching a PostgreSQL timezone with the user specified time zone, or the host system time zone. For example, when selecting a default timezone, Greenplum Database uses an algorithm to select a PostgreSQL timezones based on the host system timezone. If the system timezone includes leap second information, Greenplum Database cannot match the system timezone with a PostgreSQL timezone. Greenplum Database calculates a best match with a PostgreSQL timezone based on information from the host system.
		Workaround: Set the Greenplum Database and host system timezones to a timezone that is supported by both Greenplum Database and the host system. For example, you can show and set the Greenplum Database timezone with the `gpconfig` utility. These commands show the Greenplum Database timezone and set the timezone to `US/Pacific`.
		`# gpconfig -s TimeZone`
		`# gpconfig -c TimeZone -v 'US/Pacific'`
		You must restart Greenplum Database after changing the timezone. The command `gpstop -ra` restarts Greenplum Database.
		The Greenplum Database catalog view `pg_timezone_names` provides Greenplum Database timezone information.

162317340	Client Tools	On Pivotal Network in the file listings for Greenplum Database releases between 5.7.1 and 5.14.0, the Greenplum Database AIX Client Tools download file is incorrectly labeled as *Loaders for AIX 7*. The file you download is the correct AIX 7 Client Tools file.
163807792	gpbackup/ gprestore	When the `%` sign was specified as the delimeter in an external table text format, `gpbackup` escaped the `%` sign incorrectly in the `CREATE EXTERNAL TABLE` command. This has been resolved. The `%` sign is correctly escaped.
164671144	gpssh-exkeys	The `gpssh-exkeys` utility uses the Paramiko SSH library for Python, which has a dependency on the Python Cryptography Toolkit (PyCrypto) library. The following security vulnerabilities have been identified in some versions of PyCrypto. • https://nvd.nist.gov/vuln/detail/CVE-2018-6594 - lib/Crypto/PublicKey/ElGamal.py in PyCrypto through 2.6.1 generates weak ElGamal key parameters, which allows attackers to obtain sensitive information by reading ciphertext data (i.e., it does not have semantic security in face of a ciphertext-only attack). The Decisional Diffie-Hellman (DDH) assumption does not

Table 1. Key Known Issues in Pivotal Greenplum 5.x

Issue	Category	Description
		hold for PyCrypto's ElGamal implementation. Paramiko does not import this algorithm, so Greenplum Database is unaffected by this vulnerability. • https://nvd.nist.gov/vuln/detail/CVE-2013-7459 - Heap-based buffer overflow in the ALGnew function in block_templace.c in Python Cryptography Toolkit (aka pycrypto) allows remote attackers to execute arbitrary code as demonstrated by a crafted iv parameter to cryptmsg.py. This bug was introduced in PyCrypto 2.6. Greenplum Database has PyCrypto 2.0.1, and is unaffected by the vulnerability. • https://nvd.nist.gov/vuln/detail/CVE-2013-1445 - The Crypto.Random.atfork function in PyCrypto before 2.6.1 does not properly reseed the pseudo-random number generator (PRNG) before allowing a child process to access it, which makes it easier for context-dependent attackers to obtain sensitive information by leveraging a race condition in which a child process is created and accesses the PRNG within the same rate-limit period as another process. Paramiko version 1.7.6-9, used in Greenplum Database 4.x and 5.x, introduced a workaround to this bug. Paramiko version 1.18.4 in Greenplum Database 6 solves it in a more permanent way. • https://nvd.nist.gov/vuln/detail/CVE-2012-2417 - PyCrypto before 2.6 does not produce appropriate prime numbers when using an ElGamal scheme to generate a key, which reduces the signature space or public key space and makes it easier for attackers to conduct brute force attacks to obtain the private key. Paramiko does not import this algorithm, so Greenplum Database is unaffected by the vulnerability. Through testing and investigation, Pivotal has determined that these vulnerabilities do not affect Greenplum Database, and no actions are required for existing Greenplum Database 4.3 or 5.x releases. However, there may be additional unidentified vulnerabilities in the PyCrypto library, and users who install a later version of PyCrypto could be exposed to other vulnerabilities. The PyCrypto library will be removed from Greenplum Database 6.0. **Workaround**: Administrators can set up passwordless SSH between hosts in the Greenplum Database cluster without using the `gpssh-exkeys` utility. This must be done before initializing the Greenplum Database system. 1. Create an SSH key for the `gpadmin` user on each host. For example, log in to each host as `gpadmin` and use the `ssh-keygen` command to generate an SSH key. Do not enter a passphrase. 2. Add the public key for each host in the cluster to every other host in the cluster. For example, use the `ssh-copy-id` command from each host to copy the public key to every other host in the cluster. When adding new hosts to the Greenplum Database system, you must create a new SSH key for each new host and exchange keys between the existing hosts and new hosts.

Table 1. Key Known Issues in Pivotal Greenplum 5.x

Issue	Category	Description
165434975	search_path	An identified PostgreSQL security vulnerability (https://nvd.nist.gov/vuln/detail/CVE-2018-1058) also exists in Greenplum Database. The problem centers around the default `public` schema and how Greenplum Database uses the `search_path` setting. The ability to create objects with the same names in different schemas, combined with how Greenplum Database searches for objects within schemas, presents an opportunity for a user to modify the behavior of a query for other users. For example, a malicious user could insert a trojan-horse function that, when executed by a superuser, grants escalated privileges to the malicious user. There are methods to protect from this vulnerability. See A Guide to CVE-2018-1058: Protect Your Search Path on the PostgreSQL wiki for a full explanation of the vulnerability and the steps you can take to protect your data.
168142530	Backup and Restore	Backups created on Greenplum Database versions before 4.3.33.0 or 5.1.19.0 may fail to restore to Greenplum Database versions 4.3.33.0 or 5.1.19.0 or later. In Greenplum Database 4.3.33.0 and 5.1.19.0, a check was introduced to ensure that the distribution key for a table is equal to the primary key or is a left-subset of the primary key. If you add a primary key to a table that contains no data, Greenplum Database automatically updates the distribution key to match the primary key. The index key for any unique index on a table must also match or be a left-subset of the distribution key. Earlier Greenplum Database versions did not enforce these policies. Restoring a table from an older backup that has a different distribution key causes errors because the backup data file on each segment contains data that was distributed using the original distribution key. Restoring a unique index with a key that does not match the distribution key will fail with an error when attempting to create the index. This issue affects the `gprestore`, `gpdbrestore`, and `pg_restore` utilities.
168548176	gpbackup	When using `gpbackup` to back up a Greenplum Database 5.7.1 or earlier 5.x release with resource groups enabled, `gpbackup` returns a column not found error for `t6.value AS memoryauditor`.
168689202	PXF	PXF fails to run any query on Java 11 that specifies a `Hive*` profile due to this Hive known issue: ClassCastException when initializing HiveMetaStoreClient on JDK10 or newer. **Workaround:** Run PXF on Java 8 *or* use the PXF JDBC Connector to access Hive.
168957894	PXF	The PXF Hive Connector does not support using the `Hive*` profiles to access Hive transactional tables. **Workaround:** Use the PXF JDBC Connector to access Hive.

Table 1. Key Known Issues in Pivotal Greenplum 5.x

Issue	Category	Description
169052763	gprestore	You can create a full backup of a database with `gpbackup` using the `--with-stats` option to back up table statistics. However, when you try to restore only some of the tables and the statistics for the tables using `gprestore` with a table filter option and the `--with-stats` option, `gprestore` attempts to restore all the table statistics from the backup, not just the statistics for the tables being restored. When restoring all the table statistics, if a table is not in the target database, `gprestore` returns an error. If a table exists in the database, `gprestore` replaces the existing table statistics.
169200795	Greenplum Stream Server	When loading Kafka data into Greenplum Database in `UPDATE` and `MERGE` modes, GPSS requires that a `MAPPING` exist for each column name identified in the `MATCH_COLUMNS` and `UPDATE_COLUMNS` lists.
170202002	Greenplum-Kafka Integration	Updating the `METADATA:SCHEMA` property and restarting a previously-run load job could cause `gpkafka` to re-read Kafka messages published to the topic, and load duplicate messages into Greenplum Database.
26675	gpcrondump	During the transition from Daylight Saving Time to Standard Time, this sequence of events might cause a `gpcrondump` backup operation to fail. If an initial backup is taken between 1:00AM and 2:00AM Daylight Saving Time, and a second backup is taken between 1:00AM and 2:00AM Standard Time, the second backup might fail if the first backup has a timestamp newer than the second. Pivotal recommends performing only a single backup between the hours of 1:00AM and 2:00AM on the days when the time changes: • November 1, 2020 • November 7, 2021 • November 6, 2022 If the failure scenario is encountered, it can be remedied by restarting the backup operation after 2:00AM Standard Time.

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 full text search and text analysis using Pivotal GPText
- Support for data connectors:
  - Greenplum-Spark Connector
  - Greenplum-Informatica Connector
  - Greenplum-Kafka Integration
  - Gemfire-Greenplum Connector
  - Greenplum Stream Server
- Data Direct ODBC/JDBC Drivers

- `gpcopy` utility for copying or migrating objects between Greenplum systems
- Greenplum backup plugin for DD Boost
- Backup/restore storage plugin API (Beta)

Supported Platforms

Pivotal Greenplum 5.26.0 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 (See the following Note)
- SuSE Linux Enterprise Server 64-bit 12 SP2 and SP3 with kernel version greater than 4.4.73-5. (See the following Note)
- SuSE Linux Enterprise Server 64-bit 11 SP4 (See the following Note)
- CentOS 64-bit 7.x
- CentOS 64-bit 6.x (See the following Note)
- Oracle Linux 64-bit 7.4, using the Red Hat Compatible Kernel (RHCK)

Note: For the supported Linux operating systems, Pivotal Greenplum Database is supported on system hosts using either AMD or Intel CPUs based on the x86-64 architecture. Pivotal recommends using a homogeneous set of hardware (system hosts) in a Greenplum Database system.
Important: Significant Greenplum Database performance degradation has been observed when enabling resource group-based workload management on Red Hat 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.

If you use Red Hat 6 and the performance with resource groups is acceptable for your use case, upgrade your kernel to version 2.6.32-696 or higher to benefit from other fixes to the cgroups implementation.

SuSE 11 does not have a kernel version that resolves this issue; resource groups are still considered to be a Beta feature on this platform. Resource groups are not supported on SuSE 11 for production use. See known issue 149789783.

Pivotal Greenplum on SuSE 12 supports resource groups for production use. SuSE 12 resolves the Linux cgroup kernel issues that caused the performance degradation when Greenplum Database resource groups are enabled.

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 these features.

- The PL/Perl procedural language
- The `gpmapreduce` tool
- The PL/Container language extension
- The Greenplum Platform Extension Framework (PXF)

Note: PL/Container is not supported on RHEL/CentOS 6.x systems, because those platforms do not officially support Docker.
Greenplum Database support on Dell EMC DCA.

- Pivotal Greenplum Database 5.26.0 is supported on DCA systems that are running DCA software version 3.4 or greater.
- Only Pivotal Greenplum Database is supported on DCA systems. Open source versions of Greenplum Database are not supported.

- FIPS is supported on DCA software version 3.4 and greater with Pivotal Greenplum Database 5.2.0 and greater.

Note: These Greenplum Database releases are not certified on DCA because of an incompatibility in configuring timezone information.

5.5.0, 5.6.0, 5.6.1, 5.7.0, 5.8.0

These Greenplum Database releases are certified on DCA.

5.7.1, 5.8.1, 5.9.0 and later releases, and 5.x releases prior to 5.5.0.

Pivotal Greenplum 5.26.0 supports these Java versions:

- 8.*xxx*
- 7.*xxx*

Greenplum Database 5.26.0 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 5.26.0 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 2. Data Domain Boost Compatibility

Pivotal Greenplum	Data Domain Boost[2]	DDOS
5.20.x	3.3	6.1 (all versions)
5.19.0	3.0.0.3[1]	6.0 (all versions)
5.18.0		
5.17.0		
5.16.0		
5.14.0		
5.13.0		
5.12.0		
5.11.1		
5.11.0		
5.10.2		
5.9.0		
5.8.1		
5.8.0		
5.7.1		
5.7.0		
5.4.1		
5.4.0		
5.2.0		
5.1.0		
5.0.0		

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.

[1]Support for Data Domain Boost 3.0.0.3 is deprecated. The DELL EMC end of Primary Support date

is December 31, 2017.

[2]The Greenplum Database utilities `gpbackup` and `gprestore` support Data Domain DD Boost File System Plugin (BoostFS) v1.1 with DDOS 6.0 or greater.

The `gpbackup` and `gprestore` utilities support using Dell EMC Data Domain Boost software with the DD Boost Storage Plugin.

Note: Pivotal Greenplum 5.26.0 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 5.26.0 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.

- Pivotal Greenplum is supported using either IPV4 or IPV6 protocols.
- 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 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), Microsoft Azure, and Google Cloud Compute (GCP).
    - **AWS** - For production workloads, r4.8xlarge and r4.16xlarge instance types with four 12TB ST1 EBS volumes for each segment host, or d2.8xlarge with ephemeral storage configured with 4 RAID 0 volumes, are supported. EBS storage is recommended. EBS storage is more reliable and provides more features than ephemeral storage. Note that Amazon has no provisions to replace a bad ephemeral drive; when a disk failure occurs, you must replace the node with the bad disk.

      Pivotal recommends using an Auto Scaling Group (ASG) to provision nodes in AWS. An ASG automatically replaces bad nodes, and you can add further automation to recover the Greenplum processes on the new nodes automatically.

      Deployments should be in a Placement Group within a single Availability Zone. Because Amazon recommends using the same instance type in a Placement Group, use a single instance type for all nodes, including the masters.
    - **Azure** - For production workloads, Pivotal recommends configuring Standard_H8 instance type with 4 2TB disks and 2 segments per host and recommend using 8 2TB

disks and 4 segments per host with Standard_H16 instance type. Standard_H16 uses 8 2TB disks and 4 segments per host. This means software RAID 0 is required so that the number of volumes do not exceed the number of segments.
For Azure deployments, you must also configure the Greenplum Database system to not use port 65330. Add the following line to the `sysctl.conf` file on all Greenplum Database hosts.

```
$net.ipv4.ip_local_reserved_ports=65330
```

- **GCP** - For all workloads, n1-standard-8 and n1-highmem-8 are supported which are relatively small instance types. This is because of the disk performance in GCP forces the configuration to have just 2 segments per host but with many hosts to scale. Use pd-standard disks and the size of the disk is recommended to be 6 TB. For performance perspective, use a factor of 8 when determining how many nodes to deploy in GCP, so a 16 segment host cluster in AWS would require 128 nodes in GCP.

- 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 `gpinitsystem` 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

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 3. Pivotal Greenplum 5.26.0 Tools Compatibility

Tool	Description of Contents	Tool Version(s)	Server Version(s)
Pivotal Greenplum Clients	Greenplum Database Command-Line Interface (psql)	5.8	5.x
Pivotal Greenplum Loaders	Greenplum Database Parallel Data Loading Tools (gpfdist, gpload)	5.8	5.x
Pivotal Greenplum Command Center	Greenplum Database management tool	4.7	5.19 and later
		4.0.0	5.7.0 and later
		3.3.2	5.0.0 and later
		3.2.2	5.0.0 - 5.2.x
Pivotal Greenplum Workload Manager[1]	Greenplum Database query monitoring and management tool	1.8.0	5.0.0

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

- AIX 7.2 (64-bit) (Client and Load Tools only)[2]

- Red Hat Enterprise Linux x86_64 7.x (RHEL 7)
- Red Hat Enterprise Linux x86_64 6.x (RHEL 6)
- SuSE Linux Enterprise Server x86_64 SLES 11 SP4, or SLES 12 SP2/SP3
- 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)

Note: [1]For Pivotal Greenplum Command Center 4.0.0 and later, workload management is an integrated Command Center feature rather than the separate tool Pivotal Greenplum Workload Manager.

[2]For Greenplum Database 5.4.1 and earlier 5.x releases, download the AIX Client and Load Tools package either from the Greenplum Database 5.11.1 file collection or the Greenplum Database 5.0.0 file collection on Pivotal Network.

Extensions

Table 4. Pivotal Greenplum 5.26.0 Extensions Compatibility

Pivotal Greenplum Extension	Versions
MADlib machine learning for Greenplum Database 5.x[1]	MADlib 1.17, 1.16, 1.15.1, 1.15, 1.14
PL/Java for Greenplum Database 5.x	PL/Java 1.4.3[2]
PL/R for Greenplum Database 5.x	2.3.3
PostGIS Spatial and Geographic Objects for Greenplum Database 5.x	2.1.5+pivotal.2
Python Data Science Module Package for Greenplum Database 5.x[3]	1.1.1, 1.1.0, 1.0.0
R Data Science Library Package for Greenplum Database 5.x[4]	1.0.1, 1.0.0
PL/Container for Greenplum Database 5.x	1.1[5], 1.2[6], 1.3, 1.4, 1.5, 1.6

Note: [1]Pivotal recommends that you upgrade to the most recent version of MADlib. For information about MADlib support and upgrade information, see the MADlib FAQ. For information on installing the MADlib extension in Greenplum Database, see Greenplum MADlib Extension for Analytics.

[2]The PL/Java extension package version 1.4.3 is compatible only with Greenplum Database 5.11.0 and later, it is not compatible with 5.10.x or earlier. If you are upgrading from Greenplum Database 5.10.x or earlier and have installed PL/Java 1.4.2, you must upgrade the PL/Java extension to version 1.4.3.

[3]For information about the Python package, including the modules provided, see the Python Data Science Module Package.

[4]For information about the R package, including the libraries provided, see the R Data Science Library Package.

[5]To upgrade from PL/Container 1.0 to PL/Container 1.1 and later, you must drop the PL/Container 1.0 language before registering the new version of PL/Container. For information on upgrading the PL/Container extension in Greenplum Database, see PL/Container Language Extension.

[6]PL/Container version 1.2 can utilize the resource group capabilities that were introduced in Greenplum Database 5.8.0. If you downgrade to a Greenplum Database system that uses PL/Container 1.1 or earlier, you must use `plcontainer runtime-edit` to remove any `resource_group_id` settings from the PL/Container runtime configuration file. See Upgrading from PL/Container 1.1.

These Greenplum Database extensions are installed with Pivotal Greenplum Database

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

Pivotal Greenplum Data Connectors

- Greenplum Platform Extension Framework (PXF) - PXF, integrated with Greenplum Database, provides access to HDFS, Hive, HBase, and SQL external data stores. Refer to Accessing External Data with PXF in the *Greenplum Database Administrator Guide* for PXF configuration and usage information.
  Note: PXF is available only for supported Red Hat and CentOS platforms. PXF is not available for supported SuSE platforms.
- 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.
- Greenplum-Informatica Connector - The Pivotal Greenplum-Informatica connector supports high speed data transfer from an Informatica PowerCenter cluster to a Pivotal Greenplum Database cluster for batch and streaming ETL operations. See the Pivotal Greenplum-Informatica Connector Documentation.
- Greenplum-Kafka Integration - The Pivotal Greenplum-Kafka Integration provides high speed, parallel data transfer from a Kafka cluster to a Pivotal Greenplum Database cluster for batch and streaming ETL operations. Refer to the Pivotal Greenplum-Kafka Integration Documentation for more information about this feature.
- Greenplum Stream Server - The Pivotal Greenplum Stream Server is an ETL tool that provides high speed, parallel data transfer from Informatica, Kafka, and custom client data sources to a Pivotal Greenplum Database cluster. Refer to the Performing ETL Operations with the Pivotal Greenplum Stream Server Documentation for more information about this feature.
- Gemfire-Greenplum Connector - The Pivotal Gemfire-Greenplum Connector supports the transfer of data between a GemFire region and a Greenplum Database cluster. The Gemfire-Greenplum Connector is available as a separate download from Pivotal Network. Refer to the Gemfire-Greenplum Connector documentation for compatibility and usage information.

Pivotal GPText Compatibility

Pivotal Greenplum Database 5.26.0 is compatible with Pivotal GPText version 2.1.3 and later.

Pivotal Greenplum Command Center

See the Greenplum Command Center documentation for GPCC and Greenplum Workload Manager compatibility information, see the Pivotal Greenplum Command Center 3.x and 2.x Release Notes.
Note: For Pivotal Greenplum Command Center 4.0.0 and later, workload management is an integrated Command Center feature rather than the separate tool Pivotal Greenplum Workload Manager.

Hadoop Distribution Compatibility

Greenplum Database provides access to HDFS with `gphdfs` and the Greenplum Platform Extension Framework (PXF).

PXF Hadoop Distribution Compatibility

PXF can use Cloudera, Hortonworks Data Platform, MapR, and generic Apache Hadoop distributions. PXF bundles all of the JAR files on which it depends, and includes and supports the following

Hadoop library versions:

Table 5. PXF Hadoop Supported Platforms

PXF Version	Hadoop Version	Hive Server Version	HBase Server Version
5.10, 5.11	2.x, 3.1+	1.x, 2.x, 3.1+	1.3.2
<= 5.8.2	2.x	1.x	1.3.2

If you plan to access JSON format data stored in a Cloudera Hadoop cluster, PXF requires a Cloudera version 5.8 or later Hadoop distribution.

gphdfs Hadoop Distribution Compatibility

The supported Hadoop distributions for `gphdfs` are listed below:

Table 6. Supported `gphdfs` Hadoop Distributions

Hadoop Distribution	Version	gp_hadoop_ target_version
Cloudera	CDH 5.x	`cdh`
Hortonworks Data Platform	HDP 2.x	`hdp`
MapR	MapR 4.x, MapR 5.x	`mpr`
Apache Hadoop	2.x	`hadoop`

Note: MapR requires the MapR client.

Upgrading to Greenplum Database 5.26.0

The upgrade path supported for this release is Greenplum Database 5.x to Greenplum Database 5.26.0. Upgrading a Greenplum Database 4.3.x release to Pivotal Greenplum 5.x is not supported. See Migrating Data to Pivotal Greenplum 5.x.

Note: If you are upgrading Greenplum Database on a DCA system, see Pivotal Greenplum on DCA Systems.
Important: Pivotal recommends that customers set the Greenplum Database timezone to a value that is compatible with their host systems. Setting the Greenplum Database timezone prevents Greenplum Database from selecting a timezone each time the cluster is restarted and sets the timezone for the Greenplum Database master and segment instances. After you upgrade to this release and if you have not set a Greenplum Database timezone value, verify that the selected Greenplum Database timezone is acceptable for your deployment. See Configuring Timezone and Localization Settings for more information.

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 to Greenplum Database 5.26.0 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.

If you have configured the Greenplum Platform Extension Framework (PXF) in your previous Greenplum Database installation, you must stop the PXF service, and you might need to back up PXF configuration files before upgrading to a new version of Greenplum Database. Refer to PXF Pre-Upgrade Actions for instructions.

If you do not plan to use PXF, or you have not yet configured PXF, no action is necessary.

Upgrading from 5.x to 5.26.0

An upgrade from 5.x to 5.26.0 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 you are upgrading from Greenplum Database 5.10.x or earlier and have installed the PL/Java extension, you must upgrade the PL/Java extension to extension package version 1.4.3. Previous releases of the PL/Java extension are not compatible with Greenplum Database 5.11.0 and later. For information about the PL/Java extension package, see Pivotal Greenplum Tools and Extensions Compatibility.
Note: If you have databases that were created with Greenplum Database 5.10.x or an earlier 5.x release, upgrade the `gp_bloat_expected_pages` view in the `gp_toolkit` schema. For information about the issue and how check a database for the issue, see Update for gp_toolkit.gp_bloat_expected_pages Issue.
Note: If you are upgrading from Greenplum Database 5.7.0 or an earlier 5.x release and have configured PgBouncer in your Greenplum Database installation, you must migrate to the new PgBouncer when you upgrade Greenplum Database. Refer to Migrating PgBouncer for specific migration instructions.
Note: If you have databases that were created with Greenplum Database 5.3.0 or an earlier 5.x release, upgrade the `gp_bloat_diag`function and view in the `gp_toolkit` schema. For information about the issue and how check a database for the issue, see Update for gp_toolkit.gp_bloat_diag Issue.
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 5.26.0 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-5.26.0
   ```

   If you install Greenplum Database with the rpm (as `root`), the installation directory is `/usr/local/greenplum-db-5.26.0`.

   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 needed, update the `greenplum_path.sh` file for use with your specific installation. These are some examples.

   - If Greenplum Database uses LDAP authentication, edit the `greenplum_path.sh` file to add the line:

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

   - If Greenplum Database uses PL/Java, you might need to set or update the environment variables `JAVA_HOME` and `LD_LIBRARY_PATH` in `greenplum_path.sh`.

   Note: When comparing the previous and new `greenplum_path.sh` files, be aware that installing some Greenplum Database extensions also updates the `greenplum_path.sh` file. The `greenplum_path.sh` from the previous release might contain updates that were the result of those extensions. See step 9 for installing Greenplum Database extensions.

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-5.26.0/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-5.26.0 /usr/local/greenplum-db
   ```

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

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

7. Run the `gpseginstall` utility to install the 5.26.0 binaries on all the segment hosts specified

in the *hostfile*. For example:

```
$ gpseginstall -f hostfile
```

Note: The `gpseginstall` utility copies the installed files from the current host to the remote hosts. It does not use `rpm` to install Greenplum Database on the remote hosts, even if you used `rpm` to install Greenplum Database on the current host.

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.

   Also copy any additional files that are used by the extensions (such as JAR files, shared object files, and libraries) from the previous version installation directory to the new version installation directory on the master and segment host systems.

9. If you are upgrading from Greenplum Database 5.7 or an earlier 5.x release and have configured PgBouncer in your Greenplum Database installation, you must migrate to the new PgBouncer when you upgrade Greenplum Database. Refer to Migrating PgBouncer for specific migration instructions.

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

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

11. If you are utilizing Data Domain Boost, you have to re-enter your DD Boost credentials after upgrading from Greenplum Database to 5.26.0 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.

12. If you configured PXF in your previous Greenplum Database installation, you must re-initialize the PXF service after you upgrade Greenplum Database. Refer to Upgrading PXF for instructions.

After upgrading Greenplum Database, ensure features work as expected. For example, you should test that backup and restore perform as expected, and Greenplum Database features such as user-defined functions, and extensions such as MADlib and PostGIS perform as expected.

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.

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`. The `gpcopy` utility can be used to migrate data from Greenplum Database 4.3.26 or later to 5.9 or later.

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.
  Important: For Greenplum Database 5.x, Pivotal recommends that customers set the Greenplum Database timezone to a value that is compatible with the host systems. Setting the Greenplum Database timezone prevents Greenplum Database from selecting a timezone each time the cluster is restarted and sets the timezone for the Greenplum Database master and segment instances. See Configuring Timezone and Localization Settings for more information.
- 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) in order to migrate data with `gpcrondump` and `gpdbrestore`. 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.

  If you attempt to migrate your data on the same hardware but run out of free space, `gpcopy` provides the `--truncate-source-after` option to truncate each source table after copying the table to the destination cluster and validating the copy succeeded. This reduces the amount of free space needed to migrate clusters that reside on the same hardware. See Migrating Data with gpcopy for more information.
- 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, and the clusters have different numbers of segments, 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 `gptransfer` 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.
- Greenplum Database 5.x removes automatic implicit casts between the text type and other data types. After you migrate from Greenplum Database version 4.3.x to version 5.x, this change in behavior may impact existing applications and queries. Refer to About Implicit Text Casting in Greenplum Database in the *Greenplum Database Installation Guide* for information, including a discussion about supported and unsupported workarounds.
- 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.

- If you are migrating from Greenplum Database 4.3.27 or an earlier 4.3.x release and have configured PgBouncer in your Greenplum Database installation, you must migrate to the new PgBouncer when you upgrade Greenplum Database. Refer to Migrating PgBouncer for specific migration instructions.

Pivotal Greenplum on DCA Systems

On supported Dell EMC DCA systems, you can install Pivotal Greenplum 5.26.0, or you can upgrade from Pivotal Greenplum 5.x to 5.26.0.

Only Pivotal Greenplum Database is supported on DCA systems. Open source versions of Greenplum Database are not supported.

- Installing the Pivotal Greenplum 5.26.0 Software Binaries on DCA Systems
- Upgrading from 5.x to 5.26.0 on DCA Systems

Important: Upgrading Pivotal Greenplum Database 4.3.x to Pivotal Greenplum 5.26.0 is not supported. See Migrating Data to Pivotal Greenplum 5.x.
Note: These Greenplum Database releases are not certified on DCA because of an incompatibility in configuring timezone information.

5.5.0, 5.6.0, 5.6.1, 5.7.0, 5.8.0

These Greenplum Database releases are certified on DCA.

5.7.1, 5.8.1, 5.9.0 and later releases, and 5.x releases prior to 5.5.0.

Installing the Pivotal Greenplum 5.26.0 Software Binaries on DCA Systems

Important: This section is for installing Pivotal Greenplum 5.26.0 only on DCA systems. Also, see the information on the DELL EMC support site (requires login).

For information about installing Pivotal Greenplum on non-DCA systems, see the *Greenplum Database Installation Guide*.

Prerequisites

- Ensure your DCA system supports Pivotal Greenplum 5.26.0. See Supported Platforms.
- Ensure Greenplum Database 4.3.x is not installed on your system.

  Installing Pivotal Greenplum 5.26.0 on a DCA system with an existing Greenplum Database 4.3.x installation is not supported. For information about uninstalling Greenplum Database software, see your Dell EMC DCA documentation.

Installing Pivotal Greenplum 5.26.0

1. Download or copy the Greenplum Database DCA installer file `greenplum-db-appliance-5.26.0-RHEL6-x86_64.bin` to the Greenplum Database master host.
2. As root, run the DCA installer for 5.26.0 on the Greenplum Database master host and specify the file `hostfile` that lists all hosts in the cluster, one host name per line. If necessary, copy `hostfile` to the directory containing the installer before running the installer.

   This example command runs the installer for Greenplum Database 5.26.0.

   ```
   # ./greenplum-db-appliance-5.26.0-RHEL6-x86_64.bin hostfile
   ```

Upgrading from 5.x to 5.26.0 on DCA Systems

Upgrading Pivotal Greenplum from 5.x to 5.26.0 on a Dell EMC DCA system involves stopping Greenplum Database, updating the Greenplum Database software binaries, and restarting Greenplum Database.

Important: This section is only for upgrading to Pivotal Greenplum 5.26.0 on DCA systems. For information about upgrading on non-DCA systems, see Upgrading to Greenplum Database 5.26.0.
Note: If you are upgrading from Greenplum Database 5.10.x or earlier and have installed the PL/Java extension, you must upgrade the PL/Java extension to extension package version 1.4.3. Previous releases of the PL/Java extension are not compatible with Greenplum Database 5.11.0 and later. For information about the PL/Java extension package, see Pivotal Greenplum Tools and Extensions Compatibility.
Note: If you have databases that were created with Greenplum Database 5.10.x or an earlier 5.x release, upgrade the `gp_bloat_expected_pages` view in the `gp_toolkit` schema. For information about the issue and how check a database for the issue, see Update for gp_toolkit.gp_bloat_expected_pages Issue.
Note: If you are upgrading from Greenplum Database 5.7.0 or an earlier 5.x release and have configured PgBouncer in your Greenplum Database installation, you must migrate to the new PgBouncer when you upgrade Greenplum Database. Refer to Migrating PgBouncer for specific migration instructions.
Note: If you have databases that were created with Greenplum Database 5.3.0 or an earlier 5.x release, upgrade the `gp_bloat_diag`function and view in the `gp_toolkit` schema. For information about the issue and how check a database for the issue, see Update for gp_toolkit.gp_bloat_diag Issue.

1. Log in to your Greenplum Database master host as the Greenplum administrative user (`gpadmin`):

   ```
   # su - gpadmin
   ```

2. Download or copy the installer file `greenplum-db-appliance-5.26.0-RHEL6-x86_64.bin` to the Greenplum Database master host.

3. 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
   ```

4. As root, run the Greenplum Database DCA installer for 5.26.0 on the Greenplum Database master host and specify the file `hostfile` that lists all hosts in the cluster. If necessary, copy `hostfile` to the directory containing the installer before running the installer.

   This example command runs the installer for Greenplum Database 5.26.0 for Red Hat Enterprise Linux 6.x.

   ```
   # ./greenplum-db-appliance-5.26.0-RHEL6-x86_64.bin hostfile
   ```

   The file `hostfile` is a text file that lists all hosts in the cluster, one host name per line.

5. If needed, update the `greenplum_path.sh` file for use with your specific installation. These are some examples.

   - If Greenplum Database uses LDAP authentication, edit the `greenplum_path.sh` file to add the line:

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

   - If Greenplum Database uses PL/Java, you might need to set or update the environment variables `JAVA_HOME` and `LD_LIBRARY_PATH` in `greenplum_path.sh`.

   Note: When comparing the previous and new `greenplum_path.sh` files, be aware that installing some Greenplum Database extensions also updates the `greenplum_path.sh` file.

The `greenplum_path.sh` from the previous release might contain updates that were the result of those extensions. See step 6 for installing Greenplum Database extensions.

6. Install Greenplum Database extension packages. For information about installing a Greenplum Database extension package, see `gppkg` in the *Greenplum Database Utility Guide*.

   Also migrate any additional files that are used by the extensions (such as JAR files, shared object files, and libraries) from the previous version installation directory to the new version installation directory.

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

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

8. If you are utilizing Data Domain Boost, you have to re-enter your DD Boost credentials after upgrading to Greenplum Database 5.26.0 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.

After upgrading Greenplum Database, ensure features work as expected. For example, you should test that backup and restore perform as expected, and Greenplum Database features such as user-defined functions, and extensions such as MADlib and PostGIS perform as expected.

Update for gp_toolkit.gp_bloat_expected_pages Issue

In Greenplum Database 5.10.x and earlier 5.x releases, the Greenplum Database view `gp_toolkit.gp_bloat_expected_pages` view might incorrectly report that a root partition table is bloated even though root partition tables do not contain data. This information could cause a user to run a `VACUUM FULL` operation on the partitioned table when the operation was not required. The issue was resolved in Greenplum Database 5.11.0 (resolved issue 29523) .

When updating Greenplum Database, the `gp_toolkit.gp_bloat_expected_pages` view must be updated in databases created with a Greenplum Database 5.10.x or an earlier 5.x release. This issue has been fixed in databases created with Greenplum Database 5.11.0 and later. For information about using `template0` as the template database after upgrading from Greenplum Database 5.10.x or an earlier 5.x release, see known issue 29523.

To check whether the `gp_toolkit.gp_bloat_expected_pages` view in a database requires an update, run the `psql` command `\d+` to display the view definition.

```
\d+ gp_toolkit.gp_bloat_expected_pages
```

The updated view definition contains this predicate.

```
AND NOT EXISTS
( SELECT parrelid
     FROM pg_partition
     WHERE parrelid = pgc.oid )
```

Perform the following steps as the gpadmin user to update the view on each database that was created with Greenplum Database 5.11.0 or an earlier 5.x release.

1. Copy the script into a text file on the Greenplum Database master.
2. Run the script on each database that requires the update.

This example updates `gp_toolkit.gp_bloat_expected_pages` view in the database `mytest` and assumes that the script is in the `gp_bloat_expected_pages` in the `gpadmin` home directory.

```
psql -f /home/gpadmin/gp_bloat_expected_pages.sql -d mytest
```

Run the script during a low activity period. Running the script during a high activity period does not affect database functionality but might affect performance.

Script to Update `gp_toolkit.gp_bloat_expected_pages` View

```
BEGIN;
CREATE OR REPLACE VIEW gp_toolkit.gp_bloat_expected_pages
AS
    SELECT
      btdrelid,
      btdrelpages,
       CASE WHEN btdexppages < numsegments
          THEN numsegments
          ELSE btdexppages
        END as btdexppages
    FROM
    ( SELECT
        oid as btdrelid,
        pgc.relpages as btdrelpages,
        CEIL((pgc.reltuples * (25 + width))::numeric / current_setting('block_size')::
numeric) AS btdexppages,
        (SELECT numsegments FROM gp_toolkit.__gp_number_of_segments) AS numsegments
        FROM
        ( SELECT pgc.oid, pgc.reltuples, pgc.relpages
            FROM pg_class pgc
            WHERE NOT EXISTS
            ( SELECT iaooid
                FROM gp_toolkit.__gp_is_append_only
                WHERE iaooid = pgc.oid AND iaotype = 't' )
            AND NOT EXISTS
            ( SELECT parrelid
                FROM pg_partition
                WHERE parrelid = pgc.oid )) AS pgc
        LEFT OUTER JOIN
          ( SELECT  starelid, SUM(stawidth * (1.0 - stanullfrac)) AS width
              FROM pg_statistic pgs
              GROUP BY 1) AS btwcols
        ON pgc.oid = btwcols.starelid
        WHERE starelid IS NOT NULL) AS subq;

GRANT SELECT ON TABLE gp_toolkit.gp_bloat_expected_pages TO public;
COMMIT;
```

Update for gp_toolkit.gp_bloat_diag Issue

In Greenplum Database 5.3.0 or an earlier 5.x release, Greenplum Database returned an integer out of range error in some cases when performing a query against the `gp_toolkit.gp_bloat_diag` view. The issue was resolved in Greenplum Database 5.4.0 (resolved issue 26518) .

When updating Greenplum Database, the `gp_toolkit.gp_bloat_diag` function and view must be updated in databases created with a Greenplum Database 5.3.0 or an earlier 5.x release. This issue has been fixed in databases created with Greenplum Database 5.4.0 and later. For information about upgrading from Greenplum Database 5.3.0 or an earlier 5.x release and then using `template0` as the template database, see known issue 29523.

To check whether the `gp_toolkit.gp_bloat_diag` function and view in a database requires an update, run the `psql` command `\df` to display information about the `gp_toolkit.gp_bloat_diag` function.

```
\df gp_toolkit.gp_bloat_diag
```

If the data type for `btdexppages` is `integer`, an update is required. If the data type is `numeric` an update is not required. In this example, the `btdexppages` data type is `integer` and requires an update.

```
List of functions
-[ RECORD 1 ]-------+---------------------------------------------------------------------
------------------------------
Schema              | gp_toolkit
Name                | gp_bloat_diag
Result data type    | record
Argument data types | btdrelpages integer, btdexppages integer, aotable boolean, OUT bltid
x integer, OUT bltdiag text
Type                | normal
```

Perform the following steps as the gpadmin user to update the function and view to fix the issue on each database that was created with Greenplum Database 5.3.0 or an earlier 5.x release.

1. Copy the script into a text file on the Greenplum Database master.
2. Run the script on each database that requires the update.
   This example updates `gp_toolkit.gp_bloat_diag` function and view in the database `mytest` and assumes that the script is in the `update_bloat_diag.sql` in the `gpadmin` home directory.

   ```
   psql -f /home/gpadmin/update_bloat_diag.sql -d mytest
   ```

Run the script during a low activity period. Running the script during a high activity period does not affect database functionality but might affect performance.

Script to Update `gp_toolkit.gp_bloat_diag` Function and View

```
BEGIN;
CREATE OR REPLACE FUNCTION gp_toolkit.gp_bloat_diag(btdrelpages int, btdexppages numer
ic, aotable bool,
    OUT bltidx int, OUT bltdiag text)
AS
$$
    SELECT
        bloatidx,
        CASE
            WHEN bloatidx = 0
                THEN 'no bloat detected'::text
            WHEN bloatidx = 1
                THEN 'moderate amount of bloat suspected'::text
            WHEN bloatidx = 2
                THEN 'significant amount of bloat suspected'::text
            WHEN bloatidx = -1
                THEN 'diagnosis inconclusive or no bloat suspected'::text
        END AS bloatdiag
    FROM
    (
        SELECT
            CASE
                WHEN $3 = 't' THEN 0
                WHEN $1 < 10 AND $2 = 0 THEN -1
                WHEN $2 = 0 THEN 2
                WHEN $1 < $2 THEN 0
                WHEN ($1/$2)::numeric > 10 THEN 2
                WHEN ($1/$2)::numeric > 3 THEN 1
                ELSE -1
            END AS bloatidx
    ) AS bloatmapping

$$
LANGUAGE SQL READS SQL DATA;
```

```
GRANT EXECUTE ON FUNCTION gp_toolkit.gp_bloat_diag(int, numeric, bool, OUT int, OUT te
xt) TO public;

CREATE OR REPLACE VIEW gp_toolkit.gp_bloat_diag
AS
    SELECT
        btdrelid AS bdirelid,
        fnnspname AS bdinspname,
        fnrelname AS bdirelname,
        btdrelpages AS bdirelpages,
        btdexppages AS bdiexppages,
        bltdiag(bd) AS bdidiag
    FROM
    (
        SELECT
            fn.*, beg.*,
            gp_toolkit.gp_bloat_diag(btdrelpages::int, btdexppages::numeric, iao.iaoty
pe::bool) AS bd
        FROM
            gp_toolkit.gp_bloat_expected_pages beg,
            pg_catalog.pg_class pgc,
            gp_toolkit.__gp_fullname fn,
            gp_toolkit.__gp_is_append_only iao

        WHERE beg.btdrelid = pgc.oid
            AND pgc.oid = fn.fnoid
            AND iao.iaooid = pgc.oid
    ) as bloatsummary
    WHERE bltidx(bd) > 0;

GRANT SELECT ON TABLE gp_toolkit.gp_bloat_diag TO public;
COMMIT;
```

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.
- **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.
- **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.
- **Validating Your Systems**
  Validate your operating system settings, hardware, and network.
- **Initializing a Greenplum Database System**
  Describes how to initialize a Greenplum Database database system.
- **Installing Optional Extensions**
  Information about installing optional Greenplum Database extensions and packages, such as the Procedural Language extensions and the Python and R Data Science Packages.
- **Configuring Timezone and Localization Settings**
  Describes the available timezone and localization features of Greenplum Database.
- **About Implicit Text Casting in Greenplum Database**
  Greenplum Database version 4.3.x is based on PostgreSQL version 8.2. Greenplum Database version 5.x is based on PostgreSQL version 8.3. PostgreSQL 8.3 removed automatic implicit casts between the `text` type and other data types. When you migrate from Greenplum Database version 4.3.x to version 5.x, this change in behavior might impact existing applications and queries.
- **Installation Management Utilities**
  References for the command-line management utilities used to install and initialize a Greenplum Database system.
- **Greenplum Environment Variables**
  Reference of the environment variables to set for Greenplum Database.

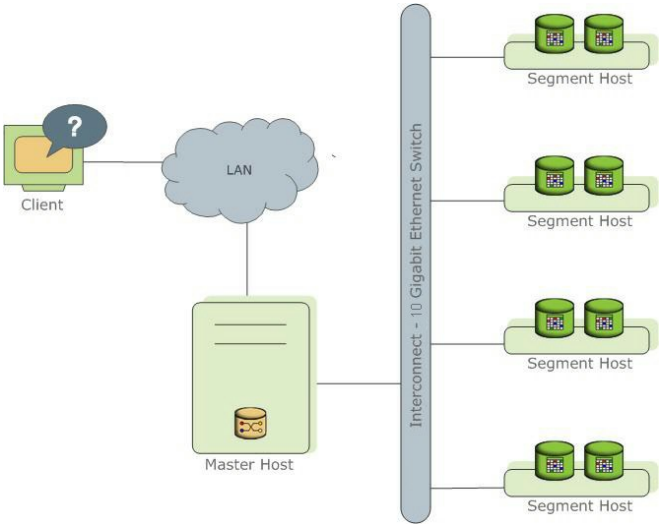
A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Greenplum Database Installation Guide

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.

Parent topic: Introduction to Greenplum

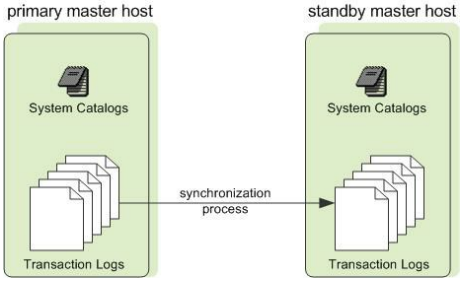
Master Redundancy

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.

Parent topic: Introduction to Greenplum

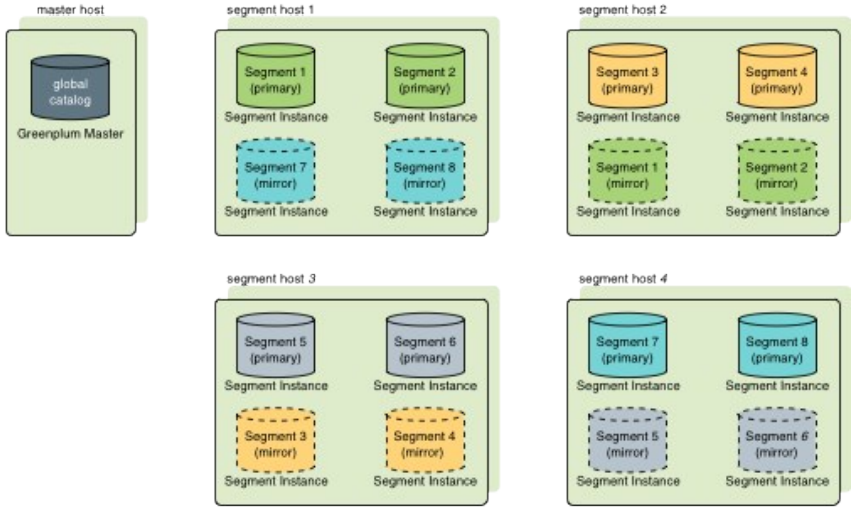
Segment Redundancy

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 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

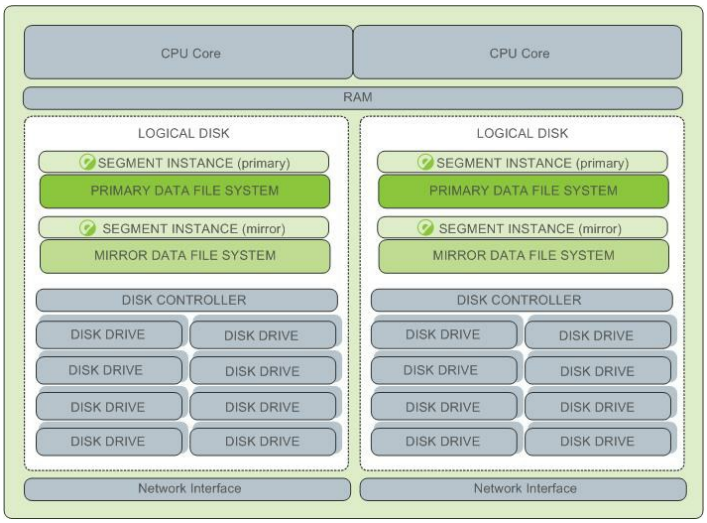
Regardless of the hardware platform you choose, a production Greenplum Database processing node (a segment host) is typically configured as described in this section.

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.

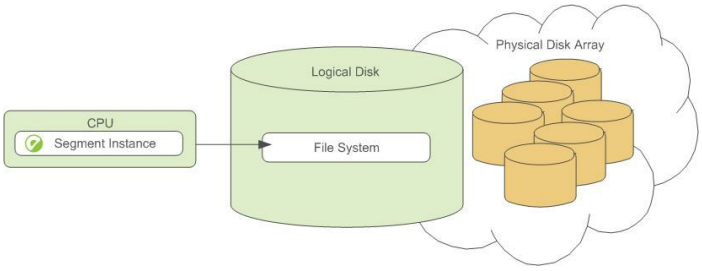
Figure 3. Example Greenplum Database Segment Host Configuration



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*.

Parent topic: Introduction to Greenplum

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

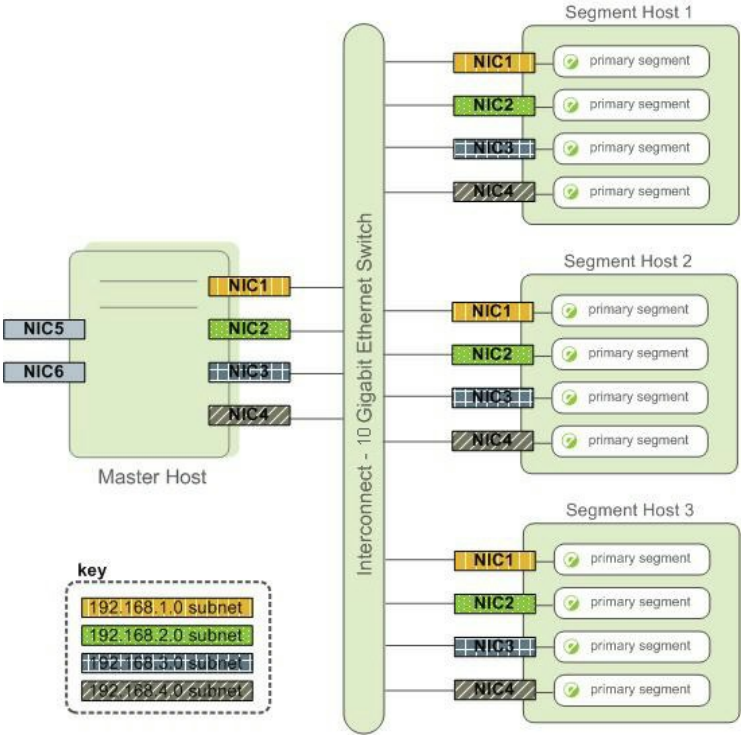
A segment host typically has multiple network interfaces designated to Greenplum interconnect traffic. The master host typically has additional external network interfaces in addition to the interfaces used for interconnect traffic.

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.

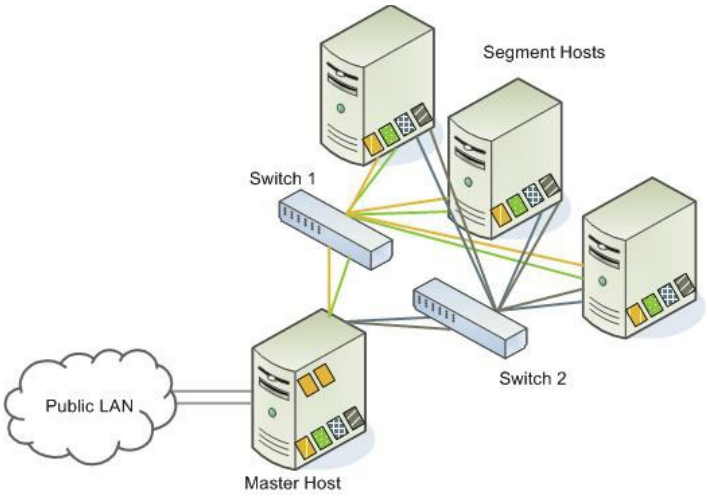
Figure 5. Example Network Interface Architecture



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

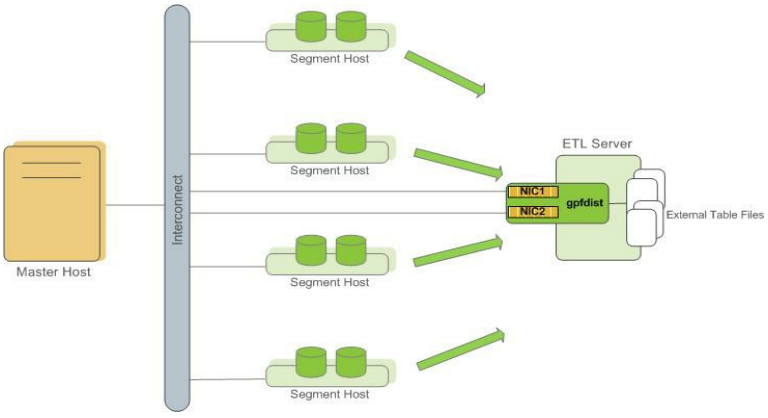
Greenplum supports fast, parallel data loading with its external tables feature. By using external tables in conjunction with Greenplum Database's parallel file server (`gpfdist`), administrators can achieve maximum parallelism and load bandwidth from their Greenplum Database system. Many production 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:

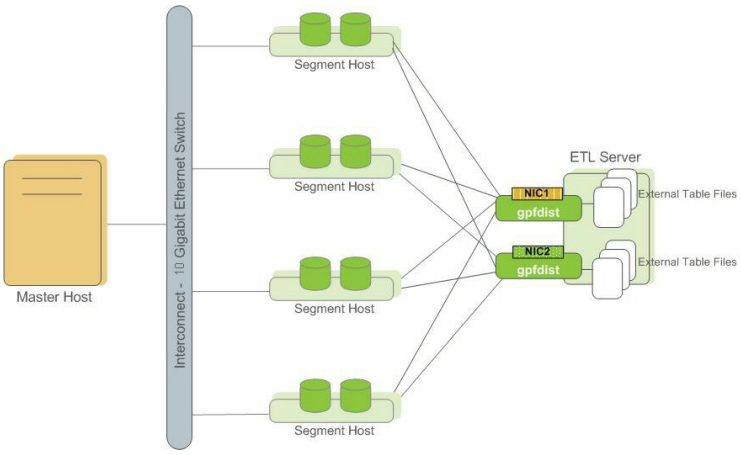
- If your ETL machine is configured with multiple network interface cards (NICs) as described in Network Interface Configuration, run one instance of `gpfdist` on your ETL host and then define your external table definition so that the host name of each NIC is declared in the `LOCATION` clause (see `CREATE 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



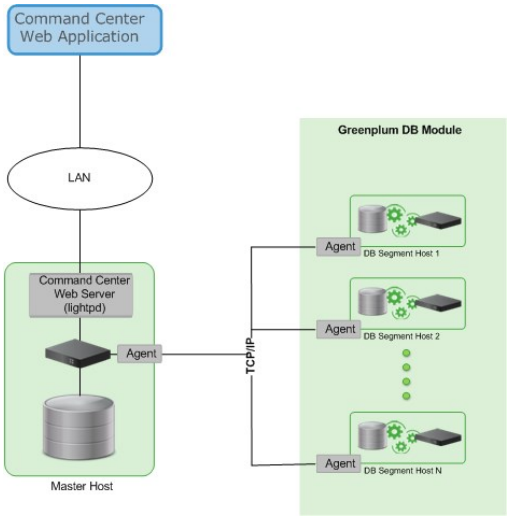
Parent topic: Introduction to Greenplum

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



Parent topic: Introduction to Greenplum

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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**
- **Calculating User Data Size**

- **Calculating Space Requirements for Metadata and Logs**

Parent topic: Greenplum Database Installation Guide

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

Parent topic: Estimating Storage Capacity

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.

  B-tree: *unique_values* * (*data_type_size* + 24 bytes)

  Bitmap: (*unique_values* * *number_of_rows* * 1 bit * *compression_ratio* / 8) + (*unique_values* * 32)

Parent topic: Estimating Storage Capacity

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 * 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 to 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.

Parent topic: Estimating Storage Capacity

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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. Disabling SELinux and Firewall Software
3. Setting the Greenplum Recommended OS Parameters
4. Creating the Greenplum Database Administrative User Account
5. Installing the Greenplum Database Software
6. Installing and Configuring Greenplum on all Hosts
7. Creating the Data Storage Areas
8. Synchronizing System Clocks
9. Next Steps

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

For information about running Greenplum Database in the cloud see *Cloud Services* in the Pivotal Greenplum Partner Marketplace.

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**
- **Disabling SELinux and Firewall Software**
- **Setting the Greenplum Recommended OS Parameters**
- **Installing the Greenplum Database Software**
- **Installing and Configuring Greenplum on all Hosts**
- **Creating the Data Storage Areas**
- **Synchronizing System Clocks**
- **Enabling iptables**
- **Next Steps**

Parent topic: Greenplum Database Installation Guide

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 1. System Prerequisites for Greenplum Database 5.x

Operating System	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	Any x86_64 compatible CPU
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
Network Requirements	10 Gigabit Ethernet within the array NIC bonding is recommended when multiple interfaces are present Pivotal Greenplum is supported using either IPV4 or IPV6 protocols.
Software and Utilities	zlib compression libraries bash shell GNU tar GNU zip GNU sed (used by Greenplum Database `gpinitsystem`) perl secure shell

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, SELinux must be disabled. You should also disable firewall software, although firewall software can be enabled if it is required for security purposes. See Disabling SELinux and Firewall Software.
Parent topic: Configuring Your Systems and Installing Greenplum

Disabling SELinux and Firewall Software

For all Greenplum Database host systems, SELinux must be disabled. Follow these steps:

1. As the root user, check the status of SELinux:

   ```
   # sestatus
   SELinuxstatus: disabled
   ```

2. If SELinux is not disabled, disable it by editing the `/etc/selinux/config` file. As root, change the value of the `SELINUX` parameter in the `config` file as follows:

   ```
   SELINUX=disabled
   ```

3. Reboot the system to apply any changes that you made to `/etc/selinux/config` and verify that SELinux is disabled.

For information about disabling SELinux, see the SELinux documentation.

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). Follow these steps:

1. As the root user, check the status of `iptables`:

   ```
   # /sbin/chkconfig --list iptables
   ```

   If `iptables` is disabled, the command output is:

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

2. If necessary, execute this command as root to disable `iptables`:

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

   You will need to reboot your system after applying the change.

3. For systems with `firewalld`, check the status of `firewalld` with the command:

   ```
   # systemctl status firewalld
   ```

   If `firewalld` is disabled, the command output is:

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

4. If necessary, execute these commands as root to disable `firewalld`:

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

For more information about configuring your firewall software, see the documentation for the firewall or your operating system.

Parent topic: Configuring Your Systems and Installing Greenplum

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.

Parent topic: Configuring Your Systems and Installing Greenplum

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 reload with `sysctl -p`:

  ```
  # kernel.shmall = _PHYS_PAGES / 2 # See Note 1
  kernel.shmall = 197951838
  # kernel.shmmax = kernel.shmall * PAGE_SIZE # See Note 1
  kernel.shmmax = 810810728448
  ```

```
kernel.shmmni = 4096
vm.overcommit_memory = 2
vm.overcommit_ratio = 95 # See Note 2
net.ipv4.ip_local_port_range = 10000 65535 # See Note 3
kernel.sem = 500 2048000 200 4096
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_max_syn_backlog = 4096
net.ipv4.conf.all.arp_filter = 1
net.core.netdev_max_backlog = 10000
net.core.rmem_max = 2097152
net.core.wmem_max = 2097152
vm.swappiness = 10
vm.zone_reclaim_mode = 0
vm.dirty_expire_centisecs = 500
vm.dirty_writeback_centisecs = 100
vm.dirty_background_ratio = 0 # See Note 5
vm.dirty_ratio = 0
vm.dirty_background_bytes = 1610612736
vm.dirty_bytes = 4294967296
```

Note: The listed `sysctl.conf` parameters are for performance in a wide variety of environments. However, the settings might require changes in specific situations. These are additional notes about some of the `sysctl.conf` parameters.

1. Greenplum Database uses shared memory to communicate between `postgres` processes that are part of the same `postgres` instance. `kernel.shmall` sets the total amount of shared memory, in pages, that can be used system wide. `kernel.shmmax` sets the maximum size of a single shared memory segment in bytes.

   Set `kernel.shmall` and `kernel.shmax` values based on your system's physical memory and page size. In general, the value for both parameters should be one half of the system physical memory.

   Use the operating system variables `_PHYS_PAGES` and `PAGE_SIZE` to set the parameters.

   ```
   kernel.shmall = ( _PHYS_PAGES / 2)
   kernel.shmmax = ( _PHYS_PAGES / 2) * PAGE_SIZE
   ```

   To calculate the values for `kernel.shmall` and `kernel.shmax`, run the following commands using the `getconf` command, which returns the value of an operating system variable.

   ```
   $ echo $(expr $(getconf _PHYS_PAGES) / 2)
   $ echo $(expr $(getconf _PHYS_PAGES) / 2 \* $(getconf PAGE_SIZE))
   ```

   As best practice, we recommend you set the following values in the `/etc/sysctl.conf` file using calculated values. For example, a host system has 1583 GB of memory installed and returns these values: _PHYS_PAGES = 395903676 and PAGE_SIZE = 4096. These would be the `kernel.shmall` and `kernel.shmmax` values:

   ```
   kernel.shmall = 197951838
   kernel.shmmax = 810810728448
   ```

   If the Greeplum Database master the has a different shared memory configuration than the segment hosts, the _PHYS_PAGES and PAGE_SIZE values might differ, and the `kernel.shmall` and `kernel.shmax` values on the master host will differ

from those on the segment hosts.

2. When `vm.overcommit_memory` is 2, you specify a value for `vm.overcommit_ratio`. For information about calculating the value for `vm.overcommit_ratio` when using resource queue-based resource management, see the Greenplum Database server configuration parameter `gp_vmem_protect_limit` in the *Greenplum Database Reference Guide*. If you are using resource group-based resource management, tune the operating system `vm.overcommit_ratio` as necessary. If your memory utilization is too low, increase the `vm.overcommit_ratio` value; if your memory or swap usage is too high, decrease the value.

3. 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 the port ranges that are used by Greenplum Database, see gpinitsystem.

4. Azure deployments require Greenplum Database to not use port 65330. Add the following line to sysctl.conf:

   ```
   net.ipv4.ip_local_reserved_ports=65330
   ```

   For additional requirements and recommendations for cloud deployments, see *Greenplum Database Cloud Technical Recommendations*.

5. For host systems with more than 64GB of memory, these settings are recommended:

   ```
   vm.dirty_background_ratio = 0
   vm.dirty_ratio = 0
   vm.dirty_background_bytes = 1610612736 # 1.5GB
   vm.dirty_bytes = 4294967296 # 4GB
   ```

   For host systems with 64GB of memory or less, remove `vm.dirty_background_bytes` and `vm.dirty_bytes` and set the two `ratio` parameters to these values:

   ```
   vm.dirty_background_ratio = 3
   vm.dirty_ratio = 10
   ```

6. Increase `vm.min_free_kbytes` to ensure `PF_MEMALLOC` requests from network and storage drivers are easily satisfied. This is especially critical on systems with large amounts of system memory. The default value is often far too low on these systems. Use this awk command to set `vm.min_free_kbytes` to a recommended 3% of system physical memory:

   ```
   awk 'BEGIN {OFMT = "%.0f";} /MemTotal/ {print "vm.min_free_kbytes =", $2
   * .03;}'
                  /proc/meminfo >> /etc/sysctl.conf
   ```

   Do not set `vm.min_free_kbytes` to higher than 5% of system memory as doing so might cause out of memory conditions.

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

  ```
  * soft nofile 524288
  * hard nofile 524288
  * soft nproc 131072
  * hard nproc 131072
  ```

  For Red Hat Enterprise Linux (RHEL) and CentOS systems, parameter values in the `/etc/security/limits.d/90-nproc.conf` file (RHEL/CentOS 6) or `/etc/security/limits.d/20-nproc.conf` file (RHEL/CentOS 7) override the values in the `limits.conf` file. Ensure that any parameters in the override file are set to the required value. The Linux module `pam_limits` sets user limits by reading the values from the `limits.conf` file and then from the override file. For information about PAM and user limits, see the documentation on PAM and `pam_limits`.

  Execute the `ulimit -u` command on each segment host to display the maximum number of processes that are available to each user. Validate that the return value is 131072.

- 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).

  Note: The `blockdev --setra` command is not persistent. You must ensure the read-ahead value is set whenever the system restarts. How to set the value will vary based on your system.
  One method to set the `blockdev` value at system startup is by adding the `/sbin/blockdev --setra` command in the `rc.local` file. For example, add this line to the `rc.local` file to set the read-ahead value for the disk `sdb`.

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

  On systems that use systemd, you must also set the execute permissions on the `rc.local`

file to enable it to run at startup. For example, on a RHEL/CentOS 7 system, this command sets execute permissions on the file.

```
# chmod +x /etc/rc.d/rc.local
```

Restart the system to have the setting take effect.

- 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
  ```

  Note: Setting the disk I/O scheduler policy with the `echo` command is not persistent, and must be run when the system is rebooted. If you use the `echo` command to set the policy, you must ensure the command is run when the system reboots. How to run the command will vary based on your system.

  One method to set the I/O scheduler policy at boot time is 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  quiet console=tty1
      console=ttyS1,115200 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 user accounts log out. This causes the Greenplum Database utility `gpinitsystem` to fail with semaphore errors. Perform one of the following to avoid this issue.
    - When you add the `gpadmin` operating system user account to the master node in Creating the Greenplum Database Administrative User Account, create the user as a system account. You must also add the `gpadmin` user as a system account on the segment hosts manually or using the `gpseginstall` command (described in later installation step Installing and Configuring Greenplum on all Hosts).
      Note: When you run the `gpseginstall` utility as the `root` user to install Greenplum Database on host systems, the utility creates the `gpadmin` operating system user as a system account on the 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
      ```
- Certain Greenplum Database management utilities including `gpexpand`, `gpinitsystem`, and `gpaddmirrors`, utilize secure shell (SSH) connections between systems to perform their tasks. In large Greenplum Database deployments, cloud deployments, or deployments with a large number of segments per host, these utilities may exceed the host's maximum threshold for unauthenticated connections. When this occurs, you receive errors such as: `ssh_exchange_identification: Connection closed by remote host.`.

  To increase this connection threshold for your Greenplum Database system, update the SSH `MaxStartups` and `MaxSessions` configuration parameters in one of the `/etc/ssh/sshd_config` or `/etc/sshd_config` SSH daemon configuration files.

  If you specify `MaxStartups` and `MaxSessions` using a single integer value, you identify the maximum number of concurrent unauthenticated connections (`MaxStartups`) and maximum number of open shell, login, or subsystem sessions permitted per network connection (`MaxSessions`). For example:

  ```
  MaxStartups 200
  MaxSessions 200
  ```

  If you specify `MaxStartups` using the "*start:rate:full*" syntax, you enable random early connection drop by the SSH daemon. *start* identifies the maximum number of unauthenticated SSH connection attempts allowed. Once *start* number of unauthenticated connection attempts is reached, the SSH daemon refuses *rate* percent of subsequent

connection attempts. *full* identifies the maximum number of unauthenticated connection attempts after which all attempts are refused. For example:

```
Max Startups 10:30:200
MaxSessions 200
```

Restart the SSH daemon after you update `MaxStartups` and `MaxSessions`. For example, on a CentOS 6 system, run the following command as the `root` user:

```
# service sshd restart
```

For detailed information about SSH configuration options, refer to the SSH documentation for your Linux distribution.

- 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 operating system 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"
   GRUB_CMDLINE_LINUX="crashkernel=auto rd.lvm.lv=cl/root rd.lvm.lv=cl/swap rhgb q
   uiet transparent_hugepage=never"
   GRUB_DISABLE_RECOVERY="true"
   ```

2. As root, run the `grub2-mkconfig` command to update the kernels.

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

3. Reboot the system.

Creating the Greenplum Database Administrative User Account

You must create a dedicated operating system user account on the master node to run Greenplum Database. You administer Greenplum Database as this operating system user. This user account is named, by convention, `gpadmin`.

Note: If you are installing the Greenplum Database RPM distribution, create the `gpadmin` user on *every* host in the Greenplum Database cluster because the installer does not create the `gpadmin` user for you. See the note under Installing the Greenplum Database Software for more information.

You cannot run the Greenplum Database server as `root`.

The `gpadmin` user account must have permission to access the services and directories required to install and run Greenplum Database.

To create the `gpadmin` operating system user account, run the `groupadd`, `useradd`, and `passwd` commands as the `root` user.

Note: If you are installing Greenplum Database on RHEL 7.2 or CentOS 7.2 and chose to disable IPC object removal by creating the `gpadmin` user as a system account, provide the options `-r` (create the user as a system account) and `-m` (create the user home directory if it does not exist) to the

useradd command.

Note: Make sure the gpadmin user has the same user id (uid) and group id (gid) numbers on each host to prevent problems with scripts or services that use them for identity or permissions. For example, backing up Greenplum databases to some networked filesystems or storage appliances could fail if the gpadmin user has different uid or gid numbers on different segment hosts. When you create the gpadmin group and user, you can use the groupadd -g option to specify a gid number and the useradd -u option to specify the uid number. Use the command id gpadmin to see the uid and gid for the gpadmin user on the current host.

This example creates the gpadmin operating system group and creates the user account as a system account:

```
# groupadd gpadmin
# useradd gpadmin -r -m -g gpadmin
# passwd gpadmin
New password: <changeme>
Retype new password: <changeme>
```

Installing the Greenplum Database Software

Pivotal distributes the Greenplum Database software both as a downloadable RPM file and as a binary installer. You can use either distribution to install the software, but there are important differences between the two installation methods:

- If you use the RPM distribution, install the RPM file on the master, standby master, and every segment host. You will need to create the gpadmin user on every host. (See Creating the Greenplum Database Administrative User Account.) After the RPM file is installed on every host, you must enable passwordless SSH access for the gpadmin user from each host to every other host.
- If you use the binary installer, you can install the distribution on the master host only, and then use the Greenplum Database gpseginstall utility to copy the installation from the master host to all other hosts in the cluster. The gpseginstall utility creates the gpadmin user on each host, if it does not already exist, and enables passwordless SSH for the gpadmin user.

Warning: It is possible to install the RPM distribution on the master host, and then use the gpseginstall utility to copy the Greenplum Database installation directory to all other hosts. However, this is not recommended because gpseginstall does not install the RPM package on the other hosts, so you will be unable to use the OS package management utilities to remove or upgrade the Greenplum software on the standby master host or segment hosts.

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.

Parent topic: Configuring Your Systems and Installing Greenplum

Installing the RPM Distribution

Perform these steps on the master host, standby master host, and every segment host in the Greenplum Database cluster.

1. Log in as root.
2. Download or copy the RPM distribution file to the host machine. The RPM distribution filename has the format greenplum-db-<version>-<platform>.rpm where <platform> is similar to rhel7-x86_64 (Red Hat 64-bit) or sles11-x86_64 (SUSE Linux 64-bit).
3. Install the local RPM file:

```
# rpm -ivh ./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*
```

Enabling Passwordless SSH

After the RPM has been installed on all hosts in the cluster, use the `gpssh-exkeys` utility to set up passwordless SSH for the `gpadmin` user.

1. Log in to the master host as the `gpadmin` user.
2. Source the `path` file in the Greenplum Database installation directory.

```
$ source /usr/local/greenplum-db-<version>/greenplum_path.sh
```

3. In the `gpadmin` home directory, create a file named `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 segment hosts). Make sure there are no blank lines or extra spaces. Check the `/etc/hosts` file on your systems for the correct host names to use for your environment. For example, if you have a master, standby master, and three segment hosts 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
```

4. Run the `gpssh-exkeys` utility with your `hostfile_exkeys` file to enable passwordless SSH for the `gpadmin` user.

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

Note: You can run the `gpssh-exkeys` utility again as the `root` user if you want to enable passwordless SSH for `root`.

Follow the steps in Confirming Your Installation to verify that the Greenplum Database software is installed correctly.

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 similar to `rhel7-x86_64` (Red Hat 64-bit) or

`sles11-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 `gpinitsystem` 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 operating system user account

(typically named `gpadmin`), sets the account 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 user account).

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.
Note: The `gpseginstall` utility copies the installed files from the current host to the remote hosts. It does not use `rpm` to install Greenplum Database on the remote hosts, even if you used `rpm` to install Greenplum Database on the current host.

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. In the `gpadmin` user's home directory, 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 operating system user account `gpadmin` as a system account on all hosts and sets the account password to `changeme` for that user on all segment hosts.

   ```
   # gpseginstall -f hostfile_exkeys
   ```

   Use the `-u` and `-p` options to specify a different operating system account name and password. See `gpseginstall` for option information and information about 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.

Parent topic: Configuring Your Systems and Installing Greenplum

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`:

   ```
   $ su - gpadmin
   ```

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

   ```
   # 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:

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

   If the installation was successful, you can 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:

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

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**
- **Creating Data Storage Areas on Segment Hosts**

Parent topic: Configuring Your Systems and Installing Greenplum

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`:

```
# mkdir -p /data/master
```

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

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

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

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

Parent topic: Creating the Data Storage Areas

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`:

```
# 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:

```
# source /usr/local/greenplum-db/greenplum_path.sh
# gpssh -f hostfile_gpssh_segonly -e 'mkdir -p /data/primary'
# gpssh -f hostfile_gpssh_segonly -e 'mkdir -p /data/mirror'
# gpssh -f hostfile_gpssh_segonly -e 'chown -R gpadmin /data/*'
```

Parent topic: Creating the Data Storage Areas

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 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'
```

Parent topic: Configuring Your Systems and Installing Greenplum

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
```

Parent topic: Configuring Your Systems and Installing Greenplum

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 intialize a Greenplum Database system. See Initializing a Greenplum Database System for information about the *hostfile_gpinitsystem* file and the `gpinitsystem` command.

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 -j ACCEPT --syn -m state --state NEW
-A INPUT -i eth5 -p tcp -s 198.51.100.0/22 -j ACCEPT --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
# 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
:INPUT DROP
:FORWARD DROP
:OUTPUT ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A INPUT -i eth2 -p udp -s 192.0.2.0/22 -j ACCEPT
-A INPUT -i eth3 -p udp -s 198.51.100.0/22 -j ACCEPT
-A INPUT -i eth2 -p tcp -s 192.0.2.0/22 -j ACCEPT --syn -m state --state NEW
-A INPUT -i eth3 -p tcp -s 198.51.100.0/22 -j ACCEPT --syn -m state --state NEW
-A INPUT -i eth0 -p udp --dport snmp -s 203.0.113.0/21 -j ACCEPT
-A INPUT -i eth0 -p tcp --dport snmp -j ACCEPT --syn -m state --state NEW
-A INPUT -p tcp --dport ssh -j ACCEPT --syn -m state --state NEW
-A INPUT -i eth2 -p icmp -s 192.0.2.0/22 -j ACCEPT
-A INPUT -i eth3 -p icmp -s 198.51.100.0/22 -j ACCEPT
-A INPUT -i eth0 -p icmp --icmp-type echo-request -s 203.0.113.0/21 -j ACCEPT
-A INPUT -m limit --limit 5/min -j LOG --log-prefix "iptables denied: " --log-level 7
COMMIT
```

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

Parent topic: Configuring Your Systems and Installing Greenplum

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**
- **Validating Hardware Performance**
- **Validating Disk I/O and Memory Bandwidth**

Parent topic: Greenplum Database Installation Guide

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:

   ```
   $ 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:

   ```
   $ 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.

Parent topic: Validating Your Systems

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**

Parent topic: Validating Your Systems

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 1. Example Network Interface Configuration

Greenplum Host	Subnet1 NICs	Subnet2 NICs
Segment 1	sdw1-1	sdw1-2
Segment 2	sdw2-1	sdw2-2
Segment 3	sdw3-1	sdw3-2

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

Table 2. Example Network Test Host File Contents

hostfile_gpchecknet_ic1	hostfile_gpchecknet_ic2
sdw1-1	sdw1-2
sdw2-1	sdw2-2
sdw3-1	sdw3-2

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`).

Parent topic: Validating Hardware Performance

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
sdw2
sdw3
sdw4
```

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.

Parent topic: Validating Your Systems

Initializing a Greenplum Database System

Describes how to initialize a Greenplum Database database system.

The instructions in this chapter assume you have already prepared your hosts as described in Configuring Your Systems and Installing Greenplum.

This chapter contains the following topics:

Parent topic: Greenplum Database Installation Guide

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.
3. Create your Greenplum Database system configuration file. See Creating the Greenplum Database Configuration 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 Timezone and Localization Settings for more information.
5. Run the Greenplum Database initialization utility on the master host. See Running the Initialization Utility.
6. Set the Greenplum Database timezone. See Setting the Greenplum Database Timezone.
7. Set environment variables for the Greenplum Database user. See Setting Greenplum Environment Variables.

When performing the following initialization tasks, you must be logged into the master host as the `gpadmin` user, and to run Greenplum Database utilities, you must source the `greenplum_path.sh` file to set Greenplum Database environment variables. For example, if you are logged into the master, run these commands.

```
$ su - gpadmin
$ source /usr/local/greenplum-db/greenplum_path.sh
```

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. 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
```

2. 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. 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
```

2. 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.

   To specify `PORT_BASE`, review the port range specified in the `net.ipv4.ip_local_port_range` parameter in the `/etc/sysctl.conf` file. See Setting the Greenplum Recommended OS Parameters.

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

```
ARRAY_NAME="Greenplum Data Platform"
SEG_PREFIX=gpseg
PORT_BASE=6000
declare -a DATA_DIRECTORY=(/data1/primary /data1/primary /data1/primary /data2/
primary /data2/primary /data2/primary)
MASTER_HOSTNAME=mdw
MASTER_DIRECTORY=/data/master
MASTER_PORT=5432
TRUSTED SHELL=ssh
CHECK_POINT_SEGMENTS=8
ENCODING=UNICODE
```

3. (Optional) If you want to deploy mirror segments, uncomment and set the mirroring parameters according to your environment. To specify `MIRROR_PORT_BASE`, review the port range specified under the `net.ipv4.ip_local_port_range` parameter in the `/etc/sysctl.conf` file. Here is an example of the *optional* mirror parameters in the `gpinitsystem_config` file:

```
MIRROR_PORT_BASE=7000
declare -a MIRROR_DATA_DIRECTORY=(/data1/mirror /data1/mirror /data1/mirror /da
ta2/mirror /data2/mirror /data2/mirror)
```

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

4. 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.

These steps assume you are logged in as the `gpadmin` user and have sourced the `greenplum_path.sh` file to set Greenplum Database environment variables.

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:

   ```
   $ cd ~
   $ gpinitsystem -c gpconfigs/gpinitsystem_config -h gpconfigs/hostfile_gpinitsys
   tem
   ```

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

   ```
   $ gpinitsystem -c gpconfigs/gpinitsystem_config -h gpconfigs/hostfile_gpinitsys
   tem \
     -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:

   ```
   => 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:

   ```
   => 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 the Greenplum Database Timezone

As a best practice, configure Greenplum Database and the host systems to use a known, supported timezone. Greenplum Database uses a timezone from a set of internally stored PostgreSQL timezones. Setting the Greenplum Database timezone prevents Greenplum Database from selecting a timezone each time the cluster is restarted and sets the timezone for the Greenplum Database master and segment instances.

Use the `gpconfig` utility to show and set the Greenplum Database timezone. For example, these commands show the Greenplum Database timezone and set the timezone to `US/Pacific`.

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

You must restart Greenplum Database after changing the timezone. The command `gpstop -ra` restarts Greenplum Database. The catalog view `pg_timezone_names` provides Greenplum Database timezone information.

For more information about the Greenplum Database timezone, see Configuring Timezone and Localization Settings.

Setting Greenplum Environment Variables

You must set environment variables in the Greenplum Database user (`gpadmin`) environment that runs Greenplum Database on the Greenplum Database master and standby master hosts. A `greenplum_path.sh` file is provided in the Greenplum Database installation directory with environment variable settings for Greenplum Database.

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`.

These steps ensure that the environment variables are set for the `gpadmin` user after a system reboot.

To set up the gpadmin environment for Greenplum Database

1. Open the `gpadmin` profile file (such as `.bashrc`) in a text editor. For example:

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

2. 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
   ```

3. (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
```

4. (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
   ```

5. Save and close the file.
6. After editing the profile file, source it to make the changes active. For example:

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

7. 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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Installing Optional Extensions

Information about installing optional Greenplum Database extensions and packages, such as the Procedural Language extensions and the Python and R Data Science Packages.

- **Procedural Language, Machine Learning, and Geospatial Extensions**
- **Python Data Science Module Package**
- **R Data Science Library Package**
- **Greenplum Platform Extension Framework (PXF)**
- **Oracle Compatibility Functions**

- **dblink Connectivity Functions**
- **pgcrypto Cryptographic Functions**

Parent topic: Greenplum Database Installation Guide

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Procedural Language, Machine Learning, and Geospatial 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.x you must install and use a Greenplum Database extension package (gppkg files and contrib modules) that is built for Greenplum Database 5.x. Any custom modules that were used with earlier versions must be rebuilt for use with Greenplum Database 5.x.

Parent topic: Installing Optional Extensions

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Installing Optional Extensions

Python Data Science Modules

Modules provided in the Python Data Science package include:

Table 1. Data Science Modules

Module Name	Description/Used For
Beautiful Soup	Navigating HTML and XML

Table 1. Data Science Modules

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

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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Installing Optional Extensions

R Data Science Libraries

Libraries provided in the R Data Science package include:

abind	glmnet	quantreg
adabag	gplots	R2jags
arm	gtable	R6
assertthat	gtools	randomForest
BH	hms	RColorBrewer
bitops	hybridHclust	Rcpp
car	igraph	RcppEigen
caret	labeling	readr
caTools	lattice	reshape2
coda	lazyeval	rjags
colorspace	lme4	RobustRankAggreg
compHclust	lmtest	ROCR
curl	magrittr	rpart
data.table	MASS	RPostgreSQL
DBI	Matrix	sandwich
dichromat	MCMCpack	scales
digest	minqa	SparseM
dplyr	MTS	stringi
e1071	munsell	stringr
flashClust	neuralnet	survival
forecast	nloptr	tibble
foreign	nnet	tseries
gdata	pbkrtest	zoo
ggplot2	plyr	

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:

   ```
   $ 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:

   ```
   $ 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:

```
$ 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:

```
$ 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:

```
$ . /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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Greenplum Platform Extension Framework (PXF)

Optional. If you do not plan to use PXF, no action is necessary.

If you plan to use PXF, refer to Configuring PXF for instructions.

Parent topic: Installing Optional Extensions

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

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

To uninstall Oracle Compatibility Functions, use the script:

```
$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*.

Parent topic: Installing Optional Extensions

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

dblink Connectivity Functions

The PostgreSQL `dblink` module provides simple connections to other Greenplum Database databases from installations with the same major version number residing 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 `gpcopy`.

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.

Parent topic: Installing Optional Extensions

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

pgcrypto Cryptographic Functions

With the Greenplum Database `pgcrypto` extension, you can use the PostgreSQL module `pgcrypto` encryption/decryption functions. 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.

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

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.

For Greenplum Database 5.21.5 and earlier 5.x releases, you enable `pgcrypto` functions as a module using an SQL script. For Greenplum Database 5.22.0 and later 5.x releases, you enable `pgcrypto` functions as an extension.

Enable pgcrypto Extension

To enable the `pgcrypto` extension for Greenplum Database 5.22.0 and later, follow these steps.

1. If needed, uninstall the existing `pgcrypto` module with the SQL script `uninstall_pgcrypto.sql`.

   The `uninstall_pgcrypto.sql` script is in the `share/postgresql/contrib/` directory of the Greenplum Database 5.x installation that you used to install `pgcrypto`. This example `psql` command runs the SQL script to uninstall `pgcrypto` from the database `testdb`.

```
$ psql -d testdb -f <old-gp-install-dir>/share/postgresql/contrib/uninstall_pgc
rypto.sql
```

2. For each database that uses the `pgcrypto` functions, register the `pgcrypto` extension if necessary. This example `psql` command registers the `pgcrypto` extension in the database `testdb`.

```
$ psql -d testdb -c 'CREATE EXTENSION pgcrypto'
```

Disable pgcrypto Extension

When you remove `pgcrypto` extension support from a database, user-defined functions in the database that use `pgcrypto` functions will no longer work.

To disable the `pgcrypto` extension for Greenplum Database 5.22.0 and later, use the `DROP EXTENSION` command. This example `psql` command drops the `pgcrypto` extension in the database `testdb`.

```
$ psql -d testdb -c 'DROP EXTENSION pgcrypto'
```

Note: If you enabled the `pgcrypto.fips` server configuration parameter, you must disable the parameter.

Parent topic: Installing Optional Extensions

Configuring Timezone and Localization Settings

Describes the available timezone and localization features of Greenplum Database.

Parent topic: Greenplum Database Installation Guide

Configuring the Timezone

Greenplum Database selects a timezone to use from a set of internally stored PostgreSQL timezones. The available PostgreSQL timezones are taken from the Internet Assigned Numbers Authority (IANA) Time Zone Database, and Greenplum Database updates its list of available timezones as necessary when the IANA database changes for PostgreSQL.

Greenplum Database selects the timezone by matching a PostgreSQL timezone with the value of the `TimeZone` server configuration parameter, or the host system time zone if `TimeZone` is not set. For example, when selecting a default timezone from the host system time zone, Greenplum Database uses an algorithm to select a PostgreSQL timezone based on the host system timezone files. If the system timezone includes leap second information, Greenplum Database cannot match the system timezone with a PostgreSQL timezone. In this case, Greenplum Database calculates a "best match" with a PostgreSQL timezone based on information from the host system.

As a best practice, configure Greenplum Database and the host systems to use a known, supported timezone. This sets the timezone for the Greenplum Database master and segment instances, and prevents Greenplum Database from selecting a best match timezone each time the cluster is restarted, using the current system timezone and Greenplum Database timezone files (which may have been updated from the IANA database since the last restart). Use the `gpconfig` utility to show and set the Greenplum Database timezone. For example, these commands show the Greenplum Database timezone and set the timezone to `US/Pacific`.

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

You must restart Greenplum Database after changing the timezone. The command `gpstop -ra`

restarts Greenplum Database. The catalog view `pg_timezone_names` provides Greenplum Database timezone information.

About Locale Support in 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.

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 1. Greenplum Database Character Sets

Name	Description	Language	Server?	Bytes/Char	Aliases
BIG5	Big Five	Traditional Chinese	No	1-2	WIN950, Windows950
EUC_CN	Extended UNIX Code-CN	Simplified Chinese	Yes	1-3	
EUC_JP	Extended UNIX Code-JP	Japanese	Yes	1-3	
EUC_KR	Extended UNIX Code-KR	Korean	Yes	1-3	
EUC_TW	Extended UNIX Code-TW	Traditional Chinese, Taiwanese	Yes	1-3	
GB18030	National Standard	Chinese	No	1-2	
GBK	Extended National Standard	Simplified Chinese	No	1-2	WIN936, Windows936
ISO_8859_5	ISO 8859-5, ECMA 113	Latin/Cyrillic	Yes	1	
ISO_8859_6	ISO 8859-6, ECMA 114	Latin/Arabic	Yes	1	
ISO_8859_7	ISO 8859-7, ECMA 118	Latin/Greek	Yes	1	
ISO_8859_8	ISO 8859-8, ECMA 121	Latin/Hebrew	Yes	1	
JOHAB	JOHA	Korean (Hangul)	Yes	1-3	
KOI8	KOI8-R(U)	Cyrillic	Yes	1	KOI8R
LATIN1	ISO 8859-1, ECMA 94	Western European	Yes	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	Central European	Yes	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	South European	Yes	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	North European	Yes	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	Turkish	Yes	1	ISO88599
LATIN6	ISO 8859-10, ECMA 144	Nordic	Yes	1	ISO885910
LATIN7	ISO 8859-13	Baltic	Yes	1	ISO885913
LATIN8	ISO 8859-14	Celtic	Yes	1	ISO885914
LATIN9	ISO 8859-15	LATIN1 with Euro and accents	Yes	1	ISO885915
LATIN10	ISO 8859-16, ASRO SR 14111	Romanian	Yes	1	ISO885916
MULE_INTERNAL	Mule internal code	Multilingual Emacs	Yes	1-4	
SJIS	Shift JIS	Japanese	No	1-2	Mskanji, ShiftJIS, WIN932, Windows932
SQL_ASCII	unspecified[2]	any	No	1	

Table 1. Greenplum Database Character Sets

Name	Description	Language	Server?	Bytes/Char	Aliases
UHC	Unified Hangul Code	Korean	No	1-2	WIN949, Windows949
UTF8	Unicode, 8-bit	all	Yes	1-4	Unicode
WIN866	Windows CP866	Cyrillic	Yes	1	ALT
WIN874	Windows CP874	Thai	Yes	1	
WIN1250	Windows CP1250	Central European	Yes	1	
WIN1251	Windows CP1251	Cyrillic	Yes	1	WIN
WIN1252	Windows CP1252	Western European	Yes	1	
WIN1253	Windows CP1253	Greek	Yes	1	
WIN1254	Windows CP1254	Turkish	Yes	1	
WIN1255	Windows CP1255	Hebrew	Yes	1	
WIN1256	Windows CP1256	Arabic	Yes	1	
WIN1257	Windows CP1257	Baltic	Yes	1	
WIN1258	Windows CP1258	Vietnamese	Yes	1	ABC, TCVN, TCVN5712, VSCII

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 2. Client/Server Character Set Conversions

Server Character Set	Available Client Character Sets
BIG5	not supported as a server encoding

Table 2. Client/Server Character Set Conversions

Server Character Set	Available Client Character Sets
EUC_CN	EUC_CN, MULE_INTERNAL, UTF8
EUC_JP	EUC_JP, MULE_INTERNAL, SJIS, UTF8
EUC_KR	EUC_KR, MULE_INTERNAL, UTF8
EUC_TW	EUC_TW, BIG5, MULE_INTERNAL, UTF8
GB18030	not supported as a server encoding
GBK	not supported as a server encoding
ISO_8859_5	ISO_8859_5, KOI8, MULE_INTERNAL, UTF8, WIN866, WIN1251
ISO_8859_6	ISO_8859_6, UTF8
ISO_8859_7	ISO_8859_7, UTF8
ISO_8859_8	ISO_8859_8, UTF8
JOHAB	JOHAB, UTF8
KOI8	KOI8, ISO_8859_5, MULE_INTERNAL, UTF8, WIN866, WIN1251
LATIN1	LATIN1, MULE_INTERNAL, UTF8
LATIN2	LATIN2, MULE_INTERNAL, UTF8, WIN1250
LATIN3	LATIN3, MULE_INTERNAL, UTF8
LATIN4	LATIN4, MULE_INTERNAL, UTF8
LATIN5	LATIN5, UTF8
LATIN6	LATIN6, UTF8
LATIN7	LATIN7, UTF8
LATIN8	LATIN8, UTF8
LATIN9	LATIN9, UTF8
LATIN10	LATIN10, UTF8
MULE_INTERNAL	MULE_INTERNAL, BIG5, EUC_CN, EUC_JP, EUC_KR, EUC_TW, ISO_8859_5, KOI8, LATIN1 to LATIN4, SJIS, WIN866, WIN1250, WIN1251
SJIS	not supported as a server encoding
SQL_ASCII	not supported as a server encoding
UHC	not supported as a server encoding
UTF8	all supported encodings
WIN866	WIN866
ISO_8859_5	KOI8, MULE_INTERNAL, UTF8, WIN1251
WIN874	WIN874, UTF8
WIN1250	WIN1250, LATIN2, MULE_INTERNAL, UTF8
WIN1251	WIN1251, ISO_8859_5, KOI8, MULE_INTERNAL, UTF8, WIN866
WIN1252	WIN1252, UTF8
WIN1253	WIN1253, UTF8
WIN1254	WIN1254, UTF8
WIN1255	WIN1255, UTF8

Table 2. Client/Server Character Set Conversions

Server Character Set	Available Client Character Sets
WIN1256	WIN1256, UTF8
WIN1257	WIN1257, UTF8
WIN1258	WIN1258, UTF8

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:

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

  To query the current client encoding:

  ```
  => SHOW client_encoding;
  ```

  To return to the default encoding:

  ```
  => 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.

[1] Not all APIs support all the listed character sets. For example, the JDBC driver does not support MULE_INTERNAL, LATIN6, LATIN8, and LATIN10.

[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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

About Implicit Text Casting in Greenplum Database

Greenplum Database version 4.3.x is based on PostgreSQL version 8.2. Greenplum Database version 5.x is based on PostgreSQL version 8.3. PostgreSQL 8.3 removed automatic implicit casts between the `text` type and other data types. When you migrate from Greenplum Database version 4.3.x to version 5.x, this change in behavior might impact existing applications and queries.

For information about how Greenplum Database 5.x performs type casts, see Type Casts.

What is different in Greenplum Database 5.x?

Greenplum Database 5.x does not automatically implicitly cast between text and other data types. Greenplum Database 5.x also treats certain automatic implicit casts differently than version 4.3.x, and in some cases does not handle them at all. **Applications or queries that you wrote for Greenplum Database 4.3.x that rely on automatic implicit casting may fail on Greenplum Database version 5.x.**

(The term *implicit cast*, when used in the remainder of this section, refers to implicit casts automatically applied by Greenplum Database.)

- Greenplum Database 5.x has downgraded implicit casts in the to-text type direction; these casts are now treated as assignment casts. A cast from a data type to the text type will continue to work in Greenplum Database 5.x if used in assignment contexts.
- Greenplum Database 5.x no longer automatically provides an implicit cast in the to-text type direction that can be used in expression contexts. Additionally, Greenplum Database 5.x no longer provides implicit casts in the from-text type direction. When such expressions or assignments are encountered, Greenplum Database 5.x returns an error and the following message:

  ```
  HINT:  No operator matches the given name and argument type(s). You might need
  to add explicit type casts.
  ```

  To illustrate, suppose you create two tables:

  ```
  CREATE TABLE foo (a int) DISTRIBUTED RANDOMLY ;
  CREATE TABLE bar (b text) DISTRIBUTED RANDOMLY ;
  ```

  The following examples demonstrate certain types of text comparison queries that will fail on Greenplum Database 5.x.
  Note: This is not an exhaustive list of failure scenarios.
  - Queries that reference `text` type and non-text type columns in an expression. In this example query, the comparison expression returns a cast error.

    ```
    SELECT * FROM foo, bar WHERE foo.a = bar.b;
    ERROR:  operator does not exist: integer = text
    LINE 1: SELECT * FROM foo, bar WHERE foo.a = bar.b;
                                           ^
    HINT:  No operator matches the given name and argument type(s). You might
     need to add explicit type casts.
    ```

    The updated example casts the `text` type to an `integer` type.

    ```
    SELECT * FROM foo, bar WHERE foo.a = bar.b::int;
    ```

  - Queries that mix the `text` type and non-text type columns in function and aggregate arguments. In this example, the query that executes the example function `concat` returns a cast error.

    ```
    CREATE FUNCTION concat(TEXT, TEXT)
    RETURNS TEXT AS $$
      SELECT $1 || $2
    $$ STRICT LANGUAGE SQL;

    SELECT concat('a'::TEXT, 2);
    ```

    Adding an explicit cast from `integer` to `text` fixes the issue.

    ```
    SELECT concat('a', 2::text);
    ```

  - Queries that perform comparisons between a `text` type column and a non-quoted

literal such as an `integer`, `number`, `float`, or `oid`. This example query that compares text and non-quoted integer returns an error.

```
SELECT * FROM bar WHERE b = 123;
```

Adding an explicit cast to text fixes the issue.

```
SELECT * FROM bar WHERE b = 123::text;
```

- Queries that perform comparisons between a `date` type column or literal and an integer-like column (Greenplum Database internally converts date types to the text type) . This example query that compares an `integer` column with a literal of type `date` returns an error.

```
SELECT * FROM foo WHERE a = '20130101'::DATE;
```

There is no built-in cast from integer type to `date` type. However, you can explicitly cast an `integer` to `text` and then to `date`. The updated examples use the `cast` and `::` syntax.

```
SELECT * FROM foo WHERE cast(cast(a AS text) AS date)  = '20130101'::date
;
SELECT * FROM foo WHERE (a::text)::date  = '20130101'::date;
```

The only supported workaround for the implicit casting differences between Greenplum Database versions 4.3.x and 5.x is to analyze failing applications and queries and update the application or query to use explicit casts to fix the failures.

If rewriting the application or query is not feasible, you may choose to temporarily work around the change in behaviour introduced by the removal of automatic implicit casts in Greenplum Database 5.x. There are two well-known workarounds to this PostgreSQL issue:

- Re-create the implicit casts (described in Readding implicit casts in PostgreSQL 8.3).
- Create missing operators (described in Problems and workaround recreating implicit casts using 8.3+).

The workaround to re-create the implicit casts is not recommended as it breaks concatenation functionality. With the create missing operators workaround, you create the operators and functions that implement the comparison expressions that are failing in your applications and queries.

Parent topic: Greenplum Database Installation Guide

Workaround: Manually Creating Missing Operators

Warning: Use this workaround only to aid migration to Greenplum Database 5.x for evaluation purposes. Do not use this workaround in a production environment.

When you create an operator, you identify the data types of the left operand and the right operand. You also identify the name of a function that Greenplum Database invokes to evaluate the operator expression between the specified data types. The operator function evaluates the expression by performing either to-text or from-text conversion using the INPUT/OUTPUT methods of the data types involved. By creating operators for each (text type, other data type) and (other data type, text type) combination, you effectively implement the casts that are missing in Greenplum Database 5.x.

To implement this workaround, complete the following tasks **after** you install Greenplum Database 5.x:

1. Identify and note the names of the Greenplum 5.x databases in which you want to create the missing operators. Consider applying this workaround to all databases in your Greenplum Database deployment.
2. Identify a schema in which to create the operators and functions. Use a schema other than `pg_catalog` to ensure that these objects are included in a `pg_dump` or `gpbackup` of the

database. This procedure will use a schema named `cast_fix` for illustrative purposes.

3. Review the blog entry Problems and workaround recreating implicit casts using 8.3+. The blog discusses this temporary workaround to the casting issue, i.e. creating missing operators. It also references a SQL script that you can run to create a set of equality (=) operators and functions for several text and other data type comparisons.

4. Download the `8.3 operator workaround.sql` script referenced on the blog page, noting the location to which the file was downloaded on your local system.

5. The `8.3 operator workaround.sql` script creates the equality operators and functions. Open the script in the editor of your choice, and examine the contents. For example, using the `vi` editor:

   ```
   vi 8.3 operator workaround.sql
   ```

   Notice that the script creates the operators and functions in the `pg_catalog` schema.

6. Replace occurrences of `pg_catalog` in the script with the name of the schema that you identified in Step 2, and then save the file and exit the editor. (You will create this schema in an upcoming step if the schema does not already exist.) For example:

   ```
   :s/pg_catalog/cast_fix/g
   :wq
   ```

7. Analyze your failing queries, identifying the operators and from-type and to-type data type comparisons that are the source of the failures. Compare this list to the contents of the `8.3 operator workaround.sql` script, and identify the minimum set of additional operators and left_type/right_type expression combinations that you must support.

8. For each operator and left_type/right_type combination that you identify in the previous step, add `CREATE` statements for the following *objects* to the `8.3 operator workaround.sql` script:

   a. *Create the function that implements the left_type operator right_type comparison.* For example, to create a function that implements the greater than (>) operator for text (left_type) to integer (right_type) comparison:

      ```
      CREATE FUNCTION cast_fix.textgtint(text, integer)
      RETURNS boolean
      STRICT IMMUTABLE LANGUAGE SQL AS $$
        SELECT textin(int4out($2)) > $1;
      $$;
      ```

      Be sure to schema-qualify the function name.

   b. *Create the operator*. For example, to create a greater than (>) operator for text (left_type) to integer (right_type) type comparison that specifies the function you created above:

      ```
      CREATE OPERATOR cast_fix.> (PROCEDURE=cast_fix.textgtint, LEFTARG=text, R
      IGHTARG=integer, COMMUTATOR=OPERATOR(cast_fix.>))
      ```

      Be sure to schema-qualify the operator and function names.

   c. You must create another operator and function if you want the operator to work in reverse (i.e. using the example above, if you want a greater than operator for integer (left_type) to text (right_type) comparison.)

9. For each database that you identified in Step 1, add the missing operators. For example:

   a. Connect to the database as an administrative user. For example:

      ```
      $ psql -d database1 -U gpadmin
      ```

b. Create the schema if it does not already exist. For example:

```
CREATE SCHEMA cast_fix;
```

c. Run the script. For example, if you downloaded the file to the `/tmp` directory:

```
\i '/tmp/8.3 operator workaround.sql'
```

You must create the schema and run the script for every new database that you create in your Greenplum Database cluster.

10. Identify and note the names of the users/roles to which you want to provide this capability. Consider exposing this to all roles in your Greenplum Database deployment.

11. For each role that you identified in Step 10, add the schema to the role's `search_path`. For example:

```
SHOW search_path;
ALTER ROLE bill SET search_path TO existing_search_path, cast_fix;
```

If required, also grant schema-level permissions to the role.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`.

- `gpactivatestandby`
- `gpaddmirrors`
- `gpcheck` (deprecated)
- `gpcheckperf`
- `gpcopy`
- `gpdeletesystem`
- `gpinitstandby`
- `gpinitsystem`
- `gppkg`
- `gpscp`
- `gpseginstall`
- `gpssh`
- `gpssh-exkeys`
- `gpstart`
- `gpstop`
- `gptransfer` (deprecated)

Parent topic: Greenplum Database Installation Guide

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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**
- **Optional Environment Variables**

Parent topic: Greenplum Database Installation Guide

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

Parent topic: Greenplum Environment Variables

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.

Parent topic: Greenplum Environment Variables

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...`)

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.
- **Greenplum Database Ports and Protocols**
  Lists network ports and protocols used within the Greenplum cluster.
- **Configuring Client Authentication**
  Describes the available methods for authenticating Greenplum Database clients.
- **Configuring Database Authorization**
  Describes how to restrict authorization access to database data at the user level by using roles and permissions.
- **Greenplum Command Center Security**
- **Auditing**
  Describes Greenplum Database events that are logged and should be monitored to detect security threats.
- **Encrypting Data and Database Connections**
  Describes how to encrypt data at rest in the database or in transit over the network, to protect from evesdroppers or man-in-the-middle attacks.
- **Enabling gphdfs Authentication with a Kerberos-secured Hadoop Cluster (Deprecated)**
  Provides steps for configuring Greenplum Database to access external tables in a Hadoop cluster secured with Kerberos.
- **Security Best Practices**
  Describes basic security best practices that you should follow to ensure the highest level of system security.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 Configuring PXF for Secure HDFS for the configuration procedure for PXF. See Enabling gphdfs Authentication with a Kerberos-secured Hadoop Cluster (Deprecated) for the steps to perform this setup for `gphdfs` (deprecated).

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.

Parent topic: Greenplum Database Security Configuration Guide

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:

```
[gpadmin@mdw ~]$ gpstate -p
20190403:02:57:04:011030 gpstate:mdw:gpadmin-[INFO]:-Starting gpstate with args: -p
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-local Greenplum Version: 'postgre
s (Greenplum Database) 5.17.0 build commit:fc9a9d4cad8dd4037b9bc07bf837c0b958726103'
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-master Greenplum Version: 'Postgr
eSQL 8.3.23 (Greenplum Database 5.17.0 build commit:fc9a9d4cad8dd4037b9bc07bf837c0b958
726103) on x86_64-pc-linux-gnu, compiled by GCC gcc (GCC) 6.2.0, 64-bit compiled on Fe
b 13 2019 15:26:34'
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-Obtaining Segment details from ma
ster...
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:--Master segment instance  /data/m
aster/gpseg-1  port = 5432
```

```
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:--Segment instance port assignment
s
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-----------------------------------
-
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   Host   Datadir
Port
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw1   /data/primary/gpseg0
20000
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw2   /data/mirror/gpseg0
21000
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw1   /data/primary/gpseg1
20001
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw2   /data/mirror/gpseg1
21001
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw1   /data/primary/gpseg2
20002
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw2   /data/mirror/gpseg2
21002
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw2   /data/primary/gpseg3
20000
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw3   /data/mirror/gpseg3
21000
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw2   /data/primary/gpseg4
20001
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw3   /data/mirror/gpseg4
21001
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw2   /data/primary/gpseg5
20002
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw3   /data/mirror/gpseg5
21002
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw3   /data/primary/gpseg6
20000
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw1   /data/mirror/gpseg6
21000
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw3   /data/primary/gpseg7
20001
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw1   /data/mirror/gpseg7
21001
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw3   /data/primary/gpseg8
20002
20190403:02:57:05:011030 gpstate:mdw:gpadmin-[INFO]:-   sdw1   /data/mirror/gpseg8
21002
```

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.

Note: 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 values outside of that range:

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

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 1. Greenplum Database Ports and Protocols

Service	Protocol/Port	Description

Master SQL client connection	TCP 5432, libpq	SQL client connection port on the Greenplum master host. Supports clients using the PostgreSQL libpq API. Configurable.
Segment SQL client connection	varies, libpq	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 primary (p) or mirror (m) segment in the `port` column. Run `gpstate -p` to view the ports in use.
Segment mirroring port	varies, libpq	The port where a segment receives mirrored blocks from its primary. The port is assigned when the mirror is set up. The `gp_segment_configuration` system catalog records port numbers for each primary (p) or mirror (m) segment in the `port` column. Run `gpstate -p` to view the ports in use.
Greenplum Database Interconnect	UDP 1025-65535, dynamically allocated	The Interconnect transports database tuples between Greenplum segments during query execution.
Standby master client listener	TCP 5432, libpq	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.
Standby master replicator	TCP 1025-65535, gpsyncmaster	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.
Greenplum database file load and transfer utilities: gpfdist, gpload, gptransfer	TCP 8080, HTTP TCP 9000, HTTPS	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. Note: The gptransfer utility is deprecated and will be removed in the next major release of Greenplum Database.
Gpperfmon agents	TCP 8888	Connection port for gpperfmon agents (`gpmmon` and `gpsmon`) executing on Greenplum Database hosts. Configure by setting the `gpperfmon_port` configuration variable in postgresql.conf on master and segment hosts.
Backup completion notification	TCP 25, TCP 587, SMTP	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.
Greenplum Database secure shell (SSH): gpssh, gpscp, gpssh-exkeys, gppkg, gpseginstall	TCP 22, SSH	Many Greenplum utilities use scp and ssh to transfer files between hosts and manage the Greenplum system within the cluster.
gphdfs (deprecated)	TCP 8020	The gphdfs protocol allows access to data in a Hadoop file system via Greenplum external tables. The URL in the `LOCATION` clause of the `CREATE EXTERNAL TABLE` command specifies the host address and port number for the Hadoop namenode service.
Greenplum Platform Extension Framework (PXF)	TCP 5888	The PXF Java service runs on port number 5888 on each Greenplum Database segment host.
Greenplum Command Center (GPCC)	TCP 28080, HTTP/HTTPS, WebSocket (WS), Secure WebSocket (WSS)	The GPCC web server (`gpccws` process) executes on the Greenplum Database master host or standby master host. The port number is configured at installation time.

Table 1. Greenplum Database Ports and Protocols

Service	Protocol/Port	Description
	TCP 8899, rcp port	A GPCC agent (`ccagent` process) on each Greenplum Database segment host connects to the GPCC rpc backend at port number 8899 on the GPCC web server host.
	UNIX domain socket, agent	Greenplum Database processes transmit datagrams to the GPCC agent (`ccagent` process) on each segment host using a UNIX domain socket.
GPText	TCP 2188 (base port)	ZooKeeper client ports. ZooKeeper uses a range of ports beginning at the base port number. The base port number and maximum port number are set in the GPText installation configuration file at installation time. The default base port number is 2188.
	TCP 18983 (base port)	GPText (Apache Solr) nodes. GPText nodes use a range of ports beginning at the base port number. The base port number and maximum port number are set in the GPText installation configuration file at installation time. The default base port number is 18983.
EMC Data Domain and DD Boost	TCP/UDP 111, NFS portmapper	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.
	TCP 2052	Main port used by NFS mountd. This port can be set on the Data Domain system using the `nfs set mountd-port` command .
	TCP 2049, NFS	Main port used by NFS. This port can be configured using the `nfs set server-port` command on the Data Domain server.
	TCP 2051, replication	Used when replication is configured on the Data Domain system. This port can be configured using the `replication modify` command on the Data Domain server.
Symantec NetBackup	TCP/UDB 1556, veritas-pbx	The Symantec NetBackup client network port.
	TCP 13724, vnetd	Symantec NetBackup vnetd communication port.
Pgbouncer connection pooler	TCP, libpq	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 pgbouncer.ini configuration file.

Parent topic: Greenplum Database Security Configuration Guide

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

Parent topic: Greenplum Database Security Configuration Guide

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:

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

A UNIX-domain socket access record is in this format:

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

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 a `@`.

address

Specifies the client machine addresses that this record matches. This field can contain an IP address, an IP address range, or a host name.

An IP address range is specified using standard numeric notation for the range's starting address, then a slash (`/`) and a CIDR mask length. The mask length indicates the number of high-order bits of the client IP address that must match. Bits to the right of this should 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 an IPv4 address range specified this way 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. An IPv6 address range might look like `::1/128` for a single host (in this case the IPv6 loopback address) or `fe80::7a31:c1ff:0000:0000/96` for a small network. `0.0.0.0/0` represents all IPv4 addresses, and `::0/0` represents all IPv6 addresses. To specify a single host, use a mask length of 32 for IPv4 or 128 for IPv6. In a network address, do not omit trailing zeroes.

An entry given in IPv4 format will match only IPv4 connections, and an entry given in IPv6 format will match only IPv6 connections, even if the represented address is in the IPv4-in-IPv6 range.

Note: Entries in IPv6 format will be rejected if the host system C library does not have support for IPv6 addresses.

If a host name is specified (an address that is not an IP address or IP range is treated as a host name), that name is compared with the result of a reverse name resolution of the client IP address (for example, reverse DNS lookup, if DNS is used). Host name comparisons are case insensitive. If there is a match, then a forward name resolution (for example, forward DNS lookup) is performed on the host name to check whether any of the addresses it resolves to are equal to the client IP address. If both directions match, then the entry is considered to match.

Some host name databases allow associating an IP address with multiple host names, but the operating system only returns one host name when asked to resolve an IP address. The host name that is used in `pg_hba.conf` must be the one that the address-to-name resolution of the client IP address returns, otherwise the line will not be considered a match.

When host names are specified in `pg_hba.conf`, you should ensure that name resolution is reasonably fast. It can be of advantage to set up a local name resolution cache such as `nscd`. Also, you can enable the server configuration parameter `log_hostname` to see the client host name instead of the IP address in the log.

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.

authentication-method

Specifies the authentication method to use when connecting. See Authentication Methods for options.

CAUTION:

For a more secure system, consider removing records for remote connections that use trust authentication from the `pg_hba.conf` file. Trust authentication grants any user who can connect to the server access to the database using any role they specify. You can safely replace trust authentication with ident authentication for local UNIX-socket connections. You can also use ident authentication for local and remote TCP clients, but the client host must be running an ident service and you must trust the integrity of that machine.

Editing the pg_hba.conf File

Initially, the `pg_hba.conf` file is set up with generous permissions for the gpadmin user and no database access for other Greenplum Database roles. You will need to edit the `pg_hba.conf` file to enable users' access to databases and to secure the gpadmin user. Consider removing entries that have trust authentication, since they allow anyone with access to the server to connect with any role they choose. For local (UNIX socket) connections, use ident authentication, which requires the operating system user to match the role specified. For local and remote TCP connections, ident authentication requires the client's host to run an indent service. You could install an ident service on the master host and then use ident authentication for local TCP connections, for example `127.0.0.1/28`. Using ident authentication for remote TCP connections is less secure because it requires you to trust the integrity of the ident service on the client's host.

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.

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:

```
# 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. This is secure for local socket connections. Using ident for TCP connections from remote hosts requires that the client's host is running an ident service. The ident authentication method should only be used with remote hosts on a trusted, closed network.

Following are some sample `pg_hba.conf` basic authentication entries:

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

SSL 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=*mapping*
  : mapping.
  This is specified in the `pg_ident.conf`, or in the file specified in the `ident_file` server setting.

  Following are sample `pg_hba.conf` entries for SSL client authentication:

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

OpenSSL Configuration

Greenplum Database reads the OpenSSL configuration file specified in `$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.

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 `openssl` command:

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

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:

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

4. Enter the old passphrase to unlock the existing key. Then run the following command:

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

   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:

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

For more details on how to create your server private key and certificate, refer to the OpenSSL documentation.

Configuring postgresql.conf for SSL Authentication

The following Server settings need to be specified in the `postgresql.conf` configuration file:

- `ssl` *boolean*. Enables SSL connections.
- `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

The "PAM" (Pluggable Authentication Modules) authentication method validates username/password pairs, similar to basic authentication. To use PAM authentication, the user must already exist as a Greenplum Database role name.

Greenplum uses the `pamservice` authentication parameter to identify the service from which to

obtain the PAM configuration.

Note: If PAM is set up to read `/etc/shadow`, authentication will fail because the PostgreSQL server is started by a non-root user. This is not an issue when PAM is configured to use LDAP or another authentication method.

Greenplum Database does not install a PAM configuration file. If you choose to use PAM authentication with Greenplum, you must identify the PAM service name for Greenplum and create the associated PAM service configuration file and configure Greenplum Database to use PAM authentication as described below:

1. Log in to the Greenplum Database master host and set up your environment. For example:

   ```
   $ ssh gpadmin@<gpmaster>
   gpadmin@gpmaster$ . /usr/local/greenplum-db/greenplum_path.sh
   ```

2. Identify the `pamservice` name for Greenplum Database. In this procedure, we choose the name `greenplum`.

3. Create the PAM service configuration file, `/etc/pam.d/greenplum`, and add the text below. You must have operating system superuser privileges to create the `/etc/pam.d` directory (if necessary) and the `greenplum` PAM configuration file.

   ```
   #%PAM-1.0
   auth    include    password-auth
   account   include    password-auth
   ```

   This configuration instructs PAM to authenticate the local operating system user.

4. Ensure that the `/etc/pam.d/greenplum` file is readable by all users:

   ```
   sudo chmod 644 /etc/pam.d/greenplum
   ```

5. Add one or more entries to the `pg_hba.conf` configuration file to enable PAM authentication in Greenplum Database. These entries must specify the `pam` *auth-method*. You must also specify the `pamservice=greenplum` *auth-option*. For example:

   ```
   host     <user-name>     <db-name>     <address>     pam     pamservice=greenpl
   um
   ```

6. Reload the Greenplum Database configuration:

   ```
   $ gpstop -u
   ```

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:

```
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):

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

In `SEGMENT_DATA_DIRECTORY/postgresql.conf` for all segment instances:

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

Note: 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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Configuring Database Authorization

Describes how to restrict authorization access to database data at the user level by using roles and permissions.

Parent topic: Greenplum Database Security Configuration Guide

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:

Object Type	Privileges
Tables, Views, Sequences	• SELECT • INSERT • UPDATE • DELETE • RULE • ALL
External Tables	• SELECT • RULE • ALL
Databases	• CONNECT • CREATE • TEMPORARY | TEMP • ALL
Functions	EXECUTE
Procedural Languages	USAGE
Schemas	• CREATE • USAGE • ALL

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:

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

To revoke privileges, use the `REVOKE` command. For example:

```
=# 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:

```
=# 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, `testdb`.

   ```
   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 `pg_authid.rolpasswod`.

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>
   ```

Restricting Access by Time

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.

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 `SET ROLE` and `SET SESSION AUTHORIZATION` commands are not affected by any time-based constraints.

Superuser or `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 `DENY` in the `CREATE ROLE` or `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 `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.

English Term	Number
DAY 'Sunday'	DAY 0
DAY 'Monday'	DAY 1

English Term	Number
DAY 'Tuesday'	DAY 2
DAY 'Wednesday'	DAY 3
DAY 'Thursday'	DAY 4
DAY 'Friday'	DAY 5
DAY 'Saturday'	DAY 6

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'
        DAY 2 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 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'
  ...
```

Excample 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 contraint 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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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:

```
local    gpperfmon   gpmon         md5
```

```
host     all         gpmon         127.0.0.1/28    md5
host     all         gpmon         ::1/128         md5
```

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`:

```
local   gpperfmon   gpmon                           md5
host    gpperfmon   gpmon    127.0.0.1/28           md5
host    gpperfmon   gpmon    ::1/128                md5
```

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

Parent topic: Greenplum Database Security Configuration Guide

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 enviroment for system status information. For more information, including usage, refer to *The gp_tookit 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`).

#	Field Name	Data Type	Description
1	event_time	timestamp with time zone	Time that the log entry was written to the log
2	user_name	varchar(100)	The database user name
3	database_name	varchar(100)	The database name
4	process_id	varchar(10)	The system process id (prefixed with "p")
5	thread_id	varchar(50)	The thread count (prefixed with "th")
6	remote_host	varchar(100)	On the master, the hostname/address of the client machine. On the segment, the hostname/address of the master.
7	remote_port	varchar(10)	The segment or master port number
8	session_start_time	timestamp with time zone	Time session connection was opened
9	transaction_id	int	Top-level transaction ID on the master. This ID is the parent of any subtransactions.
10	gp_session_id	text	Session identifier number (prefixed with "con")

#	Field Name	Data Type	Description
11	gp_command_count	text	The command number within a session (prefixed with "cmd")
12	gp_segment	text	The segment content identifier (prefixed with "seg" for primaries or "mir" for mirrors). The master always has a content id of -1.
13	slice_id	text	The slice id (portion of the query plan being executed)
14	distr_tranx_id	text	Distributed transaction ID
15	local_tranx_id	text	Local transaction ID
16	sub_tranx_id	text	Subtransaction ID
17	event_severity	varchar(10)	Values include: LOG, ERROR, FATAL, PANIC, DEBUG1, DEBUG2
18	sql_state_code	varchar(10)	SQL state code associated with the log message
19	event_message	text	Log or error message text
20	event_detail	text	Detail message text associated with an error or warning message
21	event_hint	text	Hint message text associated with an error or warning message
22	internal_query	text	The internally-generated query text
23	internal_query_pos	int	The cursor index into the internally-generated query text
24	event_context	text	The context in which this message gets generated
25	debug_query_string	text	User-supplied query string with full detail for debugging. This string can be modified for internal use.
26	error_cursor_pos	int	The cursor index into the query string
27	func_name	text	The function in which this message is generated
28	file_name	text	The internal code file where the message originated

#	Field Name	Data Type	Description
29	file_line	int	The line of the code file where the message originated
30	stack_trace	text	Stack trace text associated with this message

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:

Field Name	Value Range	Default	Description
log_connections	Boolean	off	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	Boolean	off	This outputs a line in the server log at termination of a client session, and includes the duration of the session.
log_statement	NONE DDL MOD ALL	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.

Field Name	Value Range	Default	Description
log_duration	Boolean	off	Causes the duration of every completed statement which satisfies log_statement to be logged.
log_error_verbosity	TERSE DEFAULT VERBOSE	DEFAULT	Controls the amount of detail written in the server log for each message that is logged.
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.
log_min_messages	DEBUG5 DEBUG4 DEBUG3 DEBUG2 DEBUG1 INFO NOTICE WARNING ERROR LOG FATAL PANIC	NOTICE	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.
log_rotation_age	Any valid time expression (number and unit)	1d	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.
log_statement_stats	Boolean	off	For each query, write total performance statistics of the query parser, planner, and executor to the server log. This is a crude profiling instrument.

Field Name	Value Range	Default	Description
log_truncate_on_rotation	Boolean	off	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.

Parent topic: Greenplum Database Security Configuration Guide

Encrypting Data and Database Connections

Describes how to encrypt data at rest in the database or in transit over the network, to protect from evesdroppers 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` module 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.

Parent topic: Greenplum Database Security Configuration Guide

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:

```
=# 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 module 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 is installed by default when you install Greenplum Database. You must explicitly enable pgcrypto in each database in which you want to use the module.

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 then 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 1. Pgcrypto Supported Encryption Functions

Value Functionality	Built-in	With OpenSSL
MD5	yes	yes
SHA1	yes	yes
SHA224/256/384/512	yes	yes [1]
Other digest algorithms	no	yes [2]
Blowfish	yes	yes
AES	yes	yes[3]
DES/3DES/CAST5	no	yes
Raw Encryption	yes	yes
PGP Symmetric-Key	yes	yes
PGP Public Key	yes	yes

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/. 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
```

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 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>
ssb   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 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)

mQENBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jtWiXZTh
2His1ojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgKlGSgd9texg2nnSL9Admqik/yX
R5syVKG+qcdWuvyZg9oOOmeyjhc3n+kkbRTEMuM3flbMs8shOwzMvstCUVmuHU/V
vG5rJAe8PuYDSJCJ74I6w7SOH3RiRIc7IfL6xYddV42l3ctd44bl8/i71hq2UyN2
/Hbsjii2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFnFnE6Pwbhn+
ic8MD0lK5/GAlR6Hc0ZIHf8KEcavruQlikjnABEBAAG0HHRlc3Qga2V5IDx0ZXN0
a2V5QGVtYWlsLmNvbT6JAT4EEwECACgFAlS1Zf0CGwMFCQHhM4AGCwkIBwMCBhUI
AgkKCwQWAgMBAh4BAheAAAoJEAd9cl4gJ8wwbfwH/3VyVsPkQl1owRJNxvXGt1bY
7BfrvU52yk+PPZYoes9UpdL3CMRk8gAM9bx5Sk08q2UXSZLC6fFOpEW4uWgmGYf8
JRoC3ooezTkmCBW8I1bU0qGetzVxopdXLuPGCE7hVWQe9HcSntiTLxGov1mJAwO7
TAoccXLbyuZh9Rf5vLoQdKzcCyOHh5IqXaQOT100TeFeEpb9TIiwcntg3WCSU5P0
DGoUAOanjDZ3KE8Qp7V74fhG1EZVzHb8FajR62CXSHFKqpBgiNxnTOk45NbXADn4
eTUXPSnwPi46qoAp9UQogsfGyB1XDOTB2UOqhutAMECaM7VtpePv79i0Z/NfnBe5
AQ0EVLVl/QEIANabFdQ+8QMCADOipM1bF/JrQt3zUoc4BTqICaxdyzAfz0tUSf/7
Zro2us99GlARqLWd8EqJcl/xmfcJiZyUam6ZAzzFXCgnH5Y1sdtMTJZdLp5WeOjw
gCWG/ZLu4wzxOFFzDkiPv9RDw6e5MNLtJrSp4hS5o2apKdbO4Ex83O4mJYnav/rE
iDDCWU4T0lhv3hSKCpke6LcwsX+7liozp+aNmP0Ypwfi4hR3UUMP70+V1beFqW2J
bVLz3lLLouHRgpCzla+PzzbEKs16jq77vG9kqZTCIzXoWaLljuitRlfJkO3vQ9hO
v/8yAnkcAmowZrIBlyFg2KBzhunYmN2YvkUAEQEAAYkBJQQYAQIADwUCVLVl/QIb
DAUJAeEzgAAKCRAHfXJeICfMMOHYCACFhInZA9uAM3TC44l+MrgMUJ3rW9izrO48
WrdTsxR8WkSNbIxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRrI61FPHQNPSvz62
WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8Hkk8qb5x/
HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNlGWE8pvgEx
/UUZB+dYqCwtvX0nnBu1KNCmk2AkEcFK3YoliCxomdOxhFOv9AKjjojDyC65KJci
Pv2MikPS2fKOAg1R3LpMa8zDEtl4w3vckPQNrQNnYuUtfj6ZoCxv
=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(".

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

INSERT INTO userssn(username, ssn)
SELECT robotccs.username, pgp_pub_encrypt(robotccs.ssn, keys.pubkey) AS ssn
FROM (
        VALUES ('Alice', '123-45-6788'), ('Bob', '123-45-6799'))
            AS robotccs(username, ssn)
CROSS JOIN  (SELECT  dearmor('-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.14 (GNU/Linux)

mQENBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jtWiXZTh
2His1ojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgKlGSgd9texg2nnSL9Admqik/yX
R5syVKG+qcdWuvyZg9oOOmeyjhc3n+kkbRTEMuM3flbMs8shOwzMvstCUVmuHU/V
```

```
vG5rJAe8PuYDSJCJ74I6w7SOH3RiRIc7IfL6xYddV42l3ctd44bl8/i71hq2UyN2
/Hbsjii2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFnFnE6Pwbhn+
ic8MD0lK5/GAlR6Hc0ZIHf8KEcavruQlikjnABEBAAG0HHRlc3Qga2V5IDx0ZXN0
a2V5QGVtYWlsLmNvbT6JAT4EEwECACgFAlS1Zf0CGwMFCQHhM4AGCwkIBwMCBhUI
AgkKCwQWAgMBAh4BAheAAAoJEAd9cl4gJ8wwbfwH/3VyVsPkQl1owRJNxvXGt1bY
7BfrvU52yk+PPZYoes9UpdL3CMRk8gAM9bx5Sk08q2UXSZLC6fFOpEW4uWgmGYf8
JRoC3ooezTkmCBW8I1bU0qGetzVxopdXLuPGCE7hVWQe9HcSntiTLxGov1mJAwO7
TAoccXLbyuZh9Rf5vLoQdKzcCyOHh5IqXaQOT100TeFeEpb9TIiwcntg3WCSU5P0
DGoUAOanjDZ3KE8Qp7V74fhG1EZVzHb8FajR62CXSHFKqpBgiNxnTOk45NbXADn4
eTUXPSnwPi46qoAp9UQogsfGyB1XDOTB2UOqhutAMECaM7VtpePv79i0Z/NfnBe5
AQ0EVLVl/QEIANabFdQ+8QMCADOipM1bF/JrQt3zUoc4BTqICaxdyzAfz0tUSf/7
Zro2us99GlARqLWd8EqJcl/xmfcJiZyUam6ZAzzFXCgnH5Y1sdtMTJZdLp5WeOjw
gCWG/ZLu4wzxOFFzDkiPv9RDw6e5MNLtJrSp4hS5o2apKdbO4Ex83O4mJYnav/rE
iDDCWU4T0lhv3hSKCpke6LcwsX+7liozp+aNmP0Ypwfi4hR3UUMP70+V1beFqW2J
bVLz3lLLouHRgpCzla+PzzbEKs16jq77vG9kqZTCIzXoWaLljuitRlfJkO3vQ9hO
v/8yAnkcAmowZrIBlyFg2KBzhunYmN2YvkUAEQEAAYkBJQQYAQIADwUCVLVl/QIb
DAUJAeEzgAAKCRAHfXJeICfMMOHYCACFhInZA9uAM3TC44l+MrgMUJ3rW9izrO48
WrdTsxR8WkSNbIxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRrI61FPHQNPSvz62
WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8Hkk8qb5x/
HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNlGWE8pvgEx
/UUZB+dYqCwtvX0nnBu1KNCmk2AkEcFK3YoliCxomdOxhFOv9AKjjojDyC65KJci
Pv2MikPS2fKOAg1R3LpMa8zDEtl4w3vckPQNrQNnYuUtfj6ZoCxv
=XZ8J
-----END PGP PUBLIC KEY BLOCK-----' AS pubkey) AS keys;
```

3. Verify that the `ssn` column is encrypted.

```
test_db=# select * from userssn;
ssn_id   | 1
username | Alice
ssn      | \301\300L\003\235M%_O\322\357\273\001\010\000\272\227\010\341\216\36
0\217C\020\261)_\367
[\227\034\313:C\354d<\337\006Q\351('\2330\031lX\263Qf\341\262\200\3015\235\036A
K\242fL+\315g\322
7u\270*\304\361\355\220\021\330"\200%\264\274}R\213\377\363\235\366\030\023)\36
4!\331\303\237t\277=
f \015\004\242\231\263\225%\032\271a\001\035\277\021\375X\232\304\305/\340\334\
0131\325\344[~\362\0
37-\251\336\303\340\377_\011\275\301/MY\334\343\245\244\372y\257S\374\230\346\2
77\373W\346\230\276\
017fi\226Q\307\012\326\3646\000\326\005:E\364W\252=zz\010(:\343Y\237\257iqU\032
6\350=v0\362\327\350\
315G^\027:K_9\254\362\354\215<\001\304\357\331\355\323,\302\213Fe\265\315\232\3
67\254\245%(\\\373
4\254\230\331\356\006B\257\333\326H\022\013\353\216F?\023\220\370\035vH5/\227\3
44b\322\227\026\362=\
42\033\322<\001}\243\224;)\030zqX\214\340\221\035\275U\345\327\214\032\351\223c
\2442\345\304K\016\
011\214\307\227\237\270\026'R\205\205a~1\263\236[\037C\260\031\205\374\245\317\
033k|\366\253\037
---------+--------------------------------------------------------------------
-----------------------
---------------------------------------------------------------------------
-----------------------
---------------------------------------------------------------------------
-----------------------
---------------------------------------------------------------------------
-----------------------
---------------------------------------------------------------------------
-----------------------
---------------------------------------------------------------------------
-----------------------
---------------------------------------------------------------------------
-----------------------
---------------------------------------------------------------------------
-----------------------
---------------------------------------------------------------------------
-----------------------
---------------------------------------------------------------------------
-----------------------
---------------------------------------------------------------------------
-----------------------
---------------------------------------------------------------------------
ssn_id   | 2
```

```
username | Bob
ssn      | \301\300L\003\235M%_O\322\357\273\001\007\377t>\345\343,\200\256\272
\300\012\033M4\265\032L
L[v\262k\244\2435\264\232B\357\370d9\375\011\002\327\235<\246\210b\030\012\337@
\226Z\361\246\032\00
7'\012c\353]\355d7\360T\335\314\367\370;X\371\350*\231\212\260B\010#RQ0\223\253
c7\0132b\355\242\233\34
1\000\370\370\366\013\022\357\005i\202~\005\\z\301o\012\230Z\014\362\244\324&\2
43g\351\362\325\375
\213\032\226$\2751\256XR\346k\266\030\234\267\201vUh\004\250\337A\231\223u\247\
366/i\022\275\276\350\2
20\316\306|\203+\010\261;\232\254tp\255\243\261\373Rq;\316w\357\006\207\374U\33
3\365\365\245hg\031\005
\322\347ea\220\0151\212g\337\264\336b\263\004\311\210.4\340G+\221\274D\035\375\
2216\241'\346a0\273wE\2
12\342y^\202\262|A7\202t\240\333p\345G\373\253\243oCO\011\360\247\211\014\024{\
272\271\322<\001\267
\347\240\005\213\0078\036\210\307$\317\322\311\222\035\354\006<\266\264\004\376
\251q\256\220(+\030\
3270\013c\327\272\212%\363\033\252\322\337\354\276\225\232\201\212^\304\210\226
9@\3230\370{
```

4. Extract the public.key ID from the database:

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

mQENBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jtWiXZTh
2His1ojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgKlGSgd9texg2nnSL9Admqik/yX
R5syVKG+qcdWuvyZg9oOOmeyjhc3n+kkbRTEMuM3flbMs8shOwzMvstCUVmuHU/V
vG5rJAe8PuYDSJCJ74I6w7SOH3RiRIc7IfL6xYddV42l3ctd44bl8/i71hq2UyN2
/Hbsjii2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFnFnE6Pwbhn+
ic8MD0lK5/GAlR6Hc0ZIHf8KEcavruQlikjnABEBAAG0HHRlc3Qga2V5IDx0ZXN0
a2V5QGVtYWlsLmNvbT6JAT4EEwECACgFAlS1Zf0CGwMFCQHhM4AGCwkIBwMCBhUI
AgkKCwQWAgMBAh4BAheAAAoJEAd9cl4gJ8wwbfwH/3VyVsPkQl1owRJNxvXGt1bY
7BfrvU52yk+PPZYoes9UpdL3CMRk8gAM9bx5Sk08q2UXSZLC6fFOpEW4uWgmGYf8
JRoC3ooezTkmCBW8I1bU0qGetzVxopdXLuPGCE7hVWQe9HcSntiTLxGov1mJAwO7
TAoccXLbyuZh9Rf5vLoQdKzcCyOHh5IqXaQOT100TeFeEpb9TIiwcntg3WCSU5P0
DGoUAOanjDZ3KE8Qp7V74fhG1EZVzHb8FajR62CXSHFKqpBgiNxnTOk45NbXADn4
eTUXPSnwPi46qoAp9UQogsfGyB1XDOTB2UOqhutAMECaM7VtpePv79i0Z/NfnBe5
AQ0EVLVl/QEIANabFdQ+8QMCADOipM1bF/JrQt3zUoc4BTqICaxdyzAfz0tUSf/7
Zro2us99GlARqLWd8EqJcl/xmfcJiZyUam6ZAzzFXCgnH5Y1sdtMTJZdLp5WeOjw
gCWG/ZLu4wzxOFFzDkiPv9RDw6e5MNLtJrSp4hS5o2apKdbO4Ex83O4mJYnav/rE
iDDCWU4T0lhv3hSKCpke6LcwsX+7liozp+aNmP0Ypwfi4hR3UUMP70+V1beFqW2J
bVLz3lLLouHRgpCzla+PzzbEKs16jq77vG9kqZTCIzXoWaLljuitRlfJkO3vQ9hO
v/8yAnkcAmowZrIBlyFg2KBzhunYmN2YvkUAEQEAAYkBJQQYAQIADwUCVLVl/QIb
DAUJAeEzgAAKCRAHfXJeICfMMOHYCACFhInZA9uAM3TC44l+MrgMUJ3rW9izrO48
WrdTsxR8WkSNbIxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRrI61FPHQNPSvz62
WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8Hkk8qb5x/
HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNlGWE8pvgEx
/UUZB+dYqCwtvX0nnBu1KNCmk2AkEcFK3YoliCxomdOxhFOv9AKjjojDyC65KJci
Pv2MikPS2fKOAg1R3LpMa8zDEtl4w3vckPQNrQNnYuUtfj6ZoCxv
=XZ8J
-----END PGP PUBLIC KEY BLOCK-----'));

pgp_key_id | 9D4D255F4FD2EFBB
```

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;
username | Bob
key_used | 9D4D255F4FD2EFBB
---------+-----------------
username | Alice
```

```
key_used | 9D4D255F4FD2EFBB
```

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)

lQOYBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jtWiXZTh
2His1ojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgKlGSgd9texg2nnSL9Admqik/yX
R5syVKG+qcdWuvyZg9oOOmeyjhc3n+kkbRTEMuM3flbMs8shOwzMvstCUVmuHU/V
vG5rJAe8PuYDSJCJ74I6w7SOH3RiRIc7IfL6xYddV42l3ctd44bl8/i71hq2UyN2
/Hbsjii2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFnFnE6Pwbhn+
ic8MD0lK5/GAlR6Hc0ZIHf8KEcavruQlikjnABEBAAEAB/wNfjjvP1brRfjjIm/j
XwUNm+sI4v2Ur7qZC94VTukPGf67lvqcYZJuqXxvZrZ8bl6mvl65xEUiZYy7BNA8
fe0PaM4Wy+Xr94Cz2bPbWgawnRNN3GAQy4rlBTrvqQWy+kmpbd87iTjwZidZNNmx
02iSzraq41Rt0Zx21Jh4rkpF67ftmzOH0vlrS0bWOvHUeMY7tCwmdPe9HbQeDlPr
n9CllUqBn4/acTtCClWAjREZn0zXAsNixtTIPC1V+9nO9YmecMkVwNfIPkIhymAM
OPFnuZ/Dz1rCRHjNHb5j6ZyUM5zDqUVnnezktxqrOENSxm0gfMGcpxHQogUMzb7c
6UyBBADSCXHPfo/VPVtMm5p1yGrNOR2jR2rUj9+poZzD2gjkt5G/xIKRlkB4uoQl
emu27wr9dVEX7ms0nvDq58iutbQ4d0JIDlcHMeSRQZluErblB75Vj3HtImblPjpn
4Jx6SWRXPUJPGXGI87u0UoBH0Lwij7M2PW7l1ao+MLEA9jAjQwQA+sr9BKPL4Ya2
r5nE72gsbCCLowkC0rdldf1RGtobwYDMpmYZhOaRKjkOTMG6rCXJxrf6LqiN8w/L
/gNziTmch35MCq/MZzA/bN4VMPyeIlwzxVZkJLsQ7yyqX/A7ac7B7DH0KfXciEXW
MSOAJhMmklW1Q1RRNw3cnYi8w3q7X40EAL/w54FVvvPqp3+sCd86SAAapM4UO2R3
tIsuNVemMWdgNXwvK8AJsz7VreVU5yZ4B8hvCuQj1C7geaN/LXhiT8foRsJC5o71
Bf+iHC/VNEv4k4uDb4lOgnHJYYyifB1wC+nn/EnXCZYQINMia1a4M6Vqc/RIfTH4
nwkZt/89LsAiR/20HHRlc3Qga2V5IDx0ZXN0a2V5QGVtYWlsLmNvbT6JAT4EEwEC
ACgFAlS1Zf0CGwMFCQHhM4AGCwkIBwMCBhUIAgkKCwQWAgMBAh4BAheAAAoJEAd9
cl4gJ8wwbfwH/3VyVsPkQ1lowRJNxvXGt1bY7BfrvU52yk+PPZYoes9UpdL3CMRk
8gAM9bx5Sk08q2UXSZLC6fFOpEW4uWgmGYf8JRoC3ooezTkmCBW8I1bU0qGetzVx
opdXLuPGCE7hVWQe9HcSntiTLxGov1mJAwO7TAoccXLbyuZh9Rf5vLoQdKzcCyOH
h5IqXaQOT100TeFeEpb9TIiwcntg3WCSU5P0DGoUAOanjDZ3KE8Qp7V74fhG1EZV
zHb8FajR62CXSHFKqpBgiNxnTOk45NbXADn4eTUXPSnwPi46qoAp9UQogsfGyB1X
DOTB2UOqhutAMECaM7VtpePv79i0Z/NfnBedA5gEVLVl/QEIANabFdQ+8QMCADOi
pM1bF/JrQt3zUoc4BTqICaxdyzAfz0tUSf/7Zro2us99GlARqLWd8EqJcl/xmfcJ
iZyUam6ZAzzFXCgnH5Y1sdtMTJZdLp5WeOjwgCWG/ZLu4wzxOFFzDkiPv9RDw6e5
MNLtJrSp4hS5o2apKdbO4Ex83O4mJYnav/rEiDDCWU4T0lhv3hSKCpke6LcwsX+7
liozp+aNmP0Ypwfi4hR3UUMP70+V1beFqW2JbVLz3lLLouHRgpCzla+PzzbEKs16
jq77vG9kqZTCIzXoWaLljuitRlfJkO3vQ9hOv/8yAnkcAmowZrIBlyFg2KBzhunY
mN2YvkUAEQEAAQAH/A7r4hDrnmzX3QU6FAzePlRB7niJtE2IEN8AufF05Q2PzKU/
c1S72WjtqMAIAgYasDkOhfhcxanTneGuFVYggKT3eSDm1RFKpRjX22m0zKdwy67B
Mu95V2Oklul6OCm8dO6+2fmkGxGqc4ZsKy+jQxtxK3HG9YxMC0dvA2v2C5N4TWi3
Utc7zh//k6IbmaLd7F1d7DXt7Hn2Qsmo8I1rtgPE8grDToomTnRUodToyejEqKyI
ORwsp8n8g2CSFaXSrEyU6HbFYXSxZealhQJGYLFOZdR0MzVtZQCn/7n+IHjupndC
Nd2a8DVx3yQS3dAmvLzhFacZdjXi31wvj0moFOkEAOCz1E63SKNNksniQ11lRMJp
gaov6Ux/zGLMstwTzNouI+Kr8/db0GlSAy1Z3UoAB4tFQXEApoX9A4AJ2KqQjqOX
cZVULenfDZaxrbb9Lid7ZnTDXKVyGTWDF7ZHavHJ4981mCW17lU11zHBB9xMlx6p
dhFvb0gdy0jSLaFMFr/JBAD0fz3RrhP7e6Xll2zdBqGthjC5S/IoKwwBgw6ri2yx
LoxqBr2pl9PotJJ/JUMPhD/LxuTcOZtYjy8PKgm5jhnBDq3Ss0kNKAY1f5EkZG9a
6I4iAX/NekqSyF+OgBfC9aCgS5RG8hYoOCbp8na5R3bgiuS8IzmVmm5OhZ4MDEwg
nQP7BzmR0p5BahpZ8r3Ada7FcK+0ZLLRdLmOYF/yUrZ53SoYCZRzU/GmtQ7LkXBh
Gjqied9Bs1MHdNUolq7GaexcjZmOWHEf6w9+9M4+vxtQq1nkIWqtaphewEmd5/nf
EP3sIY0EAE3mmiLmHLqBju+UJKMNwFNeyMTqgcg50ISH8J9FRIkBJQQYAQIADwUC
VLVl/QIbDAUJAeEzgAAKCRAHfXJeICfMMOHYCACFhInZA9uAM3TC44l+MrgMUJ3r
W9izrO48WrdTsxR8WkSNbIxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRrI61FPH
QNPSvz62WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8H
kk8qb5x/HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNlG
WE8pvgEx/UUZB+dYqCwtvX0nnBu1KNCmk2AkEcFK3YoliCxomdOxhFOv9AKjjojD
yC65KJciPv2MikPS2fKOAg1R3LpMa8zDEtl4w3vckPQNrQNnYuUtfj6ZoCxv
=fa+6
-----END PGP PRIVATE KEY BLOCK-----') AS privkey) AS keys;

username | decrypted_ssn
```

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

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.

[1] SHA2 algorithms were added to OpenSSL in version 0.9.8. For older versions, pgcrypto will use built-in code.

[2] Any digest algorithm OpenSSL supports is automatically picked up. This is not possible with ciphers, which need to be supported explicitly.

[3] AES is included in OpenSSL since version 0.9.7. For older versions, pgcrypto will use built-in code.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Enabling gphdfs Authentication with a Kerberos-secured Hadoop Cluster (Deprecated)

Provides steps for configuring Greenplum Database to access external tables in a Hadoop cluster secured with Kerberos.

Note: The `gphdfs` external table protocol is deprecated and will be removed in the next major release of Greenplum Database. Consider using the Greenplum Platform Extension Framework (PXF) `pxf` external table protocol to access data stored in a Hadoop file system.

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

Parent topic: Greenplum Database Security Configuration Guide

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@L
OCAL.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/smd
w gphdfs/smdw.example.com@LOCAL.DOMAIN"
# kadmin.local -q "xst -k /etc/security/keytabs/sdw1.service.keytab gpadmin/sdw
1 gphdfs/sdw1.example.com@LOCAL.DOMAIN"
# kadmin.local -q "xst -k /etc/security/keytabs/sdw2.service.keytab gpadmin/sdw
2 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.servi
ce.keytab
# scp /etc/security/keytabs/smdw.service.keytab smdw_fqdn:/home/gpadmin/smdw.se
rvice.keytab
# scp /etc/security/keytabs/sdw1.service.keytab sdw1_fqdn:/home/gpadmin/sdw1.se
rvice.keytab
# scp /etc/security/keytabs/sdw2.service.keytab sdw2_fqdn:/home/gpadmin/sdw2.se
rvice.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:

```
<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:

```
<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 cluser.

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:

```
export HADOOP_OPTS="-Djava.net.prefIPv4Stack=true -Dsun.security.krb5.debug=true ${HAD
OOP_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:

```
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.

```
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.

```
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:

```
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:

```
source /usr/local/greenplum-db/greenplum_path.sh
source env.sh
source $GPHOME/lib/hadoop-env.sh
```

4. Test the Greenplum Database HDFS reader:

```
java com.emc.greenplum.gpdb.hdfsconnector.HDFSReader 0 32 TEXT hdp2 gphdfs://namenode:8020/tmp/test1.txt
```

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 `REASIGN 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 sufficently 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.

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

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
```

Parent topic: Greenplum Database Security Configuration Guide

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.
- **Managing a Greenplum System**
  This section describes basic system administration tasks performed by a Greenplum Database system administrator.
- **Managing Greenplum Database Access**
  Securing Greenplum Database includes protecting access to the database through network configuration, database user authentication, and encryption.
- **Defining Database Objects**
  This section covers data definition language (DDL) in Greenplum Database and how to create and manage database objects.
- **Distribution and Skew**
  Greenplum Database relies on even distribution of data across segments.
- **Inserting, Updating, and Deleting Data**
  This section provides information about manipulating data and concurrent access in Greenplum Database.
- **Querying Data**
  This topic provides information about using SQL in Greenplum databases.
- **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.
- **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.
- **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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

- **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.

- **About Management and Monitoring Utilities**
  Greenplum Database provides standard command-line utilities for performing common monitoring and administration tasks.
- **About Concurrency Control in Greenplum Database**
  Greenplum Database uses the PostgreSQL Multiversion Concurrency Control (MVCC) model to manage concurrent transactions for heap tables.
- **About Parallel Data Loading**
  This topic provides a short introduction to Greenplum Database data loading features.
- **About Redundancy and Failover in Greenplum Database**
  This topic provides a high-level overview of Greenplum Database high availability features.
- **About Database Statistics in Greenplum Database**
  An overview of statistics gathered by the `ANALYZE` command in Greenplum Database.

Parent topic: Greenplum Database Administrator Guide

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.

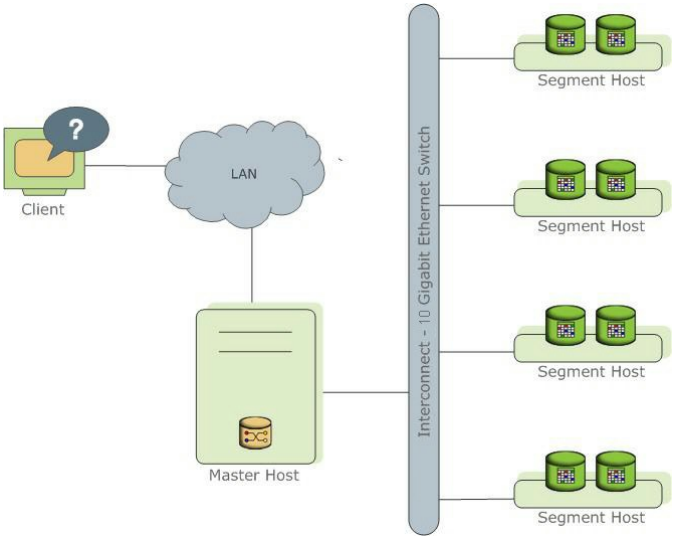
Greenplum Database also can use declarative partitions and sub-partitions to implicitly generate partition constraints.

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 1. High-Level Greenplum Database Architecture



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

- About the Greenplum Master
- About the Greenplum Segments
- About the Greenplum Interconnect

Parent topic: Greenplum Database Concepts

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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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:

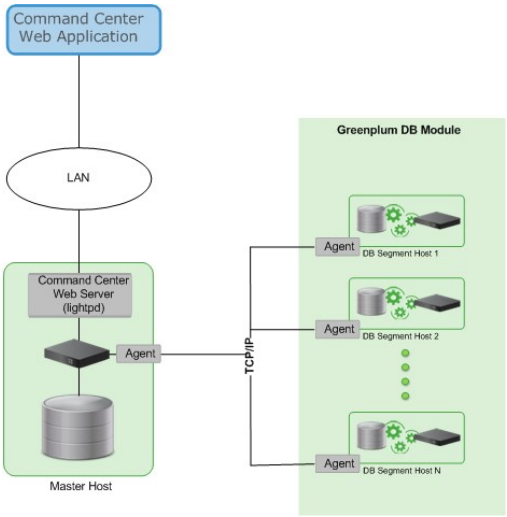
- Installing Greenplum Database on an array
- Initializing a Greenplum Database System
- Starting and stopping Greenplum Database

- 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 1. Greenplum Command Center Architecture



Parent topic: Greenplum Database Concepts

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 segment 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 `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 trans
actions
```

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 "d
atabase_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 1. Transaction Isolation Modes

Level	Dirty Read	Non-Repeatable	Phantom Read
Read Uncommitted	Possible	Possible	Possible
Read Committed	Impossible	Possible	Possible
Repeatable Read	Impossible	Impossible	Possible
Serializable	Impossible	Impossible	Impossible

The Greenplum Database SQL commands allow you to request `READ UNCOMMITTED`, `READ COMMITTED`, or `SERIALIZABLE`. Greenplum Database treats `READ UNCOMMITTED` the same as `READ COMMITTED`. Requesting `REPEATABLE READ` produces an error; use `SERIALIZABLE` instead. The default isolation mode is `READ COMMITTED`.

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

The `READ COMMITTED` isolation mode permits greater concurrency and better performance than the `SERIALIZABLE` mode. It allows *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. `READ COMMITTED` mode also permits *phantom reads*, where a query executed twice in the same transaction can return two different sets of rows.

The `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 `SERIALIZABLE` mode must be prepared to handle transactions that fail due to serialization errors. If `SERIALIZABLE` isolation mode is not required by the application, it is better to use `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 `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 *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 `SET TRANSACTION ISOLATION LEVEL` statement sets the isolation mode for the current transaction. The mode must be set before any `SELECT`, `INSERT`, `DELETE`, `UPDATE`, or `COPY` statements:

```
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
...
COMMIT;
```

The isolation mode can also be specified as part of the `BEGIN` statement:

```
BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

The default transaction isolation mode can be changed for a session by setting the *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 `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 *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.

```
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**

Parent topic: Greenplum Database Concepts

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 3

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.
Parent topic: About Concurrency Control in Greenplum Database

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, `xmin` (transaction that created the row) and `xmax` (transaction that updated the row). In the table, changes are added, in order, to the bottom of the table.

Table 1. Example Table

item	amount	xmin	xmax
widget	100	0	null

Table 1. Example Table

item	amount	xmin	xmax
giblet	200	1	null
sprocket	300	2	null
gizmo	400	3	null

The next table shows the table data on disk after some updates on the amount values have been performed.

- xid = 4: `update tbl set amount=208 where item = 'widget'`
- xid = 5: `update tbl set amount=133 where item = 'sprocket'`
- xid = 6: `update tbl set amount=16 where item = 'widget'`

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 2. Example Table with Updates

item	amount	xmin	xmax
widget	100	0	4
giblet	**200**	1	**null**
sprocket	300	2	5
gizmo	**400**	3	**null**
widget	208	4	6
sproket	**133**	5	**null**
widget	**16**	6	**null**

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_transaction
  s 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 data
  base "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 `obsolete` to indicate obsolete rows. A Greenplum Database `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 3. Example Table after VACUUM

item	amount	xmin	xmax
widget	100	obsolete	obsolete
giblet	**200**	-2	**null**
sprocket	300	obsolete	obsolete
gizmo	**400**	-2	**null**
widget	208	4	6
sproket	**133**	5	**null**
widget	**16**	6	**null**

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 4. Example Table with Wrapping XID

item	amount	xmin	xmax
widget	100	obsolete	obsolete
giblet	200	-2	1
sprocket	300	obsolete	obsolete
gizmo	400	-2	9
widget	208	4	6
sproket	**133**	5	**null**
widget	16	6	7

Table 4. Example Table with Wrapping XID

item	amount	xmin	xmax
widget	**222**	7	**null**
giblet	233	8	0
gizmo	**18**	9	**null**
giblet	88	0	1
giblet	**44**	1	**null**

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`.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

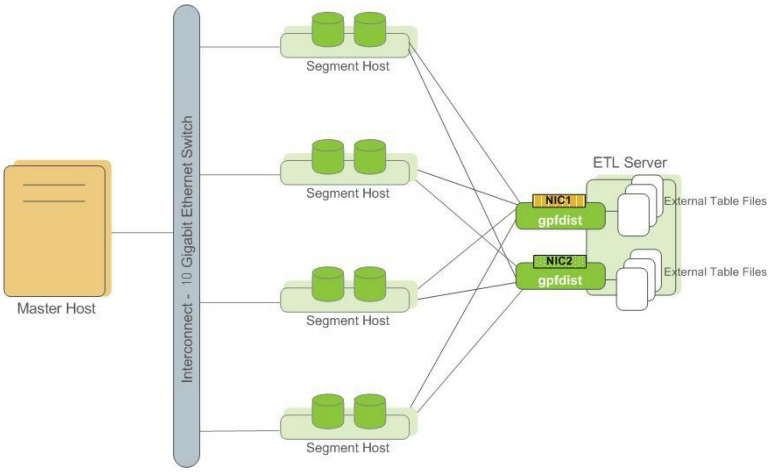
About Parallel Data Loading

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.

Figure 1. External Tables Using Greenplum Parallel File Server (gpfdist)



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.

Parent topic: Greenplum Database Concepts

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

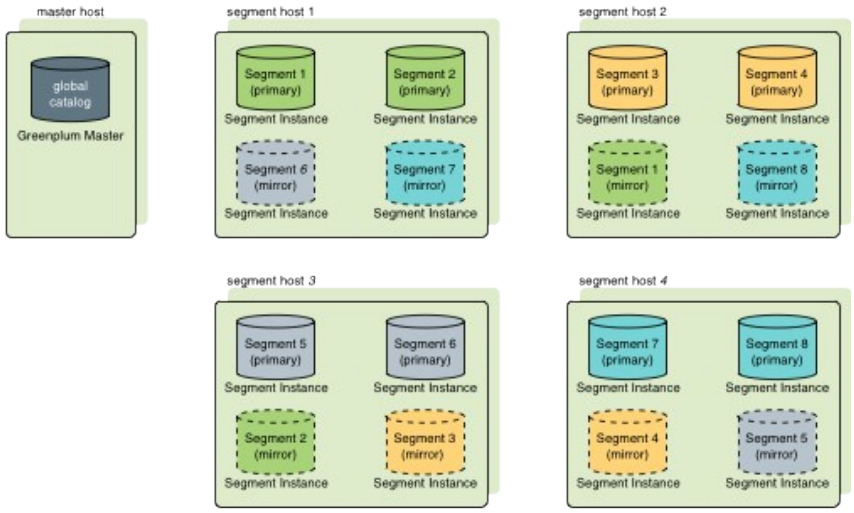
When you deploy your Greenplum Database system, you can configure *mirror* segments. Mirror segments allow database queries to fail over to a backup segment if the primary segment becomes unavailable. Mirroring is strongly recommended for production systems and required for Pivotal support.

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 1 shows how table data is distributed across segments when spread mirroring is configured.

Figure 1. 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.

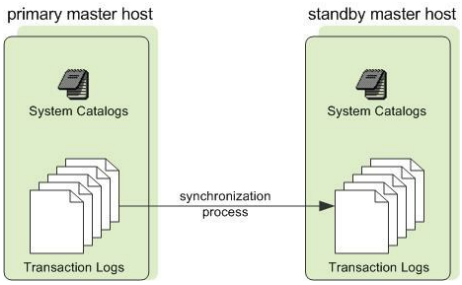
About Master Mirroring

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.

Figure 2. Master Mirroring in Greenplum Database



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.

Parent topic: Greenplum Database Concepts

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 stastistics 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.

Parent topic: Greenplum Database Concepts

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 `relpages`, 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 `relpages` 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 `relpages` 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.

staattnum
: 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.

stakind*N*
: A code number indicating the kind of statistics stored in the *N*th slot of the `pg_statistic` row.

staop*N*
: An operator used to derive the statistics stored in the *N*th slot. For example, a histogram slot would show the < operator that defines the sort order of the data.

stanumbers*N*
: float4 array containing numerical statistics of the appropriate kind for the *N*th slot, or `NULL` if the slot kind does not involve numerical values.

stavalues*N*
: Column data values of the appropriate kind for the *N*th 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 `stakind`*N* 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, 3, and 99. A value of 0 means the slot is unused. The following table describes the kinds of statistics stored for the three codes.

Table 1. Contents of pg_statistic "slots"

stakind Code	Description
1	*Most CommonValues (MCV) Slot* • `staop` contains the object ID of the "=" operator, used to decide whether values are the same or not. • `stavalues` contains an array of the *K* most common non-null values appearing in the column. • `stanumbers` contains the frequencies (fractions of total row count) of the values in the `stavalues` array. 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.
2	*Histogram Slot* – describes the distribution of scalar data. • `staop` is the object ID of the "<" operator, which describes the sort ordering. • `stavalues` contains *M* (where *M*>=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. • `stanumbers` is not used and should be `NULL`. 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 *compressed histogram* 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.

Table 1. Contents of pg_statistic "slots"

stakind Code	Description
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.
99	*Hyperloglog Slot* - for child leaf partitions of a partitioned table, stores the `hyperloglog_counter` created for the sampled data. The `hyperloglog_counter` data structure is converted into a `bytea` and stored in a `stavalues4` slot of the `pg_statistic` catalog table.

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 `avg(pg_column_size(`*`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 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 .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 Tables

When the `ANALYZE` command is run on a partitioned table, it analyzes each child leaf partition table, one at a time. You can run `ANALYZE` on just new or changed partition files to avoid analyzing partitions that have not changed.

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` or `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.

The time to analyze a partitioned table 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 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 `on` (the default), the `ROOTPARTITION` keyword is not required to collect statistics on the root partition when you run `ANALYZE`. Root partition statistics are collected when you run `ANALYZE` on the root partition, or when you run `ANALYZE` on a child leaf partition of the partitioned table and the other

child leaf partitions have statistics. If the parameter is `off`, you must run `ANALYZE ROOTPARTITION` to collect root partition statistics.

If you do not intend to execute queries on partitioned tables with GPORCA (setting the server configuration parameter `optimizer` to `off`), you can also set the server configuration parameter `optimizer_analyze_root_partition` to `off` to limit when `ANALYZE` updates the root partition statistics.

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 100 to 150 you can use the `gpconfig` utility:

```
gpconfig -c default_statistics_target -v 150
```

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 otpimization. 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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Managing a Greenplum System

This section describes basic system administration tasks performed by a Greenplum Database system administrator.

- **About the Greenplum Database Release Version Number**
  Greenplum Database version numbers and they way they change identify what has been modified from one Greenplum Database release to the next.
- **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.
- **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.
- **Configuring the Greenplum Database System**
  Server configuration parameters affect the behavior of Greenplum Database.
- **Enabling High Availability and Data Consistency Features**
  The fault tolerance and the high-availability features of Greenplum Database can be configured.
- **Backing Up and Restoring Databases**

This topic describes how to use Greenplum backup and restore features.

- **Expanding a Greenplum System**
  To scale up performance and storage capacity, expand your Greenplum Database system by adding hosts to the system. In general, adding nodes to a Greenplum cluster achieves a linear scaling of performance and storage capacity.
- **Migrating Data**
- **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.
- **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.
- **Recommended Monitoring and Maintenance Tasks**
  This section lists monitoring and maintenance activities recommended to ensure high availability and consistent performance of your Greenplum Database cluster.

Parent topic: Greenplum Database Administrator Guide

About the Greenplum Database Release Version Number

Greenplum Database version numbers and they way they change identify what has been modified from one Greenplum Database release to the next.

A Greenplum Database release version number takes the format *x.y.z*, where:

- *x* identifies the Major version number
- *y* identifies the Minor version number
- *z* identifies the Patch version number

Greenplum Database releases that have the same Major release number are guaranteed to be backwards compatible. Greenplum Database increments the Major release number when the catalog changes or when incompatible feature changes or new features are introduced. Previously deprecated functionality may be removed in a major release.

The Minor release number for a given Major release increments when backwards compatible new features are introduced or when a Greenplum Database feature is deprecated. (Previously deprecated functionality will never be removed in a minor release.)

Greenplum Database increments the Patch release number for a given Minor release for backwards-compatible bug fixes.

Parent topic: Managing a Greenplum System

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*.

Parent topic: Managing a Greenplum System

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:

```
$ 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.

Stopping Client Processes

Greenplum Database launches a new backend process for each client connection. A Greenplum Database user with `SUPERUSER` privileges can cancel and terminate these client backend processes.

Canceling a backend process with the `pg_cancel_backend()` function ends a specific queued or active client query. Terminating a backend process with the `pg_terminate_backend()` function terminates a client connection to a database.

The `pg_cancel_backend()` function has two signatures:

- `pg_cancel_backend( pid int4 )`
- `pg_cancel_backend( pid int4, msg text )`

The `pg_terminate_backend()` function has two similar signatures:

- `pg_terminate_backend( pid int4 )`
- `pg_terminate_backend( pid int4, msg text )`

If you provide a `msg`, Greenplum Database includes the text in the cancel message returned to the client. `msg` is limited to 128 bytes; Greenplum Database truncates anything longer.

The `pg_cancel_backend()` and `pg_terminate_backend()` functions return `true` if successful, and `false` otherwise.

To cancel or terminate a backend process, you must first identify the process ID of the backend. You

can obtain the process ID from the `procpid` column of the `pg_stat_activity` view. For example, to view the process information associated with all running and queued queries:

```
=# SELECT usename, procpid, waiting, current_query, datname
     FROM pg_stat_activity;
```

Sample partial query output:

```
 usename |  procpid | waiting |     current_query     | datname
---------+----------+---------+-----------------------+---------
  sammy  |   31861  |    f    | <IDLE> in transaction | testdb
  billy  |   31905  |    t    | SELECT * FROM topten; | testdb
```

Use the output to identify the process id (`procpid`) of the query or client connection.

For example, to cancel the pending query identified in the sample output above and include "Admin canceled long-running query." in the message returned to the client:

```
=# SELECT pg_cancel_backend(31905 ,'Admin canceled long-running query.');
ERROR:  canceling statement due to user request: "Admin canceled long-running query."
```

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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**
- **Supported Client Applications**
- **Greenplum Database Client Applications**
- **Connecting with psql**
- **Using the PgBouncer Connection Pooler**
- **Database Application Interfaces**
- **Troubleshooting Connection Problems**

Parent topic: Managing a Greenplum System

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 1. Connection Parameters

Connection Parameter	Description	Environment Variable
Application name	The application name that is connecting to the database. The default value, held in the `application_name` connection parameter is *psql*.	`$PGAPPNAME`
Database name	The name of the database to which you want to connect. For a newly initialized system, use the `postgres` database to connect for the first time.	`$PGDATABASE`

Table 1. Connection Parameters

Connection Parameter	Description	Environment Variable
Host name	The host name of the Greenplum Database master. The default host is the local host.	`$PGHOST`
Port	The port number that the Greenplum Database master instance is running on. The default is 5432.	`$PGPORT`
User name	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`).	`$PGUSER`

Connecting with psql provides example commands for connecting to Greenplum Database.

Parent topic: Accessing the Database

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Accessing the Database

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 1. Commonly used client applications

Name	Usage
`createdb`	create a new database
`createlang`	define a new procedural language
`createuser`	define a new database role
`dropdb`	remove a database
`droplang`	remove a procedural language
`dropuser`	remove a role
`psql`	PostgreSQL interactive terminal
`reindexdb`	reindex a database

Table 1. Commonly used client applications

Name	Usage
`vacuumdb`	garbage-collect and analyze a database

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*.

Parent topic: Accessing the Database

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`:

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

```
$ psql gpdatabase
```

```
$ psql
```

If a user-defined database has not yet been created, you can access the system by connecting to the `postgres` database. For example:

```
$ 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:

```
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 * FROM mytable;
```

See the *Greenplum Reference Guide* for information about using the `psql` client application and SQL commands and syntax.

Parent topic: Accessing the Database

up-to-date release of the Greenplum 5.x documentation.

Using the PgBouncer Connection Pooler

The PgBouncer utility manages connection pools for PostgreSQL and Greenplum Database connections.

The following topics describe how to set up and use PgBouncer with Greenplum Database. Refer to the PgBouncer web site for information about using PgBouncer with PostgreSQL.

- Overview
- Migrating PgBouncer
- Configuring PgBouncer
- Starting PgBouncer
- Managing PgBouncer

Parent topic: Accessing the Database

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 new database connections, so clients connect much faster and the server load is reduced.

The PgBouncer connection pooler, from the PostgreSQL community, is included in your Greenplum Database installation. PgBouncer is a light-weight connection pool manager for Greenplum and PostgreSQL. PgBouncer maintains a pool for connections for each database and user combination. PgBouncer either creates a new database connection for a client or reuses an existing connection for the same user and database. When the client disconnects, PgBouncer returns the connection to the pool for re-use.

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.

You can set a default pool mode for the PgBouncer instance. You can override this mode for individual databases and users.

PgBouncer supports the standard connection interface shared by PostgreSQL and Greenplum Database. The Greenplum Database client application (for example, `psql`) connects to the host and port on which PgBouncer is running rather than the Greenplum Database master host and port.

PgBouncer includes a `psql`-like administration console. Authorized users can connect to a virtual database to monitor and manage PgBouncer. You can manage a PgBouncer daemon process via the admin console. You can also use the console to update and reload PgBouncer configuration at runtime without stopping and restarting the process.

PgBouncer natively supports TLS.

Migrating PgBouncer

When you migrate to a new Greenplum Database version, you must migrate your PgBouncer

instance to that in the new Greenplum Database installation.

- **If you are migrating to a Greenplum Database version 5.8.x or earlier**, you can migrate PgBouncer without dropping connections. Launch the new PgBouncer process with the `-R` option and the configuration file that you started the process with:

  ```
  $ pgbouncer -R -d pgbouncer.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 detects 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.
- **If you are migrating to a Greenplum Database version 5.9.0 or later**, you must shut down the PgBouncer instance in your old installation and reconfigure and restart PgBouncer in your new installation.
- If you used stunnel to secure PgBouncer connections in your old installation, you must configure SSL/TLS in your new installation using the built-in TLS capabilities of PgBouncer 1.8.1 and later.
- If you used LDAP authentication in your old installation, you must configure LDAP in your new installation using the built-in PAM integration capabilities of PgBouncer 1.8.1 and later. You must also remove or replace any `ldap://`-prefixed password strings in the `auth_file`.

Configuring PgBouncer

You configure PgBouncer and its access to Greenplum Database via a configuration file. This configuration file, commonly named `pgbouncer.ini`, provides location information for Greenplum databases. The `pgbouncer.ini` file also specifies process, connection pool, authorized users, and authentication configuration for PgBouncer.

Sample `pgbouncer.ini` file contents:

```
[databases]
postgres = host=127.0.0.1 port=5432 dbname=postgres
pgb_mydb = host=127.0.0.1 port=5432 dbname=mydb

[pgbouncer]
pool_mode = session
listen_port = 6543
listen_addr = 127.0.0.1
auth_type = md5
auth_file = users.txt
logfile = pgbouncer.log
pidfile = pgbouncer.pid
admin_users = gpadmin
```

Refer to the pgbouncer.ini reference page for the PgBouncer configuration file format and the list of configuration properties it supports.

When a client connects to PgBouncer, the connection pooler looks up the the configuration for the requested database (which may be an alias for the actual database) that was specified in the `pgbouncer.ini` configuration file to find the host name, port, and database name for the database connection. The configuration file also identifies the authentication mode in effect for the database.

PgBouncer requires an authentication file, a text file that contains a list of Greenplum Database users

and passwords. The contents of the file are dependent on the `auth_type` you configure in the `pgbouncer.ini` file. Passwords may be either clear text or MD5-encoded strings. You can also configure PgBouncer to query the destination database to obtain password information for users that are not in the authentication file.

PgBouncer Authentication File Format

PgBouncer requires its own user authentication file. You specify the name of this file in the `auth_file` property of the `pgbouncer.ini` configuration file. `auth_file` is a text file in the following format:

```
"username1" "password" ...
"username2" "md5abcdef012342345" ...
```

`auth_file` contains one line per user. Each line must have at least two fields, both of which are enclosed in double quotes (" "). The first field identifies the Greenplum Database user name. The second field is either a plain-text or an MD5-encoded password. PgBouncer ignores the remainder of the line.

(The format of `auth_file` is similar to that of the `pg_auth` text file that Greenplum Database uses for authentication information. PgBouncer can work directly with this Greenplum Database authentication file.)

Use an MD5 encoded password. The format of an MD5 encoded password is:

```
"md5" + MD5_encoded(<password><username>)
```

You can also obtain the MD5-encoded passwords of all Greenplum Database users from the `pg_shadow` view.

Configuring HBA-based Authentication for PgBouncer

PgBouncer supports HBA-based authentication. To configure HBA-based authentication for PgBouncer, you set `auth_type=hba` in the `pgbouncer.ini` configuration file. You also provide the filename of the HBA-format file in the `auth_hba_file` parameter of the `pgbouncer.ini` file.

Contents of an example PgBouncer HBA file named `hba_bouncer.conf`:

```
local       all     bouncer             trust
host        all     bouncer      127.0.0.1/32       trust
```

Example excerpt from the related `pgbouncer.ini` configuration file:

```
[databases]
p0 = port=15432 host=127.0.0.1 dbname=p0 user=bouncer pool_size=2
p1 = port=15432 host=127.0.0.1 dbname=p1 user=bouncer
...

[pgbouncer]
...
auth_type = hba
auth_file = userlist.txt
auth_hba_file = hba_bouncer.conf
...
```

Refer to the HBA file format discussion in the PgBouncer documentation for information about PgBouncer support of the HBA authentication file format.

Starting PgBouncer

You can run PgBouncer 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, add the following text to a file named `pgbouncer.ini`:

   ```
   [databases]
   postgres = host=127.0.0.1 port=5432 dbname=postgres
   pgb_mydb = host=127.0.0.1 port=5432 dbname=mydb

   [pgbouncer]
   pool_mode = session
   listen_port = 6543
   listen_addr = 127.0.0.1
   auth_type = md5
   auth_file = users.txt
   logfile = pgbouncer.log
   pidfile = pgbouncer.pid
   admin_users = gpadmin
   ```

   The file lists databases and their connection details. The file also configures the PgBouncer instance. Refer to the pgbouncer.ini reference page for information about the format and content of a PgBouncer configuration file.

2. Create an authentication file. The filename should be the name you specified for the `auth_file` parameter of the `pgbouncer.ini` file, `users.txt`. Each line contains a user name and password. The format of the password string matches the `auth_type` you configured in the PgBouncer configuration file. If the `auth_type` parameter is `plain`, the password string is a clear text password, for example:

   ```
   "gpadmin" "gpadmin1234"
   ```

   If the `auth_type` in the following example is `md5`, the authentication field must be MD5-encoded. The format for an MD5-encoded password is:

   ```
   "md5" + MD5_encoded(<password><username>)
   ```

3. Launch `pgbouncer`:

   ```
   $ $GPHOME/bin/pgbouncer -d pgbouncer.ini
   ```

   The `-d` option runs PgBouncer as a background (daemon) process. Refer to the pgbouncer reference page for the `pgbouncer` command syntax and options.

4. Update your client applications to connect to `pgbouncer` instead of directly to Greenplum Database server. For example, to connect to the Greenplum database named `mydb` configured above, run `psql` as follows:

   ```
   $ psql -p 6543 -U someuser pgb_mydb
   ```

   The `-p` option value is the `listen_port` that you configured for the PgBouncer instance.

Managing PgBouncer

PgBouncer provides a `psql`-like administration console. You log in to the PgBouncer Administration Console by specifying the PgBouncer port number and a virtual database named `pgbouncer`. The console accepts SQL-like commands that you can use to monitor, reconfigure, and manage PgBouncer.

For complete documentation of PgBouncer Administration Console commands, refer to the PgBouncer Administration Console command reference.

Follow these steps to get started with the PgBouncer Administration Console.

1. Use `psql` to log in to the `pgbouncer` virtual database:

   ```
   $ psql -p 6543 -U username pgbouncer
   ```

   The *username* that you specify must be listed in the `admin_users` parameter in the `pgbouncer.ini` configuration file. You can also log in to the PgBouncer Administration Console with the current Unix username if the `pgbouncer` process is running under that user's UID.

2. To view the available PgBouncer Administration Console commands, run the `SHOW help` command:

   ```
   pgbouncer=# SHOW help;
   NOTICE:  Console usage
   DETAIL:
       SHOW HELP|CONFIG|DATABASES|POOLS|CLIENTS|SERVERS|VERSION
       SHOW FDS|SOCKETS|ACTIVE_SOCKETS|LISTS|MEM
       SHOW DNS_HOSTS|DNS_ZONES
       SHOW STATS|STATS_TOTALS|STATS_AVERAGES
       SET key = arg
       RELOAD
       PAUSE [<db>]
       RESUME [<db>]
       DISABLE <db>
       ENABLE <db>
       KILL <db>
       SUSPEND
       SHUTDOWN
   ```

3. If you update PgBouncer configuration by editing the `pgbouncer.ini` configuration file, you use the `RELOAD` command to reload the file:

   ```
   pgbouncer=# RELOAD;
   ```

Mapping PgBouncer Clients to Greenplum Database Server Connections

To map a PgBouncer client to a Greenplum Database server connection, use the PgBouncer Administration Console `SHOW CLIENTS` and `SHOW SERVERS` commands:

1. Use `ptr` and `link` to map the local client connection to the server connection.
2. Use the `addr` and the `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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 1. Greenplum Database Interfaces

API	PostgreSQL Driver	Download Link

ODBC	Greenplum DataDirect ODBC Driver	https://network.pivotal.io/products/pivotal-gpdb.
JDBC	Greenplum DataDirect JDBC Driver	https://network.pivotal.io/products/pivotal-gpdb
Perl DBI	pgperl	http://search.cpan.org/dist/DBD-Pg/
Python DBI	pygresql	http://www.pygresql.org/
libpq C Library	libpq	https://www.postgresql.org/docs/8.3/static/libpq.html

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.

Parent topic: Accessing the Database

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 1. Common connection problems

Problem	Solution
No pg_hba.conf entry for host or user	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.
Greenplum Database is not running	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.
Network problems Interconnect timeouts	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`* . 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`.

Table 1. Common connection problems

Problem	Solution
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.

Parent topic: Accessing the Database

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**
- **Setting Configuration Parameters**
- **Viewing Server Configuration Parameter Settings**
- **Configuration Parameter Categories**

Parent topic: Managing a Greenplum System

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.

Parent topic: Configuring the Greenplum Database System

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**
- **Setting a Master Configuration Parameter**

Parent topic: Configuring the Greenplum Database System

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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
```

Parent topic: Setting Configuration Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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**
- **Setting Parameters at the Database Level**
- **Setting Parameters at the Role Level**
- **Setting Parameters in a Session**

Parent topic: Setting Configuration Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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*.

Parent topic: Setting a Master Configuration Parameter

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.

Parent topic: Setting a Master Configuration Parameter

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.

Parent topic: Setting a Master Configuration Parameter

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.

Parent topic: Setting a Master Configuration Parameter

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
```

Parent topic: Configuring the Greenplum Database System

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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**
- **System Resource Consumption Parameters**
- **Query Tuning Parameters**
- **Error Reporting and Logging Parameters**
- **System Monitoring Parameters**
- **Runtime Statistics Collection Parameters**
- **Automatic Statistics Collection Parameters**
- **Client Connection Default Parameters**
- **Lock Management Parameters**
- **Resource Management Parameters**
- **External Table Parameters**
- **Database Table Parameters**
- **Database and Tablespace/Filespace Parameters**
- **Past Version Compatibility Parameters**
- **Greenplum Array Configuration Parameters**
- **Greenplum Master and Segment Mirroring Parameters**
- **Greenplum Database Extension Parameters**

Parent topic: Configuring the Greenplum Database System

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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**
- **Security and Authentication Parameters**

Parent topic: Configuration Parameter Categories

A newer version of this documentation is available. Use the version menu above to view the most

up-to-date release of the Greenplum 5.x documentation.

Connection Parameters

gp_connection_send_timeout	tcp_keepalives_count
gp_vmem_idle_resource_timeout	tcp_keepalives_idle
listen_addresses	tcp_keepalives_interval
max_connections	unix_socket_directory
max_prepared_transactions	unix_socket_group
superuser_reserved_connections	unix_socket_permissions

Parent topic: Connection and Authentication Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Security and Authentication Parameters

authentication_timeout	krb_srvname
db_user_namespace	password_encryption
krb_caseins_users	password_hash_algorithm
krb_server_keyfile	ssl
	ssl_ciphers

Parent topic: Connection and Authentication Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

System Resource Consumption Parameters

- **Memory Consumption Parameters**
- **Free Space Map Parameters**
- **OS Resource Parameters**
- **Cost-Based Vacuum Delay Parameters**
- **Transaction ID Management Parameters**

Parent topic: Configuration Parameter Categories

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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	max_stack_depth
gp_vmem_protect_limit	shared_buffers
gp_vmem_protect_segworker_cache_limit	temp_buffers
gp_workfile_limit_files_per_query	
gp_workfile_limit_per_query	
gp_workfile_limit_per_segment	
max_appendonly_tables	
max_prepared_transactions	

Parent topic: System Resource Consumption Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: System Resource Consumption Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

OS Resource Parameters

- `max_files_per_process`
- `shared_preload_libraries`

Parent topic: System Resource Consumption Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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_page_hit
vacuum_cost_limit	vacuum_cost_page_miss
vacuum_cost_page_dirty	

Parent topic: System Resource Consumption Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Transaction ID Management Parameters

- xid_stop_limit
- xid_warn_limit

Parent topic: System Resource Consumption Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Query Tuning Parameters

- **GPORCA Configuration Parameters**
- **Query Plan Operator Control Parameters**
- **Legacy Query Optimizer Costing Parameters**
- **Database Statistics Sampling Parameters**
- **Sort Operator Configuration Parameters**
- **Aggregate Operator Configuration Parameters**
- **Join Operator Configuration Parameters**
- **Other Legacy Query Optimizer Configuration Parameters**

Parent topic: Configuration Parameter Categories

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

GPORCA Configuration Parameters

optimizer	optimizer_join_arity_for_associativity_commutativity
optimizer_analyze_root_partition	optimizer_join_order
optimizer_array_expansion_threshold	optimizer_join_order_threshold
optimizer_cte_inlining_bound	optimizer_mdcache_size
optimizer_enable_associativity	optimizer_metadata_caching
optimizer_control	optimizer_parallel_union
optimizer_enable_master_only_queries	optimizer_print_missing_stats
optimizer_force_multistage_agg	optimizer_print_optimization_stats
optimizer_force_three_stage_scalar_dqa	optimizer_sort_factor

Parent topic: Query Tuning Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

enable_bitmapscan	gp_enable_agg_distinct_pruning
enable_groupagg	gp_enable_direct_dispatch
enable_hashagg	gp_enable_fallback_plan
enable_hashjoin	gp_enable_fast_sri
enable_indexscan	gp_enable_groupext_distinct_ gather
enable_mergejoin	gp_enable_groupext_distinct_ pruning
enable_nestloop	gp_enable_multiphase_agg
enable_seqscan	gp_enable_predicate_ propagation
enable_sort	gp_enable_preunique
enable_tidscan	gp_enable_sequential_window_ plans
gp_enable_agg_distinct	gp_enable_sort_distinct
	gp_enable_sort_limit

Parent topic: Query Tuning Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

cpu_index_tuple_cost	gp_motion_cost_per_row
cpu_operator_cost	gp_segments_for_planner
cpu_tuple_cost	random_page_cost
cursor_tuple_fraction	seq_page_cost
effective_cache_size	

Parent topic: Query Tuning Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: Query Tuning Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Sort Operator Configuration Parameters

- `gp_enable_sort_distinct`
- `gp_enable_sort_limit`

Parent topic: Query Tuning Parameters

A newer version of this documentation is available. Use the version menu above to view the most

up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Query Tuning Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Query Tuning Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Other Legacy Query Optimizer Configuration Parameters

- from_collapse_limit
- gp_enable_predicate_propagation
- gp_max_plan_size
- gp_statistics_pullup_from_child_partition

Parent topic: Query Tuning Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Error Reporting and Logging Parameters

- **Log Rotation**
- **When to Log**
- **What to Log**

Parent topic: Configuration Parameter Categories

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Log Rotation

log_rotation_age log_rotation_size	log_truncate_on_rotation

Parent topic: Error Reporting and Logging Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

When to Log

client_min_messages	log_min_error_statement
gp_interconnect_debug_retry_interval	log_min_messages
log_error_verbosity	optimizer_minidump
log_min_duration_statement	

Parent topic: Error Reporting and Logging Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

What to Log

debug_pretty_print	log_executor_stats
debug_print_parse	log_hostname
debug_print_plan	gp_log_interconnect
debug_print_prelim_plan	log_parser_stats
debug_print_rewritten	log_planner_stats
debug_print_slice_table	log_statement
log_autostats	log_statement_stats
log_connections	log_timezone
log_disconnections	gp_debug_linger
log_dispatch_stats	gp_log_format
log_duration	gp_log_gang
	gp_max_csv_line_length
	gp_reraise_signal

Parent topic: Error Reporting and Logging Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

System Monitoring Parameters

- **SNMP Alerts**
- **Email Alerts**
- **Greenplum Command Center Agent**

Parent topic: Configuration Parameter Categories

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

SNMP Alerts

The following parameters send SNMP notifications when events occur.

gp_snmp_community	gp_snmp_use_inform_or_trap
gp_snmp_monitor_address	

Parent topic: System Monitoring Parameters

A newer version of this documentation is available. Use the version menu above to view the most

up-to-date release of the Greenplum 5.x documentation.

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_userid
gp_email_smtp_password	gp_email_to
gp_email_smtp_server	

Parent topic: System Monitoring Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Greenplum Command Center Agent

The following parameters configure the data collection agents that populate the `gpperfmon` database used by Greenplum Command Center.

gp_enable_gpperfmon	gpperfmon_log_alert_level
gp_gpperfmon_send_interval	gpperfmon_port

Parent topic: System Monitoring Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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_counts
track_activities	update_process_title

Parent topic: Configuration Parameter Categories

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Configuration Parameter Categories

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Client Connection Default Parameters

- **Statement Behavior Parameters**
- **Locale and Formatting Parameters**
- **Other Client Default Parameters**

Parent topic: Configuration Parameter Categories

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Statement Behavior Parameters

check_function_bodies	search_path
default_tablespace	statement_timeout
default_transaction_isolation	vacuum_freeze_min_age
default_transaction_read_only	

Parent topic: Client Connection Default Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Locale and Formatting Parameters

client_encoding	lc_messages
DateStyle	lc_monetary
extra_float_digits	lc_numeric
IntervalStyle	lc_time
lc_collate	TimeZone
lc_ctype	

Parent topic: Client Connection Default Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Other Client Default Parameters

dynamic_library_path	local_preload_libraries
explain_pretty_print	

Parent topic: Client Connection Default Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Lock Management Parameters

- deadlock_timeout
- max_locks_per_transaction

Parent topic: Configuration Parameter Categories

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Resource Management Parameters

The following configuration parameters configure Greenplum Database resource queues, query prioritization, memory utilization and concurrency control.

gp_resqueue_priority	max_resource_queues
gp_resqueue_priority_cpucores_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
	vmem_process_interrupt

Parent topic: Configuration Parameter Categories

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

External Table Parameters

The following parameters configure the external tables feature of Greenplum Database.

See Defining External Tables for more information about external tables.

gp_external_enable_exec	readable_external_table_timeout
gp_external_enable_filter_pushdown	writable_external_table_bufsize
gp_external_max_segs	verify_gpfdists_cert
gp_initial_bad_row_limit	
gp_reject_percent_threshold	

Parent topic: Configuration Parameter Categories

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Configuration Parameter Categories

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

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Configuration Parameter Categories

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Past Version Compatibility Parameters

The following parameters provide compatibility with older PostgreSQL and Greenplum Database versions. You do not need to change these parameters in Greenplum Database.

Parent topic: Configuration Parameter Categories

PostgreSQL

add_missing_from	regex_flavor
array_nulls	standard_conforming_strings
backslash_quote	transform_null_equals
escape_string_warning	

Greenplum Database

gp_ignore_error_table

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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**
- **Dispatch Configuration Parameters**
- **Fault Operation Parameters**
- **Distributed Transaction Management Parameters**
- **Read-Only Parameters**

Parent topic: Configuration Parameter Categories

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Interconnect Configuration Parameters

gp_interconnect_fc_method	gp_interconnect_setup_timeout
gp_interconnect_hash_multiplier	gp_interconnect_snd_ queue_depth
gp_interconnect_queue_depth	gp_interconnect_type
	gp_max_packet_size

Note: Greenplum Database supports only the UDPIFC and TCP interconnect types.

Parent topic: Greenplum Array Configuration Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Dispatch Configuration Parameters

gp_cached_segworkers_threshold	gp_segment_connect_timeout
gp_connections_per_thread	gp_set_proc_affinity
gp_enable_direct_dispatch	

Parent topic: Greenplum Array Configuration Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Fault Operation Parameters

gp_set_read_only	gp_fts_probe_threadcount
gp_fts_probe_interval	

Parent topic: Greenplum Array Configuration Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Distributed Transaction Management Parameters

- gp_max_local_distributed_cache

Parent topic: Greenplum Array Configuration Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Read-Only Parameters

- gp_command_count
- gp_content
- gp_dbid
- gp_num_contents_in_cluster
- gp_role
- gp_session_id

Parent topic: Greenplum Array Configuration Parameters

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: Configuration Parameter Categories

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: Configuration Parameter Categories

XML Data Parameters

- `xmlbinary`
- `xmloption`

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

- **Enabling Mirroring in Greenplum Database**
- **Detecting a Failed Segment**
- **Recovering a Failed Segment**
- **Recovering a Failed Master**

Parent topic: Managing a Greenplum System

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 `gpinitsystem` 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 `gpinitsystem` cluster configuration file. See gpinitsystem.

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 `gpinitsystem` 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 datatabases, 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**
- **Overview of Master Mirroring**
- **Overview of Fault Detection and Recovery**

Parent topic: Enabling High Availability and Data Consistency Features

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 Greenplum Database detects that a primary segment has failed or become unavailable, it changes the role of its mirror segment to primary segment and the role of the unavailable primary segment to mirror segment. Transactions in progress when the failure occurred roll back and must be restarted. The administrator must then recover the mirror segment, allow the mirror to syncronize with the current primary segment, and then exchange the primary and mirror segments so they are in their preferred roles.

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 filespaces, which are tablespaces internally represented with directories—also use file replication to perform various file operations in a synchronous way.

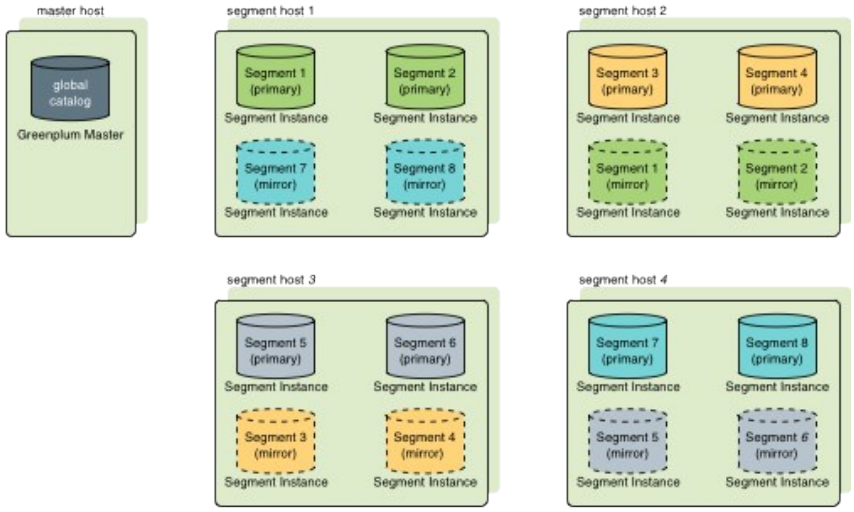
Append-optimized tables do not use the in-memory caching mechanism. Changes made to append-optimized table blocks are replicated to the mirror immediately. Typically, file write operations are asynchronous, while opening, creating, and synchronizing files are "sync-replicated," which means the primary 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.

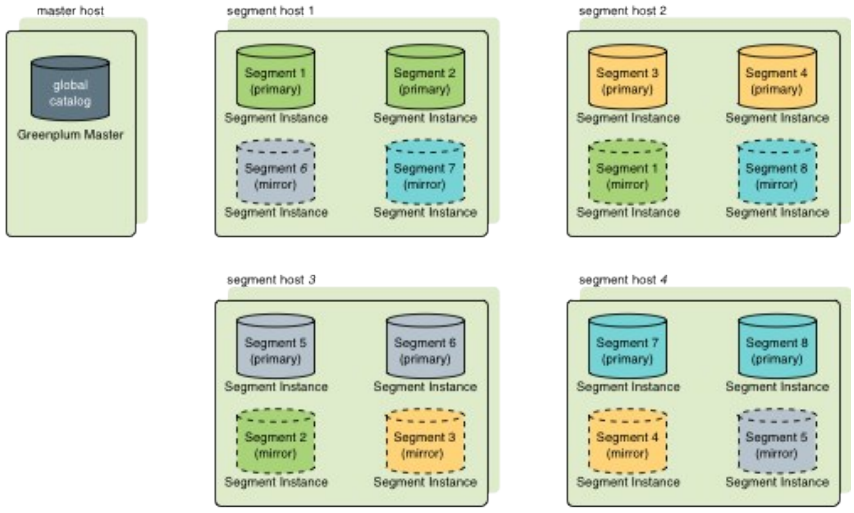
Mirror segments can be placed on hosts in the cluster in different configurations, as long as the primary and mirror instance for a segment are on different hosts. Each host must have the same number of primary and mirror segments. The default mirroring configuration is *group mirroring*, where the mirror 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 1 illustrates a group mirroring configuration.

Figure 1. Group Segment Mirroring in Greenplum Database



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 2 illustrates the placement of mirrors in a spread segment mirroring configuration.

Figure 2. Spread Segment Mirroring in Greenplum Database



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.

Parent topic: Overview of Greenplum Database High Availability

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Overview of Master Mirroring

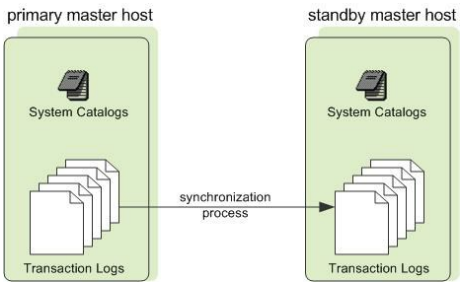
You can deploy a backup or mirror of the master instance on a separate host machine or on the same host machine. A backup master or standby master serves as a warm standby if the primary master becomes nonoperational. You create a standby master from the primary master while the primary is online.

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.

Figure 1. Master Mirroring in Greenplum Database



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.

Parent topic: Overview of Greenplum Database High Availability

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Overview of Greenplum Database High Availability

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Enabling Mirroring in Greenplum Database

You can configure your Greenplum Database system with mirroring at setup time using `gpinitsystem` or enable mirroring later using `gpaddmirrors` and `gpinitstandby`. This topic assumes you are adding mirrors to an existing system that was initialized without mirrors.

- **Enabling Segment Mirroring**
- **Enabling Master Mirroring**

Parent topic: Enabling High Availability and Data Consistency Features

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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:

   ```
   $ 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:

   ```
   $ gpaddmirrors -o filename
   ```

   The format of the mirror configuration file is:

   ```
   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* filespace:

   ```
   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:

   ```
   $ gpaddmirrors -i mirror_config_file
   ```

   Where `-i` names the mirror configuration file you created.

Parent topic: Enabling Mirroring in Greenplum Database

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.
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.

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;'
```

For information about the `pg_stat_replication` system view, see the *Greenplum Database Reference Guide*.

Parent topic: Enabling Mirroring in Greenplum Database

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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**
- **Checking for Failed Segments**
- **Checking the Log Files for Failed Segments**

Parent topic: Enabling High Availability and Data Consistency Features

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.

Parent topic: Detecting a Failed Segment

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:

   ```
   $ 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:

   ```
   $ 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:

   ```
   $ gpstate -m
   ```

Parent topic: Detecting a Failed Segment

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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:

   ```
   $ gplogfilter -t
   ```

2. Use `gpssh` to check for `WARNING`, `ERROR`, `FATAL`, or `PANIC` log level messages on each segment instance. For example:

```
$ gpssh -f seg_hosts_file -e 'source
/usr/local/greenplum-db/greenplum_path.sh ; gplogfilter -t
/data1/primary/*/pg_log/gpdb*.log' > seglog.out
```

Parent topic: Detecting a Failed Segment

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

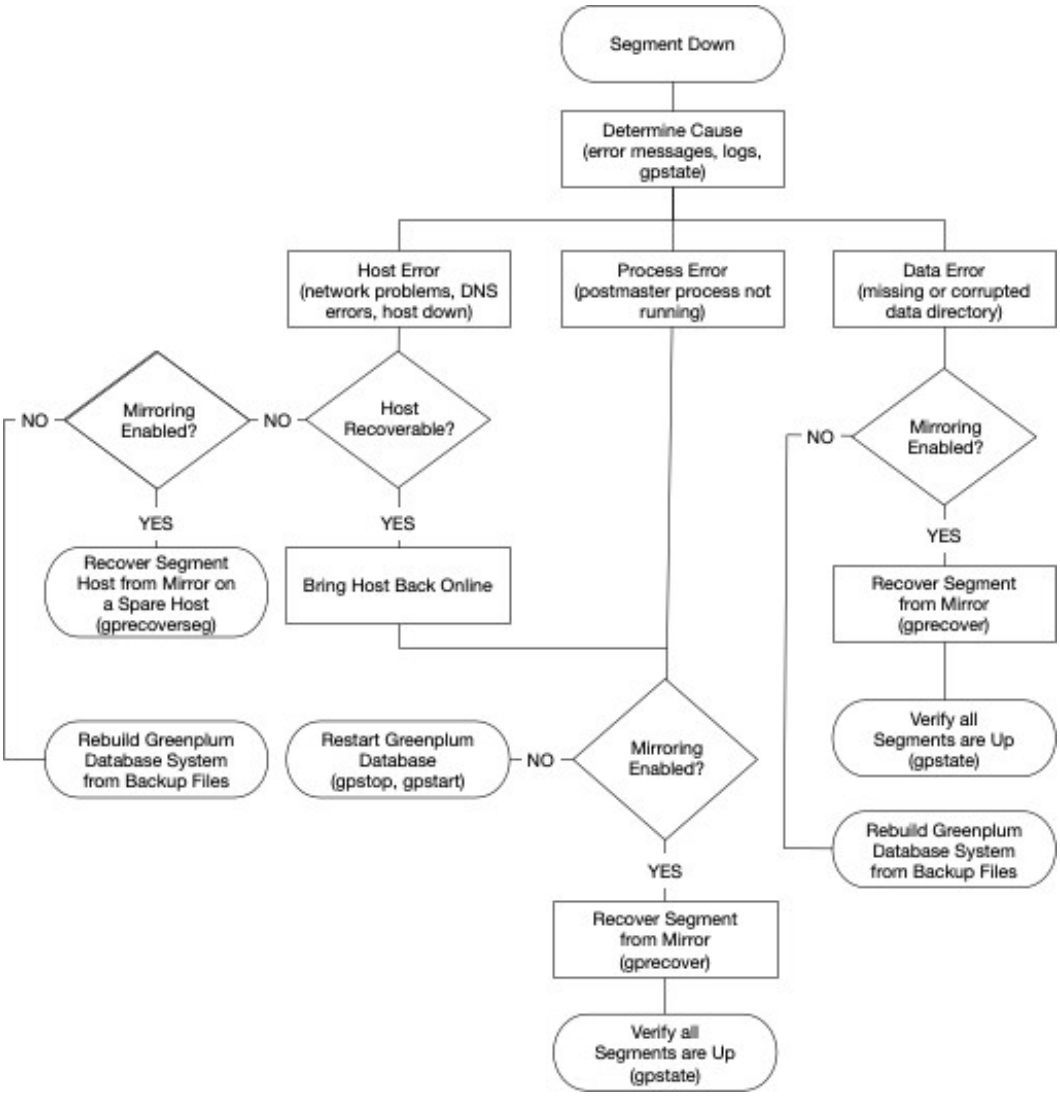
Recovering a Failed Segment

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 1 shows the high-level steps for each of the preceding failure scenarios.

Figure 1. Segment Failure Troubleshooting Matrix



- **Recovering From Segment Failures**

Parent topic: Enabling High Availability and Data Consistency Features

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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**
- **About the Segment Recovery Process**

Parent topic: Recovering a Failed Segment

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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:

```
=# 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.

Parent topic: Recovering From Segment Failures

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

	preferred_role	role	mode	status
Primary	p (primary)	p (primary)	c (change tracking)	u (up)
Mirror	m (mirror)	m (mirror)	s (synchronizing)	d (down)

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.

	preferred_role	role	mode	status
Primary	p (primary)	p (primary)	r (resynchronizing)	u (up)
Mirror	m (mirror)	m (mirror)	r (resynchronizing)	u (up)

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.

	`preferred_role`	`role`	`mode`	`status`
Primary	`p` (primary)	`p` (primary)	`s` (synchronized)	`u` (up)
Mirror	`m` (mirror)	`m` (mirror)	`s` (synchronized)	`u` (up)

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.

Parent topic: Recovering From Segment Failures

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 `gpactivatestandby` 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 `gpactivatestandby` utility from the standby master host you are activating. For example:

   ```
   $ gpactivatestandby -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. 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**

Parent topic: Enabling High Availability and Data Consistency Features

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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
   ```

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;'
```

Parent topic: Recovering a Failed Master

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**
- **Parallel Backup with gpcrondump and gpdbrestore**
- **Parallel Backup with gpbackup and gprestore**

Parent topic: Managing a Greenplum System

Backup and Restore Overview

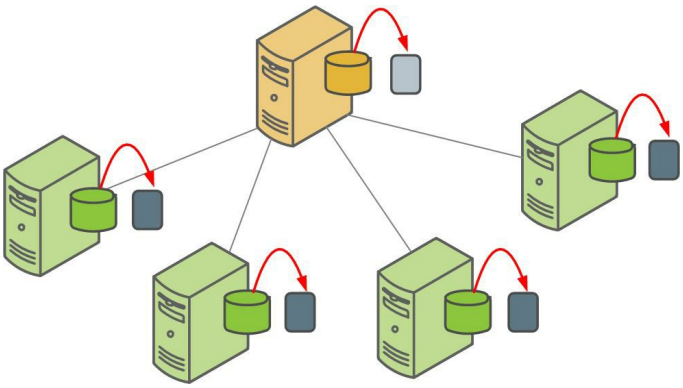
Greenplum Database supports parallel and non-parallel methods for backing up and restoring databases. Parallel operations scale regardless of the number of segments in your system, because segment hosts each write their data to local disk storage simultaneously. With non-parallel backup and restore operations, 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 with gpcrondump and gpdbrestore

The Greenplum Database parallel dump utility `gpcrondump` backs up the Greenplum master instance and each active segment instance at the same time.

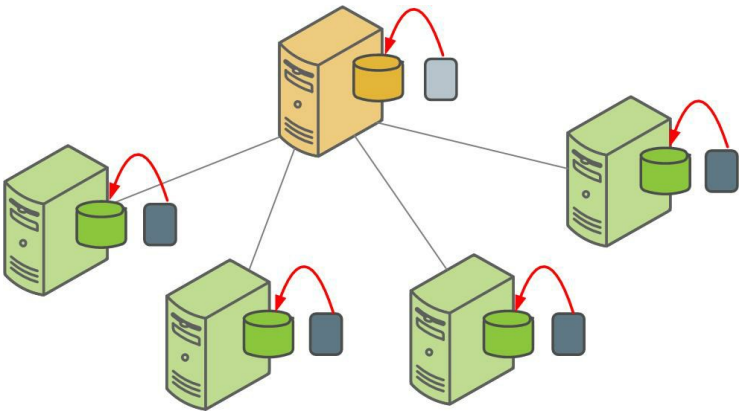
By default, `gpcrondump` creates dump files in the `db_dumps` subdirectory of each segment instance. On the master, `gpcrondump` creates several dump files, containing database information such as DDL statements, the system catalog tables, and metadata files. On each segment, `gpcrondump` creates one 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 1. 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 2. 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.

Parallel Backup with gpbackup and gprestore

`gpbackup` and `gprestore` are new utilities that are designed to improve the performance, functionality, and reliability of backups as compared to `gpcrondump` and `gpdbrestore`. `gpbackup` utilizes `ACCESS SHARE` locks at the individual table level, instead of `EXCLUSIVE` locks on the `pg_class` catalog table. This enables you to execute DML statements during the backup, such as `CREATE`, `ALTER`, `DROP`, and `TRUNCATE` operations, as long as those operations do not target the current backup set.

Backup files created with `gpbackup` are designed to provide future capabilities for restoring individual database objects along with their dependencies, such as functions and required user-defined datatypes. See Parallel Backup with gpbackup and gprestore for more information.

Non-Parallel Backup with pg_dump

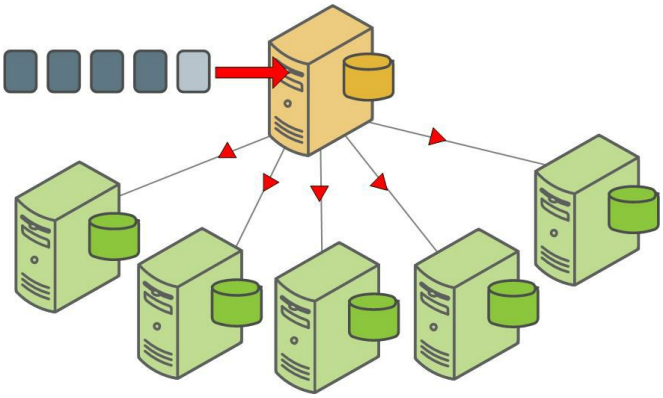
The PostgreSQL `pg_dump` and `pg_dumpall` non-parallel backup utilities can be used to create a single dump file on the master host that contains all data from all active segments.

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 `pg_restore` 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 3. 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.

Parent topic: Backing Up and Restoring Databases

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Parallel Backup with gpcrondump and gpdbrestore

`gpcrondump` backs up the Greenplum master instance and each active segment instance at the same time. `gpdbrestore` takes the timestamp key generated by `gpcrondump`, validates the backup set, and restores database objects and data into a distributed database.

- **Backup and Restore Options**
- **Backing Up with gpcrondump**
- **Backing Up a Set of Tables**
- **Creating Incremental Backups**
- **Backup Process and Locks**
- **Using Direct I/O**
- **Using Named Pipes**
- **Backing Up Databases with Data Domain Boost**
- **Backing Up Databases with Veritas NetBackup**
- **Restoring Greenplum Databases**
- **Restoring a Database Using gpdbrestore**
- **Restoring to a Different Greenplum System Configuration**

Parent topic: Backing Up and Restoring Databases

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Parallel Backup with gpcrondump and gpdbrestore

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:

```
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_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_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.

Parent topic: Parallel Backup with gpcrondump and gpdbrestore

Backing 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.

Parent topic: Parallel Backup with gpcrondump and gpdbrestore

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 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 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 `20160514`*`hhmmss`*. 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:

```
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:

```
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.

```
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`:

```
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:

```
gpcrondump -x mydb --incremental --prefix myschema
```

The following command lists the tables that were included or excluded for the full backup.

```
gpcrondump -x mydb --incremental --prefix myschema --list-filter-tables
```

Parent topic: Parallel Backup with gpcrondump and gpdbrestore

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.

```
$ gpdbrestore -s mydb
...
20151015:20:10:34:002664 gpdbrestore:mdw:gpadmin-[INFO]:------------------------------
-------------
20151015:20:10:34:002664 gpdbrestore:mdw:gpadmin-[INFO]:-Greenplum database restore pa
rameters
20151015:20:10:34:002664 gpdbrestore:mdw:gpadmin-[INFO]:------------------------------
-------------
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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

    - Verifies that sufficient disk space is available to back up 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.

Parent topic: Parallel Backup with gpcrondump and gpdbrestore

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
```

Parent topic: Parallel Backup with gpcrondump and gpdbrestore

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Parallel Backup with gpcrondump and gpdbrestore

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.

```
$ gpcrondump -x testdb -K 20150519160000 --list-backup-files -u /backups
```

2. View the files that `gpcrondump` created in the /backup directory:

```
$ 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_fi
les
```

3. View the contents of the _pipes file.

```
$ 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.

```
gpssh -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 210
00
[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:

```
nc sdw1 21000 > gp_dump_0_2_20150519160000.gz
```

6. Run `gpcrondump` to begin the backup:

```
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:

```
gpdbrestore -x testdb -t 20150519160000 -u /backups
```

`gpdbrestore` reads from the named pipes' outputs.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.
Parent topic: Parallel Backup with gpcrondump and gpdbrestore

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 remove credentials, run `gpcrondump` with the `--ddboost-config-remove` option.

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 `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 `--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:

```
# ddboost option show
```

To enable or disable distributed segment processing:

```
# 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:

   ```
   # 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:

   ```
   # ddboost ifgroup enable
   ```

4. Verify the Data Domain system configuration as follows:

```
# 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 comand 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*.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Parallel Backup with gpcrondump and gpdbrestore

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 DDBoost. Both NetBackup and DDBoost 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 1. NetBackup bp.conf parameters for Greenplum Database

Parameter	Description
`SERVER`	Host name of the NetBackup Master Server
`MEDIA_SERVER`	Host name of the NetBackup Media Server
`CLIENT_NAME`	Host name of the Greenplum Database Host

   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`:

   ```
   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`:

   ```
   export LD_LIBRARY_PATH
   ```
4. Execute this command to remove the current `LD_LIBRARY_PATH` value:

   ```
   unset LD_LIBRARY_PATH
   ```
5. Execute this command to update the environment variables for Greenplum Database:

```
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 back up 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
```

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Parallel Backup with gpcrondump and gpdbrestore

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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` *`YYYYYMMDD`* – 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.

Parent topic: Parallel Backup with gpcrondump and gpdbrestore

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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_201501121403
   16.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
   ```

Parent topic: Parallel Backup with gpcrondump and gpdbrestore

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Parallel Backup with gpbackup and gprestore

`gpbackup` and `gprestore` are Greenplum Database utilities that create and restore backup sets for Greenplum Database. By default, `gpbackup` stores only the object metadata files and DDL files for a backup in the Greenplum Database master data directory. Greenplum Database segments use the `COPY ... ON SEGMENT` command to store their data for backed-up tables in compressed CSV data files, located in each segment's backups directory.

The backup metadata files contain all of the information that `gprestore` needs to restore a full backup set in parallel. Backup metadata also provides the framework for restoring only individual objects in the data set, along with any dependent objects, in future versions of `gprestore`. (See Understanding Backup Files for more information.) Storing the table data in CSV files also provides opportunities for using other restore utilities, such as `gpload`, to load the data either in the same cluster or another cluster. By default, one file is created for each table on the segment. You can specify the `--leaf-partition-data` option with `gpbackup` to create one data file per leaf partition of a partitioned table, instead of a single file. This option also enables you to filter backup sets by leaf partitions.

Each `gpbackup` task uses a single transaction in Greenplum Database. During this transaction, metadata is backed up on the master host, and data for each table on each segment host is written to CSV backup files using `COPY ... ON SEGMENT` commands in parallel. The backup process acquires an `ACCESS SHARE` lock on each table that is backed up.

For information about the `gpbackup` and `gprestore` utility options, see gpbackup and gprestore.

- **Requirements and Limitations**
- **Objects Included in a Backup or Restore**
- **Performing Basic Backup and Restore Operations**
- **Filtering the Contents of a Backup or Restore**
- **Configuring Email Notifications**
- **Understanding Backup Files**
- **Creating Incremental Backups with gpbackup and gprestore**
- **Using gpbackup and gprestore with BoostFS**
- **Using gpbackup Storage Plugins**
- **Backup/Restore Storage Plugin API (Beta)**

Parent topic: Backing Up and Restoring Databases

Requirements and Limitations

The `gpbackup` and `gprestore` utilities are available with Greenplum Database 5.5.0 and later.

`gpbackup` and `gprestore` have the following limitations:

- If you create an index on a parent partitioned table, `gpbackup` does not back up that same index on child partitioned tables of the parent, as creating the same index on a child would cause an error. However, if you exchange a partition, `gpbackup` does not detect that the index on the exchanged partition is inherited from the new parent table. In this case, `gpbackup` backs up conflicting `CREATE INDEX` statements, which causes an error when you restore the backup set.
- You can execute multiple instances of `gpbackup`, but each execution requires a distinct timestamp.
- Database object filtering is currently limited to schemas and tables.

- If you use the `gpbackup --single-data-file` option to combine table backups into a single file per segment, you cannot perform a parallel restore operation with `gprestore` (cannot set `--jobs` to a value higher than 1).
- You cannot use the `--exclude-table-file` with `--leaf-partition-data`. Although you can specify leaf partition names in a file specified with `--exclude-table-file`, `gpbackup` ignores the partition names.
- Backing up a database with `gpbackup` while simultaneously running DDL commands might cause `gpbackup` to fail, in order to ensure consistency within the backup set. For example, if a table is dropped after the start of the backup operation, `gpbackup` exits and displays the error message `ERROR: relation <schema.table> does not exist`.

  `gpbackup` might fail when a table is dropped during a backup operation due to table locking issues. `gpbackup` generates a list of tables to back up and acquires an `ACCESS SHARED` lock on the tables. If an `EXCLUSIVE LOCK` is held on a table, `gpbackup` acquires the `ACCESS SHARED` lock after the existing lock is released. If the table no longer exists when `gpbackup` attempts to acquire a lock on the table, `gpbackup` exits with the error message.

  For tables that might be dropped during a backup, you can exclude the tables from a backup with a `gpbackup` table filtering option such as `--exclude-table` or `--exclude-schema`.

Parent topic: Parallel Backup with gpbackup and gprestore

Objects Included in a Backup or Restore

The following table lists the objects that are backed up and restored with `gpbackup` and `gprestore`. Database objects are backed up for the database you specify with the `--dbname` option. Global objects (Greenplum Database system objects) are also backed up by default, but they are restored only if you include the `--with-globals` option to `gprestore`.

Table 1. Objects that are backed up and restored

Database (for database specified with `--dbname`)	**Global (requires the `--with-globals` option to restore)**

• Session-level configuration parameter settings (GUCs) • Schemas, see Note • Procedural language extensions • Sequences • Comments • Tables • Indexes • Owners • Writable External Tables (DDL only) • Readable External Tables (DDL only) • Functions • Aggregates • Casts • Types • Views • Protocols • Triggers. (While Greenplum Database does not support triggers, any trigger definitions that are present are backed up and restored.) • Rules • Domains • Operators, operator families, and operator classes • Conversions • Extensions • Text search parsers, dictionaries, templates, and configurations	• Tablespaces • Databases • Database-wide configuration parameter settings (GUCs) • Resource group definitions • Resource queue definitions • Roles • `GRANT` assignments of roles to databases

Note: These schemas are not included in a backup.

- `gp_toolkit`
- `information_schema`
- `pg_aoseg`
- `pg_bitmapindex`
- `pg_catalog`
- `pg_toast*`
- `pg_temp*`

When restoring to an existing database, `gprestore` assumes the `public` schema exists when restoring objects to the `public` schema. When restoring to a new database (with the `--create-db` option), `gprestore` creates the `public` schema automatically when creating a database with the `CREATE DATABASE` command. The command uses the `template0` database that contains the `public` schema.

See also Understanding Backup Files.

Parent topic: Parallel Backup with gpbackup and gprestore

Performing Basic Backup and Restore Operations

To perform a complete backup of a database, as well as Greenplum Database system metadata, use the command:

```
$ gpbackup --dbname <database_name>
```

For example:

```
$ gpbackup --dbname demo
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Starting backup
of database demo
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Backup Timestamp
 = 20180105112754
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Backup Database
= demo
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Backup Type = Un
filtered Compressed Full Backup
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Gathering list o
f tables for backup
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Acquiring ACCESS
 SHARE locks on tables
Locks acquired:  6 / 6 [================================================================
==] 100.00% 0s
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Gathering additi
onal table metadata
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Writing global d
atabase metadata
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Global database
metadata backup complete
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Writing pre-data
 metadata
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Pre-data metadat
a backup complete
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Writing post-dat
a metadata
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Post-data metada
ta backup complete
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Writing data to
file
Tables backed up:  3 / 3 [================================================================
==] 100.00% 0s
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Data backup comp
lete
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Found neither /u
sr/local/greenplum-db/./bin/gp_email_contacts.yaml nor /home/gpadmin/gp_email_contacts
.yaml
20180105:11:27:54 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Email containing
 gpbackup report /gpmaster/seg-1/backups/20180105/20180105112754/gpbackup_201801051127
54_report will not be sent
20180105:11:27:55 gpbackup:gpadmin:centos6.localdomain:002182-[INFO]:-Backup completed
 successfully
```

The above command creates a file that contains global and database-specific metadata on the Greenplum Database master host in the default directory, `$MASTER_DATA_DIRECTORY/backups/<YYYYMMDD>/<YYYYMMDDHHMMSS>/`. For example:

```
$ ls /gpmaster/gpsne-1/backups/20180105/20180105112754
gpbackup_20180105112754_config.yaml   gpbackup_20180105112754_report
gpbackup_20180105112754_metadata.sql  gpbackup_20180105112754_toc.yaml
```

By default, each segment stores each table's data for the backup in a separate compressed CSV file in `<seg_dir>/backups/<YYYYMMDD>/<YYYYMMDDHHMMSS>/`:

```
$ ls /gpdata1/gpsne0/backups/20180105/20180105112754/
gpbackup_0_20180105112754_17166.gz  gpbackup_0_20180105112754_26303.gz
gpbackup_0_20180105112754_21816.gz
```

To consolidate all backup files into a single directory, include the `--backup-dir` option. Note that you must specify an absolute path with this option:

```
$ gpbackup --dbname demo --backup-dir /home/gpadmin/backups
20171103:15:31:56 gpbackup:gpadmin:0ee2f5fb02c9:017586-[INFO]:-Starting backup of data
base demo
...
20171103:15:31:58 gpbackup:gpadmin:0ee2f5fb02c9:017586-[INFO]:-Backup completed succes
sfully
$ find /home/gpadmin/backups/ -type f
/home/gpadmin/backups/gpseg0/backups/20171103/20171103153156/gpbackup_0_20171103153156
_16543.gz
/home/gpadmin/backups/gpseg0/backups/20171103/20171103153156/gpbackup_0_20171103153156
_16524.gz
/home/gpadmin/backups/gpseg1/backups/20171103/20171103153156/gpbackup_1_20171103153156
_16543.gz
/home/gpadmin/backups/gpseg1/backups/20171103/20171103153156/gpbackup_1_20171103153156
_16524.gz
/home/gpadmin/backups/gpseg-1/backups/20171103/20171103153156/gpbackup_20171103153156_
config.yaml
/home/gpadmin/backups/gpseg-1/backups/20171103/20171103153156/gpbackup_20171103153156_
predata.sql
/home/gpadmin/backups/gpseg-1/backups/20171103/20171103153156/gpbackup_20171103153156_
global.sql
/home/gpadmin/backups/gpseg-1/backups/20171103/20171103153156/gpbackup_20171103153156_
postdata.sql
/home/gpadmin/backups/gpseg-1/backups/20171103/20171103153156/gpbackup_20171103153156_
report
/home/gpadmin/backups/gpseg-1/backups/20171103/20171103153156/gpbackup_20171103153156_
toc.yaml
```

When performing a backup operation, you can use the `--single-data-file` in situations where the additional overhead of multiple files might be prohibitive. For example, if you use a third party storage solution such as Data Domain with back ups.

Restoring from Backup

To use `gprestore` to restore from a backup set, you must use the `--timestamp` option to specify the exact timestamp value (`YYYYMMDDHHMMSS`) to restore. Include the `--create-db` option if the database does not exist in the cluster. For example:

```
$ dropdb demo
$ gprestore --timestamp 20171103152558 --create-db
20171103:15:45:30 gprestore:gpadmin:0ee2f5fb02c9:017714-[INFO]:-Restore Key = 20171103
152558
20171103:15:45:31 gprestore:gpadmin:0ee2f5fb02c9:017714-[INFO]:-Creating database
20171103:15:45:44 gprestore:gpadmin:0ee2f5fb02c9:017714-[INFO]:-Database creation comp
lete
20171103:15:45:44 gprestore:gpadmin:0ee2f5fb02c9:017714-[INFO]:-Restoring pre-data met
adata from /gpmaster/gpsne-1/backups/20171103/20171103152558/gpbackup_20171103152558_p
redata.sql
20171103:15:45:45 gprestore:gpadmin:0ee2f5fb02c9:017714-[INFO]:-Pre-data metadata rest
ore complete
20171103:15:45:45 gprestore:gpadmin:0ee2f5fb02c9:017714-[INFO]:-Restoring data
20171103:15:45:45 gprestore:gpadmin:0ee2f5fb02c9:017714-[INFO]:-Data restore complete
20171103:15:45:45 gprestore:gpadmin:0ee2f5fb02c9:017714-[INFO]:-Restoring post-data me
tadata from /gpmaster/gpsne-1/backups/20171103/20171103152558/gpbackup_20171103152558_
postdata.sql
20171103:15:45:45 gprestore:gpadmin:0ee2f5fb02c9:017714-[INFO]:-Post-data metadata res
tore complete
```

If you specified a custom `--backup-dir` to consolidate the backup files, include the same `--backup-dir` option when using `gprestore` to locate the backup files:

```
$ dropdb demo
$ gprestore --backup-dir /home/gpadmin/backups/ --timestamp 20171103153156 --create-db
20171103:15:51:02 gprestore:gpadmin:0ee2f5fb02c9:017819-[INFO]:-Restore Key = 20171103
153156
...
20171103:15:51:17 gprestore:gpadmin:0ee2f5fb02c9:017819-[INFO]:-Post-data metadata res
```

```
tore complete
```

`gprestore` does not attempt to restore global metadata for the Greenplum System by default. If this is required, include the `--with-globals` argument.

By default, `gprestore` uses 1 connection to restore table data and metadata. If you have a large backup set, you can improve performance of the restore by increasing the number of parallel connections with the `--jobs` option. For example:

```
$ gprestore --backup-dir /home/gpadmin/backups/ --timestamp 20171103153156 --create-db --jobs 8
```

Test the number of parallel connections with your backup set to determine the ideal number for fast data recovery.

Note: You cannot perform a parallel restore operation with `gprestore` if the backup combined table backups into a single file per segment with the `gpbackup` option `--single-data-file`.

Report Files

When performing a backup or restore operation, `gpbackup` and `gprestore` generate a report file. When email notification is configured, the email sent contains the contents of the report file. For information about email notification, see Configuring Email Notifications.

The report file is placed in the Greenplum Database master backup directory. The report file name contains the timestamp of the operation. These are the formats of the `gpbackup` and `gprestore` report file names.

```
gpbackup_<backup_timestamp>_report
gprestore_<backup_timestamp>_<restore_timesamp>_report
```

For these example report file names, `20180213114446` is the timestamp of the backup and `20180213115426` is the timestamp of the restore operation.

```
gpbackup_20180213114446_report
gprestore_20180213114446_20180213115426_report
```

This backup directory on a Greenplum Database master host contains both a `gpbackup` and `gprestore` report file.

```
$ ls -l /gpmaster/seg-1/backups/20180213/20180213114446
total 36
-r--r--r--. 1 gpadmin gpadmin  295 Feb 13 11:44 gpbackup_20180213114446_config.yaml
-r--r--r--. 1 gpadmin gpadmin 1855 Feb 13 11:44 gpbackup_20180213114446_metadata.sql
-r--r--r--. 1 gpadmin gpadmin 1402 Feb 13 11:44 gpbackup_20180213114446_report
-r--r--r--. 1 gpadmin gpadmin 2199 Feb 13 11:44 gpbackup_20180213114446_toc.yaml
-r--r--r--. 1 gpadmin gpadmin  404 Feb 13 11:54 gprestore_20180213114446_2018021311542
6_report
```

The contents of the report files are similar. This is an example of the contents of a `gprestore` report file.

```
Greenplum Database Restore Report

Timestamp Key: 20180213114446
GPDB Version: 5.4.1+dev.8.g9f83645 build commit:9f836456b00f855959d52749d5790ed1c6efc0
42
gprestore Version: 1.0.0-alpha.3+dev.73.g0406681

Database Name: test
Command Line: gprestore --timestamp 20180213114446 --with-globals --createdb

Start Time: 2018-02-13 11:54:26
End Time: 2018-02-13 11:54:31
Duration: 0:00:05
```

```
Restore Status: Success
```

History File

When performing a backup operation, `gpbackup` appends backup information in the gpbackup history file, `gpbackup_history.yaml`, in the Greenplum Database master data directory. The file contains the backup timestamp, information about the backup options, and backup set information for incremental backups. This file is not backed up by `gpbackup`.

`gpbackup` uses the information in the file to find a matching backup for an incremental backup when you run `gpbackup` with the `--incremental` option and do not specify the `--from-timesamp` option to indicate the backup that you want to use as the latest backup in the incremental backup set. For information about incremental backups, see Creating Incremental Backups with gpbackup and gprestore.

Return Codes

One of these codes is returned after `gpbackup` or `gprestore` completes.

- **0** – Backup or restore completed with no problems
- **1** – Backup or restore completed with non-fatal errors. See log file for more information.
- **2** – Backup or restore failed with a fatal error. See log file for more information.

Parent topic: Parallel Backup with gpbackup and gprestore

Filtering the Contents of a Backup or Restore

`gpbackup` backs up all schemas and tables in the specified database, unless you exclude or include individual schema or table objects with schema level or table level filter options.

The schema level options are `--include-schema` or `--exclude-schema` command-line options to `gpbackup`. For example, if the "demo" database includes only two schemas, "wikipedia" and "twitter," both of the following commands back up only the "wikipedia" schema:

```
$ gpbackup --dbname demo --include-schema wikipedia
$ gpbackup --dbname demo --exclude-schema twitter
```

You can include multiple `--include-schema` options in a `gpbackup` *or* multiple `--exclude-schema` options. For example:

```
$ gpbackup --dbname demo --include-schema wikipedia --include-schema twitter
```

To filter the individual tables that are included in a backup set, or excluded from a backup set, specify individual tables with the `--include-table` option or the `--exclude-table` option. The table must be schema qualified, `<schema-name>.<table-name>`. The individual table filtering options can be specified multiple times. However, `--include-table` and `--exclude-table` cannot both be used in the same command.

You can create a list of qualified table names in a text file. When listing tables in a file, each line in the text file must define a single table using the format `<schema-name>.<table-name>`. The file must not include trailing lines. For example:

```
wikipedia.articles
twitter.message
```

If a table or schema name uses any character other than a lowercase letter, number, or an underscore character, then you must include that name in double quotes. For example:

```
beer."IPA"
"Wine".riesling
"Wine"."sauvignon blanc"
```

```
water.tonic
```

After creating the file, you can use it either to include or exclude tables with the `gpbackup` options `--include-table-file` or `--exclude-table-file`. For example:

```
$ gpbackup --dbname demo --include-table-file /home/gpadmin/table-list.txt
```

You can combine `--include schema` with `--exclude-table` or `--exclude-table-file` for a backup. This example uses `--include-schema` with `--exclude-table` to back up a schema except for a single table.

```
$ gpbackup --dbname demo --include-schema mydata --exclude-table mydata.addresses
```

You cannot combine `--include-schema` with `--include-table` or `--include-table-file`, and you cannot combine `--exclude-schema` with any table filtering option such as `--exclude-table` or `--include-table`.

When you use `--include-table` or `--include-table-file` dependent objects are not automatically backed up or restored, you must explicitly specify the dependent objects that are required. For example, if you back up or restore a view, you must also specify the tables that the view uses. If you backup or restore a table that uses a sequence, you must also specify the sequence.

Filtering by Leaf Partition

By default, `gpbackup` creates one file for each table on a segment. You can specify the `--leaf-partition-data` option to create one data file per leaf partition of a partitioned table, instead of a single file. You can also filter backups to specific leaf partitions by listing the leaf partition names in a text file to include. For example, consider a table that was created using the statement:

```
demo=# CREATE TABLE sales (id int, date date, amt decimal(10,2))
DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
( PARTITION Jan17 START (date '2017-01-01') INCLUSIVE ,
PARTITION Feb17 START (date '2017-02-01') INCLUSIVE ,
PARTITION Mar17 START (date '2017-03-01') INCLUSIVE ,
PARTITION Apr17 START (date '2017-04-01') INCLUSIVE ,
PARTITION May17 START (date '2017-05-01') INCLUSIVE ,
PARTITION Jun17 START (date '2017-06-01') INCLUSIVE ,
PARTITION Jul17 START (date '2017-07-01') INCLUSIVE ,
PARTITION Aug17 START (date '2017-08-01') INCLUSIVE ,
PARTITION Sep17 START (date '2017-09-01') INCLUSIVE ,
PARTITION Oct17 START (date '2017-10-01') INCLUSIVE ,
PARTITION Nov17 START (date '2017-11-01') INCLUSIVE ,
PARTITION Dec17 START (date '2017-12-01') INCLUSIVE
END (date '2018-01-01') EXCLUSIVE );
NOTICE:  CREATE TABLE will create partition "sales_1_prt_jan17" for table "sales"
NOTICE:  CREATE TABLE will create partition "sales_1_prt_feb17" for table "sales"
NOTICE:  CREATE TABLE will create partition "sales_1_prt_mar17" for table "sales"
NOTICE:  CREATE TABLE will create partition "sales_1_prt_apr17" for table "sales"
NOTICE:  CREATE TABLE will create partition "sales_1_prt_may17" for table "sales"
NOTICE:  CREATE TABLE will create partition "sales_1_prt_jun17" for table "sales"
NOTICE:  CREATE TABLE will create partition "sales_1_prt_jul17" for table "sales"
NOTICE:  CREATE TABLE will create partition "sales_1_prt_aug17" for table "sales"
NOTICE:  CREATE TABLE will create partition "sales_1_prt_sep17" for table "sales"
NOTICE:  CREATE TABLE will create partition "sales_1_prt_oct17" for table "sales"
NOTICE:  CREATE TABLE will create partition "sales_1_prt_nov17" for table "sales"
NOTICE:  CREATE TABLE will create partition "sales_1_prt_dec17" for table "sales"
CREATE TABLE
```

To back up only data for the last quarter of the year, first create a text file that lists those leaf partition names instead of the full table name:

```
public.sales_1_prt_oct17
```

```
public.sales_1_prt_nov17
public.sales_1_prt_dec17
```

Then specify the file with the `--include-table-file` option to generate one data file per leaf partition:

```
$ gpbackup --dbname demo --include-table-file last-quarter.txt --leaf-partition-data
```

When you specify `--leaf-partition-data`, `gpbackup` generates one data file per leaf partition when backing up a partitioned table. For example, this command generates one data file for each leaf partition:

```
$ gpbackup --dbname demo --include-table public.sales --leaf-partition-data
```

When leaf partitions are backed up, the leaf partition data is backed up along with the metadata for the entire partitioned table.

Note: You cannot use the `--exclude-table-file` option with `--leaf-partition-data`. Although you can specify leaf partition names in a file specified with `--exclude-table-file`, `gpbackup` ignores the partition names.

Filtering with gprestore

After creating a backup set with `gpbackup`, you can filter the schemas and tables that you want to restore from the backup set using the `gprestore --include-schema` and `--include-table-file` options. These options work in the same way as their `gpbackup` counterparts, but have the following restrictions:

- The tables that you attempt to restore must not already exist in the database.
- If you attempt to restore a schema or table that does not exist in the backup set, the `gprestore` does not execute.
- If you use the `--include-schema` option, `gprestore` cannot restore objects that have dependencies on multiple schemas.
- If you use the `--include-table-file` option, `gprestore` does not create roles or set the owner of the tables. The utility restores table indexes and rules. Triggers are also restored but are not supported in Greenplum Database.
- The file that you specify with `--include-table-file` cannot include a leaf partition name, as it can when you specify this option with `gpbackup`. If you specified leaf partitions in the backup set, specify the partitioned table to restore the leaf partition data.

  When restoring a backup set that contains data from some leaf partitions of a partitioned table, the partitioned table is restored along with the data for the leaf partitions. For example, you create a backup with the `gpbackup` option `--include-table-file` and the text file lists some leaf partitions of a partitioned table. Restoring the backup creates the partitioned table and restores the data only for the leaf partitions listed in the file.

Parent topic: Parallel Backup with gpbackup and gprestore

Configuring Email Notifications

`gpbackup` and `gprestore` can send email notifications after a back up or restore operation completes.

To have `gpbackup` or `gprestore` send out status email notifications, you must place a file named `gp_email_contacts.yaml` in the home directory of the user running `gpbackup` or `gprestore` in the same directory as the utilities (`$GPHOME/bin`). A utility issues a message if it cannot locate a `gp_email_contacts.yaml` file in either location. If both locations contain a `.yaml` file, the utility uses the file in user `$HOME`.

The email subject line includes the utility name, timestamp, status, and the name of the Greenplum Database master. This is an example subject line for a `gpbackup` email.

```
gpbackup 20180202133601 on gp-master completed
```

The email contains summary information about the operation including options, duration, and number of objects backed up or restored. For information about the contents of a notification email, see Report Files.

Note: The UNIX mail utility must be running on the Greenplum Database host and must be configured to allow the Greenplum superuser (`gpadmin`) to send email. Also ensure that the mail program executable is locatable via the `gpadmin` user's `$PATH`.

Parent topic: Parallel Backup with gpbackup and gprestore

gpbackup and gprestore Email File Format

The `gpbackup` and `gprestore` email notification YAML file `gp_email_contacts.yaml` 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.

Note: If the `status` parameters are not specified correctly, the utility does not issue a warning. For example, if the `success` parameter is misspelled and is set to `true`, a warning is not issued and an email is not sent to the email address after a successful operation. To ensure email notification is configured correctly, run tests with email notifications configured.

This is the format of the `gp_email_contacts.yaml` YAML file for `gpbackup` email notifications:

```
contacts:
  gpbackup:
  - address: user@domain
    status:
         success: [true | false]
         success_with_errors: [true | false]
         failure: [true | false]
  gprestore:
  - address: user@domain
    status:
         success: [true | false]
         success_with_errors: [true | false]
         failure: [true | false]
```

Email YAML File Sections

contacts

Required. The section that contains the `gpbackup` and `gprestore` sections. The YAML file can contain a `gpbackup` section, a `gprestore` section, or one of each.

gpbackup

Optional. Begins the `gpbackup` email section.

address

Required. At least one email address must be specified. Multiple email `address` parameters can be specified. Each `address` requires a `status` section.
user@domain is a single, valid email address.

status

Required. Specify when the utility sends an email to the specified email address. The default is to not send email notification.
You specify sending email notifications based on the completion status of a backup or restore operation. At least one of these parameters must be specified and each parameter can appear at most once.

success

Optional. Specify if an email is sent if the operation completes without errors. If the value is `true`, an email is sent if the operation completes without errors. If the value is `false` (the default), an email is not sent.

success_with_errors

Optional. Specify if an email is sent if the operation completes with errors. If the value is `true`, an email is sent if the operation completes with errors. If the value is `false` (the default), an email is not sent.

failure

Optional. Specify if an email is sent if the operation fails. If the value is `true`, an email is sent if the operation fails. If the value is `false` (the default), an email is not sent.

gprestore

Optional. Begins the `gprestore` email section. This section contains the `address` and `status` parameters that are used to send an email notification after a `gprestore` operation. The syntax is the same as the `gpbackup` section.

Examples

This example YAML file specifies sending email to email addresses depending on the success or failure of an operation. For a backup operation, an email is sent to a different address depending on the success or failure of the backup operation. For a restore operation, an email is sent to `gpadmin@example.com` only when the operation succeeds or completes with errors.

```
contacts:
  gpbackup:
  - address: gpadmin@example.com
    status:
      success:true
  - address: my_dba@example.com
    status:
      success_with_errors: true
      failure: true
  gprestore:
  - address: gpadmin@example.com
    status:
      success: true
      success_with_errors: true
```

Understanding Backup Files

Warning: All `gpbackup` metadata files are created with read-only permissions. Never delete or modify the metadata files for a `gpbackup` backup set. Doing so will render the backup files non-functional.

A complete backup set for `gpbackup` includes multiple metadata files, supporting files, and CSV data files, each designated with the timestamp at which the backup was created.

By default, metadata and supporting files are stored on the Greenplum Database master host in the directory $MASTER_DATA_DIRECTORY/backups/YYYYMMDD/YYYYMMDDHHMMSS/. If you specify a custom backup directory, this same file path is created as a subdirectory of the backup directory. The following table describes the names and contents of the metadata and supporting files.

Table 2. gpbackup Metadata Files (master)

File name	Description

gpbackup_<YYYYMMDDHHMMSS>_metadata.sql	Contains global and database-specific metadata: • DDL for objects that are global to the Greenplum Database cluster, and not owned by a specific database within the cluster. • DDL for objects in the backed-up database (specified with `--dbname`) that must be created *before* to restoring the actual data, and DDL for objects that must be created *after* restoring the data. Global objects include: • Tablespaces • Databases • Database-wide configuration parameter settings (GUCs) • Resource group definitions • Resource queue definitions • Roles • `GRANT` assignments of roles to databases **Note:** Global metadata is not restored by default. You must include the `--with-globals` option to the `gprestore` command to restore global metadata. Database-specific objects that must be created *before* to restoring the actual data include: • Session-level configuration parameter settings (GUCs) • Schemas • Procedural language extensions • Types • Sequences • Functions • Tables • Protocols • Operators and operator classes • Conversions • Aggregates • Casts • Views • Constraints Database-specific objects that must be created *after* restoring the actual data include: • Indexes • Rules • Triggers. (While Greenplum Database does not support triggers, any trigger definitions that are present are backed up and restored.)
gpbackup_<YYYYMMDDHHMMSS>_toc.yaml	Contains metadata for locating object DDL in the _predata.sql and _postdata.sql files. This file also contains the table names and OIDs used for locating the corresponding table data in CSV data files that are created on each segment. See Segment Data Files.

Table 2. gpbackup Metadata Files (master)

File name	Description
gpbackup_<YYYYMMDDHHMMSS>_report	Contains information about the backup operation that is used to populate the email notice (if configured) that is sent after the backup completes. This file contains information such as: • Command-line options that were provided • Database that was backed up • Database version • Backup type See Configuring Email Notifications.
gpbackup_<YYYYMMDDHHMMSS>_config.yaml	Contains metadata about the execution of the particular backup task, including: • `gpbackup` version • Database name • Greenplum Database version • Additional option settings such as `--no-compression`, `--compression-level`, `--metadata-only`, `--data-only`, and `--with-stats`.
gpbackup_history.yaml	Contains information about options that were used when creating a backup with `gpbackup`, and information about incremental backups. Stored on the Greenplum Database master host in the Greenplum Database master data directory. This file is not backed up by `gpbackup`. For information about incremental backups, see Creating Incremental Backups with gpbackup and gprestore.

Segment Data Files

By default, each segment creates one compressed CSV file for each table that is backed up on the segment. You can optionally specify the `--single-data-file` option to create a single data file on each segment. The files are stored in <seg_dir>/backups/YYYYMMDD/YYYYMMDDHHMMSS/.

If you specify a custom backup directory, segment data files are copied to this same file path as a subdirectory of the backup directory. If you include the `--leaf-partition-data` option, `gpbackup` creates one data file for each leaf partition of a partitioned table, instead of just one table for file.

Each data file uses the file name format gpbackup_<content_id>_<YYYYMMDDHHMMSS>_<oid>.gz where:

- <content_id> is the content ID of the segment.
- <YYYYMMDDHHMMSS> is the timestamp of the `gpbackup` operation.
- <oid> is the object ID of the table. The metadata file gpbackup_<YYYYMMDDHHMMSS>_toc.yaml references this <oid> to locate the data for a specific table in a schema.

You can optionally specify the gzip compression level (from 1-9) using the `--compression-level` option, or disable compression entirely with `--no-compression`. If you do not specify a compression level, `gpbackup` uses compression level 1 by default.

Parent topic: Parallel Backup with gpbackup and gprestore

Expanding a Greenplum System

To scale up performance and storage capacity, expand your Greenplum Database system by adding hosts to the system. In general, adding nodes to a Greenplum cluster achieves a linear scaling of performance and storage capacity.

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**
  You can perform a Greenplum Database expansion to add segment instances and segment hosts with minimal downtime. In general, adding nodes to a Greenplum cluster achieves a linear scaling of performance and storage capacity.
- **Planning Greenplum System Expansion**
  Careful planning will help to ensure a successful Greenplum expansion project.
- **Preparing and Adding Nodes**
  Verify your new nodes are ready for integration into the existing Greenplum system.
- **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.
- **Redistributing Tables**
  Redistribute tables to balance existing data over the newly expanded cluster.
- **Removing the Expansion Schema**
  To clean up after expanding the Greenplum cluster, remove the expansion schema.

Parent topic: Managing a Greenplum System

System Expansion Overview

You can perform a Greenplum Database expansion to add segment instances and segment hosts with minimal downtime. In general, adding nodes to a Greenplum cluster achieves a linear scaling of performance and storage capacity.

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 adminstrators 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:

    ```
    gpexpand -f hosts_file
    ```

2. To initialize segments and create the expansion schema:

    ```
    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:

    ```
    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:

```
gpexpand -c
```

For information about the `gpexpand` utility and the other utilities that are used for system expansion, see the *Greenplum Database Utility Guide*.

Parent topic: Expanding a Greenplum System

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.

Parent topic: Expanding a Greenplum System

System Expansion Checklist

This checklist summarizes the tasks for a Greenplum Database system expansion.

Table 1. Greenplum Database System Expansion Checklist

Online Pre-Expansion Tasks * System is up and available	
☐	Devise and execute a plan for ordering, building, and networking new hardware platforms, or provisioning cloud resources.
☐	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.
☐	Perform a complete schema dump.
☐	Install Greenplum Database binaries on new hosts.
☐	Copy SSH keys to the new hosts (`gpssh-exkeys`).
☐	Validate the operating system environment of the new hardware or cloud resources (`gpcheck`).

☐	Validate disk I/O and memory bandwidth of the new hardware or cloud resources (`gpcheckperf`).
☐	Validate that the master data directory has no extremely large files in the `pg_log` or `gpperfmon/data` directories.
☐	Validate that there are no catalog issues (`gpcheckcat`).
☐	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`).
☐	**Important:** 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 deploying 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 Adding New Nodes to the Trusted Host Environment.

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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Expanding a Greenplum System

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
gpadmin --stdin'
```

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

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Expanding a Greenplum System

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
sdw5:sdw5-2:60011:/gpdata/mirror/gp9:13:9:m:63011
sdw5:sdw5-1:60012:/gpdata/mirror/gp10:14:10:m:63011
```

For each new segment, this format of expansion input file requires the following:

Table 1. Data for the expansion configuration file

Parameter	Valid Values	Description
hostname	Hostname	Hostname for the segment host.
port	An available port number	Database listener port for the segment, incremented on the existing segment *port* base number.
fselocation	Directory name	The data directory (filespace) location for a segment as per the pg_filespace_entry system catalog.
dbid	Integer. Must not conflict with existing *dbid* values.	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.
content	Integer. Must not conflict with existing *content* values.	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`.
preferred_role	`p | m`	Determines whether this segment is a primary or mirror. Specify `p` for primary and `m` for mirror.
replication_port	An available port number	File replication port for the segment, incremented on the existing segment *replication_port* base number.

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.

After segment initialization is complete, redistribute the tables to balance existing data over the new segments.

Monitoring the Cluster Expansion State

At any time, you can check the state of cluster expansion by running the `gpstate` utility with the `-x` flag:

```
$ gpstate -x
```

If the expansion schema exists in the postgres database, `gpstate -x` reports on the progress of the expansion. During the first expansion phase, `gpstate` reports on the progress of new segment initialization. During the second phase, `gpstate` reports on the progress of table redistribution, and whether redistribution is paused or active.

You can also query the expansion schema to see expansion status. See Monitoring Table Redistribution for more information.

Rolling Back a Failed Expansion Setup

You can roll back an expansion setup operation (adding segment instances and segment hosts) 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:

```
gpexpand --rollback
```

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Expanding a Greenplum System

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:

```
=> 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.

Note: After completing table redistribution, run the `VACUUM ANALYZE` and `REINDEX`commands on the catalog tables to update table statistics, and rebuild indexes. See Routine Vacuum and Analyze in the *Administration Guide* and `VACUUM` in the *Reference Guide*.

Monitoring Table Redistribution

During the table redistribution process you can query the expansion schema to view:

- a current progress summary, the estimated rate of table redistribution, and the estimated time to completion. Use *gpexpand.expansion_progress*, as described in Viewing Expansion Status.
- per-table status information, using *gpexpand.status_detail*. See Viewing Table Status.

See also Monitoring the Cluster Expansion State for information about monitoring the overall expansion progress with the `gpstate` utility.

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:

```
=# SELECT * FROM gpexpand.expansion_progress;
             name             |         value
------------------------------+----------------------
 Bytes Left                   | 5534842880
 Bytes Done                   | 142475264
 Estimated Expansion Rate     | 680.75667095996092 MB/s
 Estimated Time to Completion | 00:01:01.008047
 Tables Expanded              | 4
 Tables Left                  | 4
(6 rows)
```

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*:

```
=> SELECT status, expansion_started, source_bytes FROM
gpexpand.status_detail WHERE fq_name = 'public.sales';
  status    |     expansion_started      | source_bytes
------------+----------------------------+--------------
 COMPLETED | 2017-02-20 10:54:10.043869 |   4929748992
(1 row)
```


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:

   ```
   $ gpexpand -c
   $
   ```

   Note: Some systems require you to press Enter twice.

Parent topic: Expanding a Greenplum System

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Migrating Data

Greenplum Database provides utilities that you can use to migrate data between Greenplum Database systems.

The gpcopy utility can migrate data between Greenplum Database systems running software version 5.9.0 or later. The utility can require less disk space to migrate between Greenplum Database systems that co-exist on the same hosts.

The gpransfer utility can migrate data between Greenplum Database installations running any 5.x version software.
Note: gptransfer is deprecated and will be removed in the next major release of Greenplum Database.

- **Migrating Data with gpcopy**
  This topic describes how to use the gpcopy utility to transfer data between databases in different Greenplum Database clusters.
- **Migrating Data with gptransfer**
  This topic describes how to use the gptransfer utility to transfer data between databases.

Parent topic: Managing a Greenplum System

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Migrating Data with gpcopy

This topic describes how to use the gpcopy utility to transfer data between databases in different Greenplum Database clusters.

gpcopy is a high-performance utility that can copy metadata and data from one Greenplum database to another Greenplum database. You can migrate the entire contents of a database, or just selected tables. The clusters can have different Greenplum Database versions. For example, you can use gpcopy to migrate data from Greenplum 5 to Greenplum 6.

Note: The gpcopy utility is available as a separate download for the commercial release of Pivotal Greenplum Database. See the Pivotal gpcopy Documentation.
Parent topic: Migrating Data

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Migrating Data with gptransfer

This topic describes how to use the gptransfer utility to transfer data between databases.

Note: Greenplum Database provides two utilities for migrating data between Greenplum Database installations, gpcopy and gptransfer. Use gpcopy to migrate data to a Greenplum Database cluster version 5.9.0 and later. Use the gptransfer utility to migrate data between Greenplum Database installations running software version 5.8.0 or earlier.

The gptransfer utiltiy is deprecated and will be removed in the next major release of Greenplum Database. For information about migrating data with gpcopy, see Migrating Data with gpcopy.

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 postgresql.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`.
- Run the `gptransfer` utility either from the source or the destination cluster.
- If you are transferring data between two different clusters, the number of *segment instances* in the destination cluster must be greater than or equal to the number of segment *hosts* in the source cluster. The number of segment hosts in the destination cluster may be smaller than the number of segment hosts 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.

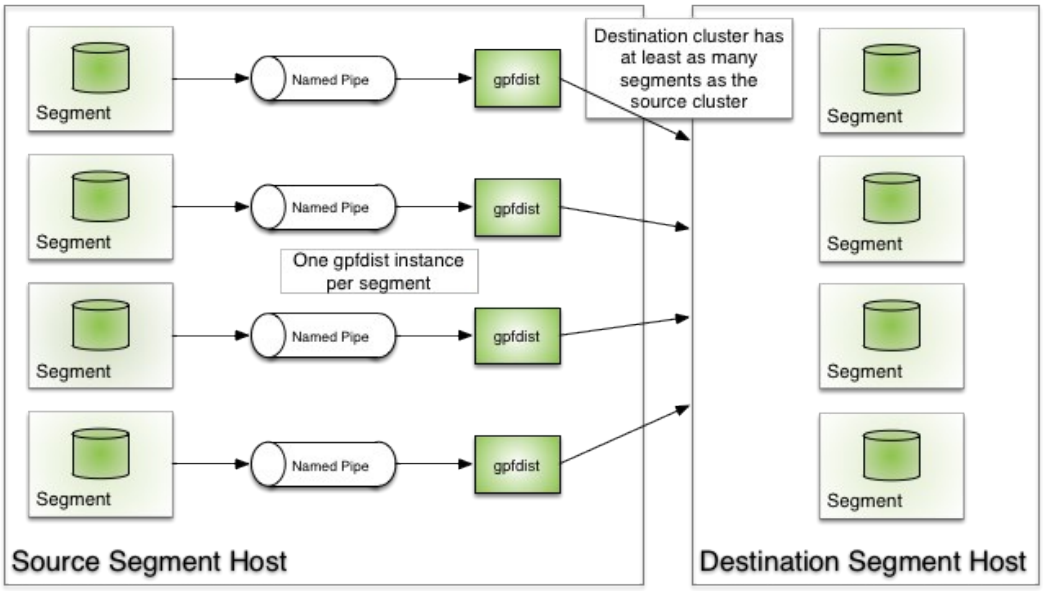
`gptransfer` orchestrates the process by processing the database objects to be transferred in batches. For each table to be transferred, it performs the following tasks:

- 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

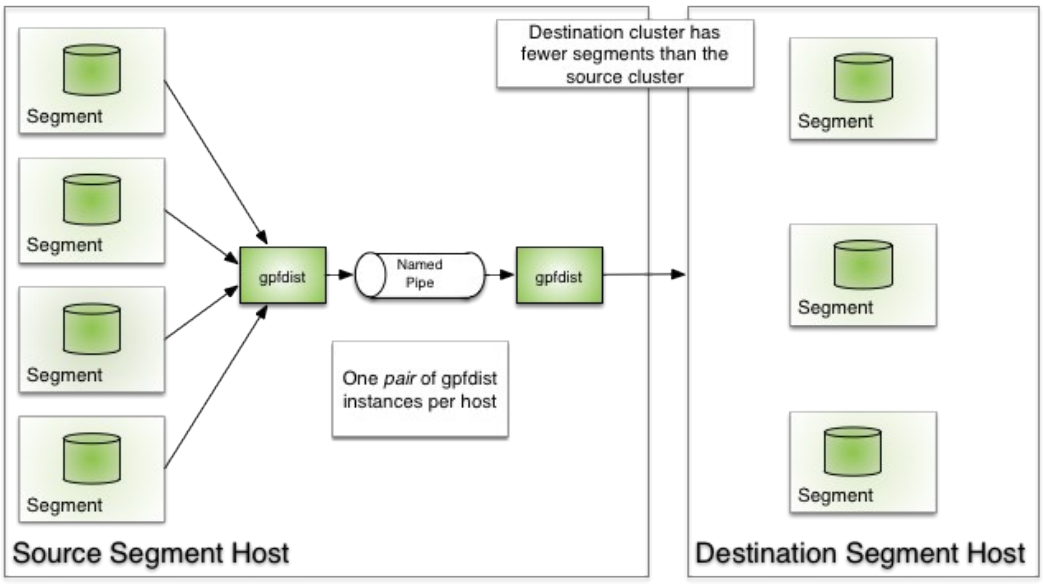
`gptransfer` sets up data transfer using the `gpfdist` parallel file serving utility, which serves the data evenly to the destination segments. Running more `gpfdist` processes increases the parallelism and the data transfer rate. When the destination cluster has the same or a greater number of

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, `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 `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 `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. 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.

When transferring data between databases, you can run only one instance of `gptransfer` at a time. Running multiple, concurrent instances of `gptransfer` is not supported.

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 s
ystem
```

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 databas
e 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.

Object	Full Mode	Table Mode
Data	Yes	Yes
Indexes	Yes	Yes
Roles	Yes	No
Functions	Yes	No
Resource Queues	Yes	No
`postgresql.conf`	No	No
`pg_hba.conf`	No	No
`gppkg`	No	No

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.

Parent topic: Migrating Data

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.

Parent topic: Managing a Greenplum 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 1. SNMP Prerequisites

Operating System	Packages[1]
Red Hat Enterprise	net-snmp net-snmp-libs net-snmp-utils
CentOS	net-snmp
SUSE	N/A

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 `gpconfig` 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 gpconfig 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, run this `psql` command to log into a database and test email notification with the `gp_elog` function:

```
$ psql -d <testdb> -U gpadmin -c "SELECT gp_elog('Test GPDB Email',true);"
```

The address you specified for the `gp_email_to` parameter should receive an email with `Test GPDB Email` in the subject line.

Note: If you have difficulty sending and receiving email notifications, verify the security settings for you 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:

```
$ gpstate
```

To see more detailed information about your Greenplum Database array configuration, use `gpstate` with the `-s` option:

```
$ 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:

```
$ gpstate -m
```

To see the primary to mirror segment mappings:

```
$ gpstate -c
```

To see the status of the standby master mirror:

```
$ 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:

```
=# 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 "retail.sales"
 Column      |     Type      | Modifiers | Description
-------------+---------------+-----------+-------------
 sale_id     | integer       |           |
 amt         | float         |           |
 date        | date          |           |
Has OIDs: no
Distributed by: (sale_id)
```

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 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:

```
=> 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:

```
=> \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
- The session_level_memory_consumption View

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:

```
$ 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.

When resource queue-based resource management is active, 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. Under the resource queue-based resource management scheme, Greenplum Database considers the session a runaway when the queries consume an excessive amount of memory. The Greenplum Database server configuration parameter `runaway_detector_activation_percent` governs the conditions under which Greenplum Database considers a session a runaway session.

The `is_runaway`, `runaway_vmem_mb`, and `runaway_command_cnt` columns are not applicable when resource group-based resource management is active.

Table 2. session_level_memory_consumption

column	type	references	description
`datname`	name		Name of the database that the session is connected to.
`sess_id`	integer		Session ID.
`usename`	name		Name of the session user.
`current_query`	text		Current SQL query that the session is running.
`segid`	integer		Segment ID.
`vmem_mb`	integer		Total vmem memory usage for the session in MB.
`is_runaway`	boolean		Session is marked as runaway on the segment.
`qe_count`	integer		Number of query processes for the session.
`active_qe_count`	integer		Number of active query processes for the session.
`dirty_qe_count`	integer		Number of query processes that have not yet released their memory. The value is `-1` for sessions that are not running.
`runaway_vmem_mb`	integer		Amount of vmem memory that the session was consuming when it was marked as a runaway session.

Table 2. session_level_memory_consumption

column	type	references	description
`runaway_command_cnt`	integer		Command count for the session when it was marked as a runaway session.
`idle_start`	timestamptz		The last time a query process in this session became idle.

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 3. Greenplum Database Server Log Format

#	Field Name	Data Type	Description
1	event_time	timestamp with time zone	Time that the log entry was written to the log
2	user_name	varchar(100)	The database user name
3	database_name	varchar(100)	The database name
4	process_id	varchar(10)	The system process ID (prefixed with "p")
5	thread_id	varchar(50)	The thread count (prefixed with "th")
6	remote_host	varchar(100)	On the master, the hostname/address of the client machine. On the segment, the hostname/address of the master.
7	remote_port	varchar(10)	The segment or master port number

Table 3. Greenplum Database Server Log Format

#	Field Name	Data Type	Description
8	session_start_time	timestamp with time zone	Time session connection was opened
9	transaction_id	int	Top-level transaction ID on the master. This ID is the parent of any subtransactions.
10	gp_session_id	text	Session identifier number (prefixed with "con")
11	gp_command_count	text	The command number within a session (prefixed with "cmd")
12	gp_segment	text	The segment content identifier (prefixed with "seg" for primaries or "mir" for mirrors). The master always has a content ID of -1.
13	slice_id	text	The slice ID (portion of the query plan being executed)
14	distr_tranx_id	text	Distributed transaction ID
15	local_tranx_id	text	Local transaction ID
16	sub_tranx_id	text	Subtransaction ID
17	event_severity	varchar(10)	Values include: LOG, ERROR, FATAL, PANIC, DEBUG1, DEBUG2
18	sql_state_code	varchar(10)	SQL state code associated with the log message
19	event_message	text	Log or error message text
20	event_detail	text	Detail message text associated with an error or warning message
21	event_hint	text	Hint message text associated with an error or warning message
22	internal_query	text	The internally-generated query text
23	internal_query_pos	int	The cursor index into the internally-generated query text
24	event_context	text	The context in which this message gets generated
25	debug_query_string	text	User-supplied query string with full detail for debugging. This string can be modified for internal use.
26	error_cursor_pos	int	The cursor index into the query string
27	func_name	text	The function in which this message is generated
28	file_name	text	The internal code file where the message originated
29	file_line	int	The line of the code file where the message originated
30	stack_trace	text	Stack trace text associated with this message

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:

```
$ 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:

```
$ 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:

```
=> 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:

```
iso(1)
identified-organization(3)
dod(6)
internet(1)
private(4)
enterprises(1)
gpdbMIB(31327)
gpdbObjects(1)
gpdbAlertMsg(1)
```

gpdbAlertMsg

```
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 4. SQL Codes

Error Code	Meaning	Constant
Class 00 — Successful Completion		
00000	SUCCESSFUL COMPLETION	successful_completion
Class 01 — Warning		
01000	WARNING	warning
0100C	DYNAMIC RESULT SETS RETURNED	dynamic_result_sets_returned
01008	IMPLICIT ZERO BIT PADDING	implicit_zero_bit_padding
01003	NULL VALUE ELIMINATED IN SET FUNCTION	null_value_eliminated_in_set_function
01007	PRIVILEGE NOT GRANTED	privilege_not_granted
01006	PRIVILEGE NOT REVOKED	privilege_not_revoked
01004	STRING DATA RIGHT TRUNCATION	string_data_right_truncation
01P01	DEPRECATED FEATURE	deprecated_feature
Class 02 — No Data (this is also a warning class per the SQL standard)		
02000	NO DATA	no_data
02001	NO ADDITIONAL DYNAMIC RESULT SETS RETURNED	no_additional_dynamic_result_sets_returned
Class 03 — SQL Statement Not Yet Complete		
03000	SQL STATEMENT NOT YET COMPLETE	sql_statement_not_yet_complete
Class 08 — Connection Exception		

Table 4. SQL Codes

Error Code	Meaning	Constant
08000	CONNECTION EXCEPTION	connection_exception
08003	CONNECTION DOES NOT EXIST	connection_does_not_exist
08006	CONNECTION FAILURE	connection_failure
08001	SQLCLIENT UNABLE TO ESTABLISH SQLCONNECTION	sqlclient_unable_to_establish_sqlconnection
08004	SQLSERVER REJECTED ESTABLISHMENT OF SQLCONNECTION	sqlserver_rejected_establishment_of_sqlconnection
08007	TRANSACTION RESOLUTION UNKNOWN	transaction_resolution_unknown
08P01	PROTOCOL VIOLATION	protocol_violation
Class 09 — Triggered Action Exception		
09000	TRIGGERED ACTION EXCEPTION	triggered_action_exception
Class 0A — Feature Not Supported		
0A000	FEATURE NOT SUPPORTED	feature_not_supported
Class 0B — Invalid Transaction Initiation		
0B000	INVALID TRANSACTION INITIATION	invalid_transaction_initiation
Class 0F — Locator Exception		
0F000	LOCATOR EXCEPTION	locator_exception
0F001	INVALID LOCATOR SPECIFICATION	invalid_locator_specification
Class 0L — Invalid Grantor		
0L000	INVALID GRANTOR	invalid_grantor
0LP01	INVALID GRANT OPERATION	invalid_grant_operation
Class 0P — Invalid Role Specification		
0P000	INVALID ROLE SPECIFICATION	invalid_role_specification
Class 21 — Cardinality Violation		
21000	CARDINALITY VIOLATION	cardinality_violation
Class 22 — Data Exception		
22000	DATA EXCEPTION	data_exception
2202E	ARRAY SUBSCRIPT ERROR	array_subscript_error
22021	CHARACTER NOT IN REPERTOIRE	character_not_in_repertoire
22008	DATETIME FIELD OVERFLOW	datetime_field_overflow
22012	DIVISION BY ZERO	division_by_zero
22005	ERROR IN ASSIGNMENT	error_in_assignment
2200B	ESCAPE CHARACTER CONFLICT	escape_character_conflict
22022	INDICATOR OVERFLOW	indicator_overflow
22015	INTERVAL FIELD OVERFLOW	interval_field_overflow
2201E	INVALID ARGUMENT FOR LOGARITHM	invalid_argument_for_logarithm
2201F	INVALID ARGUMENT FOR POWER FUNCTION	invalid_argument_for_power_function

Table 4. SQL Codes

Error Code	Meaning	Constant
2201G	INVALID ARGUMENT FOR WIDTH BUCKET FUNCTION	invalid_argument_for_width_bucket_function
22018	INVALID CHARACTER VALUE FOR CAST	invalid_character_value_for_cast
22007	INVALID DATETIME FORMAT	invalid_datetime_format
22019	INVALID ESCAPE CHARACTER	invalid_escape_character
2200D	INVALID ESCAPE OCTET	invalid_escape_octet
22025	INVALID ESCAPE SEQUENCE	invalid_escape_sequence
22P06	NONSTANDARD USE OF ESCAPE CHARACTER	nonstandard_use_of_escape_character
22010	INVALID INDICATOR PARAMETER VALUE	invalid_indicator_parameter_value
22020	INVALID LIMIT VALUE	invalid_limit_value
22023	INVALID PARAMETER VALUE	invalid_parameter_value
2201B	INVALID REGULAR EXPRESSION	invalid_regular_expression
22009	INVALID TIME ZONE DISPLACEMENT VALUE	invalid_time_zone_displacement_value
2200C	INVALID USE OF ESCAPE CHARACTER	invalid_use_of_escape_character
2200G	MOST SPECIFIC TYPE MISMATCH	most_specific_type_mismatch
22004	NULL VALUE NOT ALLOWED	null_value_not_allowed
22002	NULL VALUE NO INDICATOR PARAMETER	null_value_no_indicator_parameter
22003	NUMERIC VALUE OUT OF RANGE	numeric_value_out_of_range
22026	STRING DATA LENGTH MISMATCH	string_data_length_mismatch
22001	STRING DATA RIGHT TRUNCATION	string_data_right_truncation
22011	SUBSTRING ERROR	substring_error
22027	TRIM ERROR	trim_error
22024	UNTERMINATED C STRING	unterminated_c_string
2200F	ZERO LENGTH CHARACTER STRING	zero_length_character_string
22P01	FLOATING POINT EXCEPTION	floating_point_exception
22P02	INVALID TEXT REPRESENTATION	invalid_text_representation
22P03	INVALID BINARY REPRESENTATION	invalid_binary_representation
22P04	BAD COPY FILE FORMAT	bad_copy_file_format
22P05	UNTRANSLATABLE CHARACTER	untranslatable_character
Class 23 — Integrity Constraint Violation		
23000	INTEGRITY CONSTRAINT VIOLATION	integrity_constraint_violation
23001	RESTRICT VIOLATION	restrict_violation
23502	NOT NULL VIOLATION	not_null_violation
23503	FOREIGN KEY VIOLATION	foreign_key_violation
23505	UNIQUE VIOLATION	unique_violation
23514	CHECK VIOLATION	check_violation

Table 4. SQL Codes

Error Code	Meaning	Constant
Class 24 — Invalid Cursor State		
24000	INVALID CURSOR STATE	invalid_cursor_state
Class 25 — Invalid Transaction State		
25000	INVALID TRANSACTION STATE	invalid_transaction_state
25001	ACTIVE SQL TRANSACTION	active_sql_transaction
25002	BRANCH TRANSACTION ALREADY ACTIVE	branch_transaction_already_active
25008	HELD CURSOR REQUIRES SAME ISOLATION LEVEL	held_cursor_requires_same_isolation_level
25003	INAPPROPRIATE ACCESS MODE FOR BRANCH TRANSACTION	inappropriate_access_mode_for_branch_transaction
25004	INAPPROPRIATE ISOLATION LEVEL FOR BRANCH TRANSACTION	inappropriate_isolation_level_for_branch_transaction
25005	NO ACTIVE SQL TRANSACTION FOR BRANCH TRANSACTION	no_active_sql_transaction_for_branch_transaction
25006	READ ONLY SQL TRANSACTION	read_only_sql_transaction
25007	SCHEMA AND DATA STATEMENT MIXING NOT SUPPORTED	schema_and_data_statement_mixing_not_supported
25P01	NO ACTIVE SQL TRANSACTION	no_active_sql_transaction
25P02	IN FAILED SQL TRANSACTION	in_failed_sql_transaction
Class 26 — Invalid SQL Statement Name		
26000	INVALID SQL STATEMENT NAME	invalid_sql_statement_name
Class 27 — Triggered Data Change Violation		
27000	TRIGGERED DATA CHANGE VIOLATION	triggered_data_change_violation
Class 28 — Invalid Authorization Specification		
28000	INVALID AUTHORIZATION SPECIFICATION	invalid_authorization_specification
Class 2B — Dependent Privilege Descriptors Still Exist		
2B000	DEPENDENT PRIVILEGE DESCRIPTORS STILL EXIST	dependent_privilege_descriptors_still_exist
2BP01	DEPENDENT OBJECTS STILL EXIST	dependent_objects_still_exist
Class 2D — Invalid Transaction Termination		
2D000	INVALID TRANSACTION TERMINATION	invalid_transaction_termination
Class 2F — SQL Routine Exception		
2F000	SQL ROUTINE EXCEPTION	sql_routine_exception
2F005	FUNCTION EXECUTED NO RETURN STATEMENT	function_executed_no_return_statement
2F002	MODIFYING SQL DATA NOT PERMITTED	modifying_sql_data_not_permitted
2F003	PROHIBITED SQL STATEMENT ATTEMPTED	prohibited_sql_statement_attempted
2F004	READING SQL DATA NOT PERMITTED	reading_sql_data_not_permitted
Class 34 — Invalid Cursor Name		

Table 4. SQL Codes

Error Code	Meaning	Constant
34000	INVALID CURSOR NAME	invalid_cursor_name
Class 38 — External Routine Exception		
38000	EXTERNAL ROUTINE EXCEPTION	external_routine_exception
38001	CONTAINING SQL NOT PERMITTED	containing_sql_not_permitted
38002	MODIFYING SQL DATA NOT PERMITTED	modifying_sql_data_not_permitted
38003	PROHIBITED SQL STATEMENT ATTEMPTED	prohibited_sql_statement_attempted
38004	READING SQL DATA NOT PERMITTED	reading_sql_data_not_permitted
Class 39 — External Routine Invocation Exception		
39000	EXTERNAL ROUTINE INVOCATION EXCEPTION	external_routine_invocation_exception
39001	INVALID SQLSTATE RETURNED	invalid_sqlstate_returned
39004	NULL VALUE NOT ALLOWED	null_value_not_allowed
39P01	TRIGGER PROTOCOL VIOLATED	trigger_protocol_violated
39P02	SRF PROTOCOL VIOLATED	srf_protocol_violated
Class 3B — Savepoint Exception		
3B000	SAVEPOINT EXCEPTION	savepoint_exception
3B001	INVALID SAVEPOINT SPECIFICATION	invalid_savepoint_specification
Class 3D — Invalid Catalog Name		
3D000	INVALID CATALOG NAME	invalid_catalog_name
Class 3F — Invalid Schema Name		
3F000	INVALID SCHEMA NAME	invalid_schema_name
Class 40 — Transaction Rollback		
40000	TRANSACTION ROLLBACK	transaction_rollback
40002	TRANSACTION INTEGRITY CONSTRAINT VIOLATION	transaction_integrity_constraint_violation
40001	SERIALIZATION FAILURE	serialization_failure
40003	STATEMENT COMPLETION UNKNOWN	statement_completion_unknown
40P01	DEADLOCK DETECTED	deadlock_detected
Class 42 — Syntax Error or Access Rule Violation		
42000	SYNTAX ERROR OR ACCESS RULE VIOLATION	syntax_error_or_access_rule_violation
42601	SYNTAX ERROR	syntax_error
42501	INSUFFICIENT PRIVILEGE	insufficient_privilege
42846	CANNOT COERCE	cannot_coerce
42803	GROUPING ERROR	grouping_error
42830	INVALID FOREIGN KEY	invalid_foreign_key
42602	INVALID NAME	invalid_name
42622	NAME TOO LONG	name_too_long

Table 4. SQL Codes

Error Code	Meaning	Constant
42939	RESERVED NAME	reserved_name
42804	DATATYPE MISMATCH	datatype_mismatch
42P18	INDETERMINATE DATATYPE	indeterminate_datatype
42809	WRONG OBJECT TYPE	wrong_object_type
42703	UNDEFINED COLUMN	undefined_column
42883	UNDEFINED FUNCTION	undefined_function
42P01	UNDEFINED TABLE	undefined_table
42P02	UNDEFINED PARAMETER	undefined_parameter
42704	UNDEFINED OBJECT	undefined_object
42701	DUPLICATE COLUMN	duplicate_column
42P03	DUPLICATE CURSOR	duplicate_cursor
42P04	DUPLICATE DATABASE	duplicate_database
42723	DUPLICATE FUNCTION	duplicate_function
42P05	DUPLICATE PREPARED STATEMENT	duplicate_prepared_statement
42P06	DUPLICATE SCHEMA	duplicate_schema
42P07	DUPLICATE TABLE	duplicate_table
42712	DUPLICATE ALIAS	duplicate_alias
42710	DUPLICATE OBJECT	duplicate_object
42702	AMBIGUOUS COLUMN	ambiguous_column
42725	AMBIGUOUS FUNCTION	ambiguous_function
42P08	AMBIGUOUS PARAMETER	ambiguous_parameter
42P09	AMBIGUOUS ALIAS	ambiguous_alias
42P10	INVALID COLUMN REFERENCE	invalid_column_reference
42611	INVALID COLUMN DEFINITION	invalid_column_definition
42P11	INVALID CURSOR DEFINITION	invalid_cursor_definition
42P12	INVALID DATABASE DEFINITION	invalid_database_definition
42P13	INVALID FUNCTION DEFINITION	invalid_function_definition
42P14	INVALID PREPARED STATEMENT DEFINITION	invalid_prepared_statement_definition
42P15	INVALID SCHEMA DEFINITION	invalid_schema_definition
42P16	INVALID TABLE DEFINITION	invalid_table_definition
42P17	INVALID OBJECT DEFINITION	invalid_object_definition
Class 44 — WITH CHECK OPTION Violation		
44000	WITH CHECK OPTION VIOLATION	with_check_option_violation
Class 53 — Insufficient Resources		
53000	INSUFFICIENT RESOURCES	insufficient_resources
53100	DISK FULL	disk_full

Table 4. SQL Codes

Error Code	Meaning	Constant
53200	OUT OF MEMORY	out_of_memory
53300	TOO MANY CONNECTIONS	too_many_connections
Class 54 — Program Limit Exceeded		
54000	PROGRAM LIMIT EXCEEDED	program_limit_exceeded
54001	STATEMENT TOO COMPLEX	statement_too_complex
54011	TOO MANY COLUMNS	too_many_columns
54023	TOO MANY ARGUMENTS	too_many_arguments
Class 55 — Object Not In Prerequisite State		
55000	OBJECT NOT IN PREREQUISITE STATE	object_not_in_prerequisite_state
55006	OBJECT IN USE	object_in_use
55P02	CANT CHANGE RUNTIME PARAM	cant_change_runtime_param
55P03	LOCK NOT AVAILABLE	lock_not_available
Class 57 — Operator Intervention		
57000	OPERATOR INTERVENTION	operator_intervention
57014	QUERY CANCELED	query_canceled
57P01	ADMIN SHUTDOWN	admin_shutdown
57P02	CRASH SHUTDOWN	crash_shutdown
57P03	CANNOT CONNECT NOW	cannot_connect_now
Class 58 — System Error (errors external to Greenplum Database)		
58030	IO ERROR	io_error
58P01	UNDEFINED FILE	undefined_file
58P02	DUPLICATE FILE	duplicate_file
Class F0 — Configuration File Error		
F0000	CONFIG FILE ERROR	config_file_error
F0001	LOCK FILE EXISTS	lock_file_exists
Class P0 — PL/pgSQL Error		
P0000	PLPGSQL ERROR	plpgsql_error
P0001	RAISE EXCEPTION	raise_exception
P0002	NO DATA FOUND	no_data_found
P0003	TOO MANY ROWS	too_many_rows
Class XX — Internal Error		
XX000	INTERNAL ERROR	internal_error
XX001	DATA CORRUPTED	data_corrupted
XX002	INDEX CORRUPTED	index_corrupted

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Managing a Greenplum System

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 `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.

Important: Greenplum Database monitors transaction IDs. If you do not vacuum the database regularly, Greenplum Database will generate a warning and error.

Greenplum Database issues the following warning 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 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:

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

The Greenplum Database configuration parameter `xid_warn_limit` controls when the warning is displayed. The parameter `xid_stop_limit` controls when Greenplum Database stops creating transactions.

Recovering from a Transaction ID Limit Error

When Greenplum Database reaches the `xid_stop_limit` transaction ID limit due to infrequent `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 `xid_stop_limit` by 10,000,000.
3. Start Greenplum Database.
4. Run `VACUUM FREEZE` on all affected databases.
5. Reset the `xid_stop_limit` to its original value.
6. Restart Greenplum Database.

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

For information about transaction ID wraparound see the PostgreSQL documentation.

System Catalog Maintenance

Numerous database updates with `CREATE` and `DROP` commands increase the system catalog size and affect system performance. For example, running many `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 `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.

Regular System Catalog Maintenance

It is recommended that you periodically run `REINDEX` and `VACUUM` on the system catalog to clear the space that deleted objects occupy in the system indexes and tables. 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.

These are Greenplum Database system catalog maintenance steps.

1. Perform a `REINDEX` on the system catalog tables to rebuild the system catalog indexes. This removes bloat in the indexes and improves `VACUUM` performance.
   Note: When performing `REINDEX` on the system catalog tables, locking will occur on the tables and might have an impact on currently running queries. You can schedule the

REINDEX operation during a period of low activity to avoid disrupting ongoing business operations.

2. Perform a VACUUM on the system catalog tables.
3. Perform an ANALYZE on the system catalog tables to update the catalog table statistics.

This example script performs a REINDEX, VACUUM, and ANALYZE of a Greenplum Database system catalog. In the script, replace <*database-name*> with a database name.

```
#!/bin/bash
DBNAME="<database-name>"
SYSTABLES="' pg_catalog.' || relname || ';' FROM pg_class a, pg_namespace b
WHERE a.relnamespace=b.oid AND b.nspname='pg_catalog' AND a.relkind='r'"

reindexdb --system -d $DBNAME
psql -tc "SELECT 'VACUUM' || $SYSTABLES" $DBNAME | psql -a $DBNAME
analyzedb -s pg_catalog -d $DBNAME
```

Note: If you are performing catalog maintenance during a maintenance period and you need to stop a process due to time constraints, run the Greenplum Database function pg_cancel_backend(<*PID*>) to safely stop the Greenplum Database process.

Intensive System Catalog Maintenance

If system catalog maintenance 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 VACUUM 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.

These are steps for intensive system catalog maintenance.

1. Stop all catalog activity on the Greenplum Database system.
2. Perform a REINDEX on the system catalog tables to rebuild the system catalog indexes. This removes bloat in the indexes and improves VACUUM performance.
3. Perform a VACUUM FULL on the system catalog tables. See the following Note.
4. Perform an ANALYZE on the system catalog tables to update the catalog table statistics.

Note: The system catalog table pg_attribute is usually the largest catalog table. If the pg_attribute table is significantly bloated, a VACUUM FULL operation on the table might require a significant amount of time and might need to be performed separately. The presence of both of these conditions indicate a significantly bloated pg_attribute table that might require a long VACUUM FULL time:

- The pg_attribute table contains a large number of records.
- The diagnostic message for pg_attribute is significant amount of bloat in the gp_toolkit.gp_bloat_diag view.

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:

```
=# 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.

```
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 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. See the `analyzedb` utility.

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:

```
script_name_date.log
```

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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Managing a Greenplum System

Database State Monitoring Activities

Table 1. Database State Monitoring Activities

Activity	Procedure	Corrective Actions

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.
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	Execute the following query in the `postgres` database: `SELECT * FROM gp_segment_configuration WHERE mode = 'r';`	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.
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	Execute the following query in the `postgres` database: `SELECT * FROM gp_segment_configuration WHERE preferred_role <> role;`	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.

Table 1. Database State Monitoring Activities

Activity	Procedure	Corrective Actions
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	Execute the following query in the `postgres` database: `SELECT gp_segment_id, count(*) FROM gp_dist_random('pg_class') GROUP BY 1;`	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.
Test the state of master mirroring on a Greenplum Database 4.2 or earlier cluster. If the value is "Not Synchronized", raise an alert or warning. Recommended frequency: run every 5 to 10 minutes Severity: IMPORTANT	Execute the following query in the `postgres` database: `SELECT summary_state FROM gp_master_mirroring;`	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.
Test the state of master mirroring on Greenplum Database. If the value is not "STREAMING", raise an alert or warning. Recommended frequency: run every 5 to 10 minutes Severity: IMPORTANT	Run the following `psql` command: `psql *dbname* -c 'SELECT procpid, state FROM pg_stat_replication;'`	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.
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: `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 2. Database Alert Log Monitoring Activities

Activity	Procedure	Corrective Actions

Check for FATAL and ERROR log messages from the system. Recommended frequency: run every 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.*	Run the following query in the `gpperfmon` database: `SELECT * FROM log_alert_history WHERE logseverity in ('FATAL', 'ERROR') AND logtime > (now() - interval '15 minutes');`	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.
Set up server configuration parameters to send SNMP or email alerts. Recommended frequency: N/A. Alerts are generated by the system. 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.*	Enable server configuration parameters to send alerts via SNMP or email: • `gp_email_smtp_server` • `gp_email_smtp_userid` • `gp_email_smtp_password` or `gp_snmp_monitor_address` • `gp_snmp_community` • `gp_snmp_use_inform_or_trap`	DBA takes action based on the nature of the alert.

Hardware and Operating System Monitoring

Table 3. Hardware and Operating System Monitoring Activities

Activity	Procedure	Corrective Actions
Underlying platform check for maintenance required or system down of the hardware. Recommended frequency: real-time, if possible, or every 15 minutes Severity: CRITICAL	Set up SNMP or other system check for hardware and OS errors.	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 `gprecoverseg`.
Check disk space usage on volumes used for Greenplum Database data storage and the OS. Recommended frequency: every 5 to 30 minutes Severity: CRITICAL	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%.	Free space on the system by removing some data or files.
Check for errors or dropped packets on the network interfaces. Recommended frequency: hourly Severity: IMPORTANT	Set up a network interface checks.	Work with network and OS teams to resolve errors.
Check for RAID errors or degraded RAID performance. Recommended frequency: every 5 minutes Severity: CRITICAL	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.

Table 3. Hardware and Operating System Monitoring Activities

Activity	Procedure	Corrective Actions
Run the Greenplum `gpcheck` 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	Run `gpcheck`.	Work with system administration team to update configuration according to the recommendations made by the `gpcheck` utility.
Check for adequate I/O bandwidth and I/O skew. Recommended frequency: when create a cluster or when hardware issues are suspected.	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 4. Catalog Monitoring Activities

Activity	Procedure	Corrective Actions
Run catalog consistency checks to ensure the catalog on each host in the cluster is consistent and in a good state. Recommended frequency: weekly Severity: IMPORTANT	Run the Greenplum `gpcheckcat` utility in each database: `gpcheckcat -O`	Run repair scripts for any issues detected.
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: `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: `gpcheckcat -R pgclass`	Run the repair scripts for any issues identified.
Check for leaked temporary schema and missing schema definition. Recommended frequency: monthly Severity: IMPORTANT	During a downtime, with no users on the system, run the Greenplum `gpcheckcat` utility in each database: `gpcheckcat -R namespace`	Run the repair scripts for any issues identified.

Table 4. Catalog Monitoring Activities

Activity	Procedure	Corrective Actions
Check constraints on randomly distributed tables. Recommended frequency: monthly Severity: IMPORTANT	During a downtime, with no users on the system, run the Greenplum `gpcheckcat` utility in each database: `gpcheckcat -R distribution_policy`	Run the repair scripts for any issues identified.
Check for dependencies on non-existent objects. Recommended frequency: monthly Severity: IMPORTANT	During a downtime, with no users on the system, run the Greenplum `gpcheckcat` utility in each database: `gpcheckcat -R dependency`	Run the repair scripts for any issues identified.

Data Maintenance

Table 5. Data Maintenance Activities

Activity	Procedure	Corrective Actions
Check for missing statistics on tables.	Check the `gp_stats_missing` view in each database: `SELECT * FROM gp_toolkit.gp_stats_missing;`	Run `ANALYZE` on tables that are missing statistics.
Check for tables that have bloat (dead space) in data files that cannot be recovered by a regular `VACUUM` command. Recommended frequency: weekly or monthly Severity: WARNING	Check the `gp_bloat_diag` view in each database: `SELECT * FROM gp_toolkit.gp_bloat_diag;`	Execute a `VACUUM FULL` statement at a time when users are not accessing the table to remove bloat and compact the data.

Database Maintenance

Table 6. Database Maintenance Activities

Activity	Procedure	Corrective Actions
Mark deleted rows in heap tables so that the space they occupy can be reused. Recommended frequency: daily Severity: CRITICAL	Vacuum user tables: `VACUUM <table>;`	Vacuum updated tables regularly to prevent bloating.

Table 6. Database Maintenance Activities

Activity	Procedure	Corrective Actions
Update table statistics. Recommended frequency: after loading data and before executing queries Severity: CRITICAL	Analyze user tables. You can use the `analyzedb` management utility: `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
Note: Starting in Greenplum 5.x, `VACUUM` is supported with persistent table system catalogs, and is required to manage free space.
	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:
	`reindexdb -s <database>`
	3. `ANALYZE` each of the system tables:
	`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 7. Patch and Upgrade Activities

Activity	Procedure	Corrective Actions

Ensure any bug fixes or enhancements are applied to the kernel. Recommended frequency: at least every 6 months Severity: IMPORTANT	Follow the vendor's instructions to update the Linux kernel.	Keep the kernel current to include bug fixes and security fixes, and to avoid difficult future upgrades.
Install Greenplum Database minor releases, for example 5.0.*x*. Recommended frequency: quarterly Severity: IMPORTANT	Follow upgrade instructions in the Greenplum Database *Release Notes*. Always upgrade to the latest in the series.	Keep the Greenplum Database software current to incorporate bug fixes, performance enhancements, and feature enhancements into your Greenplum cluster.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Managing Greenplum Database Access

Securing Greenplum Database includes protecting access to the database through network configuration, database user authentication, and encryption.

- **Configuring Client Authentication**
  This topic explains how to configure client connections and authentication for Greenplum Database.
- **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.

Parent topic: Greenplum Database Administrator Guide

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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. 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.

- **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.
- **Using Kerberos Authentication**
  You can control access to Greenplum Database with a Kerberos authentication server.
- **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.
- **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.

Parent topic: Managing Greenplum Database Access

Allowing Connections to Greenplum Database

Client access and authentication is controlled by the standard PostgreSQL host-based authentication file, pg_hba.conf. 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 Database system. The Greenplum Database segments also have pg_hba.conf files, but these are already correctly configured to allow only 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. Greenplum Database ignores blank lines and any text after the `#` comment character. A record consists of a number of fields that are separated by spaces or tabs. Fields can contain white space if the field value is quoted. Records cannot be continued across lines. Each remote client access record has the following format:

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

Each UNIX-domain socket access record is in this format:

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

The following table describes meaning of each field.

Table 1. pg_hba.conf Fields

Field	Description
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` server configuration parameter.
hostnossl	Matches connection attempts made over TCP/IP that do not use SSL.
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 a `@`.
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 a `@`.

Table 1. pg_hba.conf Fields

Field	Description
address	Specifies the client machine addresses that this record matches. This field can contain an IP address, an IP address range, or a host name. An IP address range is specified using standard numeric notation for the range's starting address, then a slash (/) and a CIDR mask length. The mask length indicates the number of high-order bits of the client IP address that must match. Bits to the right of this should 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 an IPv4 address range specified this way 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. An IPv6 address range might look like `::1/128` for a single host (in this case the IPv6 loopback address) or `fe80::7a31:c1ff:0000:0000/96` for a small network. `0.0.0.0/0` represents all IPv4 addresses, and `::0/0` represents all IPv6 addresses. To specify a single host, use a mask length of 32 for IPv4 or 128 for IPv6. In a network address, do not omit trailing zeroes. An entry given in IPv4 format will match only IPv4 connections, and an entry given in IPv6 format will match only IPv6 connections, even if the represented address is in the IPv4-in-IPv6 range. Note: Entries in IPv6 format will be rejected if the host system C library does not have support for IPv6 addresses. If a host name is specified (an address that is not an IP address or IP range is treated as a host name), that name is compared with the result of a reverse name resolution of the client IP address (for example, reverse DNS lookup, if DNS is used). Host name comparisons are case insensitive. If there is a match, then a forward name resolution (for example, forward DNS lookup) is performed on the host name to check whether any of the addresses it resolves to are equal to the client IP address. If both directions match, then the entry is considered to match. Some host name databases allow associating an IP address with multiple host names, but the operating system only returns one host name when asked to resolve an IP address. The host name that is used in `pg_hba.conf` must be the one that the address-to-name resolution of the client IP address returns, otherwise the line will not be considered a match. When host names are specified in `pg_hba.conf`, you should ensure that name resolution is reasonably fast. It can be of advantage to set up a local name resolution cache such as `nscd`. Also, you can enable the server configuration parameter `log_hostname` to see the client host name instead of the IP address in the log.
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.
authentication-method	Specifies the authentication method to use when connecting. Greenplum supports the authentication methods supported by PostgreSQL 9.0.

CAUTION:
For a more secure system, consider removing records for remote connections that use trust authentication from the `pg_hba.conf` file. Trust authentication grants any user who can connect to the server access to the database using any role they specify. You can safely replace trust authentication with ident authentication for local UNIX-socket connections. You can also use ident authentication for local and remote TCP clients, but the client host must be running an ident service and you must trust the integrity of that machine.

Editing the pg_hba.conf File

Initially, the `pg_hba.conf` file is set up with generous permissions for the gpadmin user and no database access for other Greenplum Database roles. You will need to edit the `pg_hba.conf` file to enable users' access to databases and to secure the gpadmin user. Consider removing entries that have trust authentication, since they allow anyone with access to the server to connect with any role they choose. For local (UNIX socket) connections, use ident authentication, which requires the operating system user to match the role specified. For local and remote TCP connections, ident

authentication requires the client's host to run an indent service. You can install an ident service on the master host and then use ident authentication for local TCP connections, for example `127.0.0.1/28`. Using ident authentication for remote TCP connections is less secure because it requires you to trust the integrity of the ident service on the client's host.

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.

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 ldapserver=usldap1 ldapport=1389 ldapp
   refix="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 recommended value of `max_connections` on segments is 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.

   ```
   $ gpconfig -c max_connections -v 1000 -m 200
   ```

   The value on the segments must be greater than the value on the master. The recommended value of max_connections on segments is 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.

   ```
   $ 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.

   ```
   $ 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.

   ```
   $ 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.

Enabling Greenplum Database in SSL mode requires the following items.

- OpenSSL installed on both the client and the master server hosts (master and standby master).
- The SSL files server.key (server private key) and server.crt (server certificate) should be correctly generated for the master host and standby master host.

    - The private key should not be protected with a passphrase. The server does not prompt for a passphrase for the private key, and Greenplum Database start up fails with an error if one is required.
    - On a production system, there should be a key and certificate pair for the master host and a pair for the standby master host with a subject CN (Common Name) for the master host and standby master host.

  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.

- The server.key and server.crt have been added to the master data directory on both the master host and standby master host. When starting in SSL mode, the Greenplum Database master looks for server.key and server.crt. Greenplum Database does not start if the files are not in the master data directory of both the master host and standby master host.

  Also, if you use other SSL authentication files such as `root.crt` (trusted certificate authorities), the files must be on both the master host and standby master host.

Greenplum Database can be started with SSL enabled by setting the server configuration parameter `ssl=on` in the `postgresql.conf` file on the master and standby master hosts. This `gpconfig` command sets the parameter:

```
gpconfig -c ssl -m on -v off
```

Setting the parameter requires a server restart. This command restarts the system: `gpstop -ra -M fast`.

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:

```
# 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:

```
# 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.

```
# 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.

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

For more details on how to create your server private key and certificate, refer to the OpenSSL documentation.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x 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.

Enabing 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=examp
le,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=Pe
ople,dc=example,dc=com"
```

Enabing 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
    URI ldaps://ldapA.example.priv ldaps://ldapB.example.priv ldaps://ldapC.exampl
   e.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 `ldapport` 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=" ldap
```

```
suffix=",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 ldapserver=ldaps://myldap.com ldapprefix="uid=" ldap
suffix=",ou=People,dc=example,dc=com"
```

Parent topic: Configuring Client Authentication

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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/.

Prerequisites

Before configuring Kerberos authentication for Greenplum Database, ensure that:

- You can identify the KDC server you use for Kerberos authentication and the Kerberos realm for your Greenplum Database system.
    - If you plan to use an MIT Kerberos KDC server but have not yet configured it, see Installing and Configuring a Kerberos KDC Server for example instructions.
    - If you are using an existing Active Directory KDC server, also ensure that you have:
        - Installed all Active Directory service roles on your AD KDC server.
        - Enabled the LDAP service.
- System time on the Kerberos Key Distribution Center (KDC) server and Greenplum Database master is synchronized. (For example, install the `ntp` package on both servers.)
- Network connectivity exists between the KDC server and the Greenplum Database master host.
- Java 1.7.0_17 or later is installed on all Greenplum Database hosts. Java 1.7.0_17 is required to use Kerberos-authenticated JDBC on Red Hat Enterprise Linux 6.x or 7.x.

Procedure

Following are the tasks to complete to set up Kerberos authentication for Greenplum Database.

- Creating Greenplum Database Principals in the KDC Database
- Installing the Kerberos Client on the Master Host
- Configuring Greenplum Database to use Kerberos Authentication
- Mapping Kerberos Principals to Greenplum Database Roles

- Configuring JDBC Kerberos Authentication for Greenplum Database
- Configuring Kerberos on Windows for Greenplum Database Clients
- Configuring Client Authentication with Active Directory

Parent topic: Configuring Client Authentication

Creating Greenplum Database Principals in the KDC Database

Create a service principal for the Greenplum Database service and a Kerberos admin principal that allows managing the KDC database as the gpadmin user.

1. Log in to the Kerberos KDC server as the root user.

   ```
   $ ssh root@<kdc-server>
   ```

2. Create a principal for the Greenplum Database service.

   ```
   # kadmin.local -q "addprinc -randkey postgres/mdw@GPDB.KRB"
   ```

   The `-randkey` option prevents the command from prompting for a password.

   The `postgres` part of the principal names matches the value of the Greenplum Database `krb_srvname` server configuration parameter, which is `postgres` by default.

   The host name part of the principal name must match the output of the `hostname` command on the Greenplum Database master host. If the `hostname` command shows the fully qualified domain name (FQDN), use it in the principal name, for example `postgres/mdw.example.com@GPDB.KRB`.

   The `GPDB.KRB` part of the principal name is the Kerberos realm name.

3. Create a principal for the gpadmin/admin role.

   ```
   # kadmin.local -q "addprinc gpadmin/admin@GPDB.KRB"
   ```

   This principal allows you to manage the KDC database when you are logged in as gpadmin. Make sure that the Kerberos `kadm.acl` configuration file contains an ACL to grant permissions to this principal. For example, this ACL grants all permissions to any admin user in the GPDB.KRB realm.

   ```
   */admin@GPDB.KRB *
   ```

4. Create a keytab file with `kadmin.local`. The following example creates a keytab file `gpdb-kerberos.keytab` in the current directory with authentication information for the Greenplum Database service principal and the gpadmin/admin principal.

   ```
   # kadmin.local -q "ktadd -k gpdb-kerberos.keytab postgres/mdw@GPDB.KRB gadmin/a
   dmin@GPDB.KRB"
   ```

5. Copy the keytab file to the master host.

   ```
   # scp gpdb-kerberos.keytab gpadmin@mdw:~
   ```

Installing the Kerberos Client on the Master Host

Install the Kerberos client utilities and libraries on the Greenplum Database master.

1. Install the Kerberos packages on the Greenplum Database master.

```
$ sudo yum install krb5-libs krb5-workstation
```

2. Copy the `/etc/krb5.conf` file from the KDC server to `/etc/krb5.conf` on the Greenplum Master host.

Configuring Greenplum Database to use Kerberos Authentication

Configure Greenplum Database to use Kerberos.

1. Log in to the Greenplum Database master host as the gpadmin user.

```
$ ssh gpadmin@<master>
$ source /usr/local/greenplum-db/greenplum_path.sh
```

2. Set the ownership and permissions of the keytab file you copied from the KDC server.

```
$ chown gpadmin:gpadmin /home/gpadmin/gpdb-kerberos.keytab
$ chmod 400 /home/gpadmin/gpdb-kerberos.keytab
```

3. Configure the location of the keytab file by setting the Greenplum Database `krb_server_keyfile` server configuration parameter. This `gpconfig` command specifies the folder /home/gpadmin as the location of the keytab file gpdb-kerberos.keytab.

```
$ gpconfig -c krb_server_keyfile -v  '/home/gpadmin/gpdb-kerberos.keytab'
```

4. Modify the Greenplum Database file `pg_hba.conf` to enable Kerberos support. For example, adding the following line to `pg_hba.conf` adds GSSAPI and Kerberos authentication support for connection requests from all users and hosts on the same network to all Greenplum Database databases.

```
host all all 0.0.0.0/0 gss include_realm=0 krb_realm=GPDB.KRB
```

   Setting the `krb_realm` option to a realm name ensures that only users from that realm can successfully authenticate with Kerberos. Setting the `include_realm` option to `0` excludes the realm name from the authenticated user name. For information about the `pg_hba.conf` file, see The pg_hba.conf file in the Postgres documentation.

5. Restart Greenplum Database after updating the `krb_server_keyfile` parameter and the `pg_hba.conf` file.

```
$ gpstop -ar
```

6. Create the gpadmin/admin Greenplum Database superuser role.

```
$ createuser gpadmin/admin
Shall the new role be a superuser? (y/n) y
```

   The Kerberos keys for this database role are in the keyfile you copied from the KDC server.

7. Create a ticket using `kinit` and show the tickets in the Kerberos ticket cache with `klist`.

```
$ LD_LIBRARY_PATH= kinit -k -t /home/gpadmin/gpdb-kerberos.keytab gpadmin/admin
@GPDB.KRB
$ LD_LIBRARY_PATH= klist
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: gpadmin/admin@GPDB.KRB

Valid starting       Expires              Service principal
06/13/2018 17:37:35  06/14/2018 17:37:35  krbtgt/GPDB.KRB@GPDB.KRB
```

   Note: When you set up the Greenplum Database environment by sourcing the `greenplum-`

`db_path.sh` script, the `LD_LIBRARY_PATH` environment variable is set to include the Greenplum Database `lib` directory, which includes Kerberos libraries. This may cause Kerberos utility commands such as `kinit` and `klist` to fail due to version conflicts. The solution is to run Kerberos utilities before you source the `greenplum-db_path.sh` file or temporarily unset the `LD_LIBRARY_PATH` variable when you execute Kerberos utilities, as shown in the example.

8. As a test, log in to the postgres database with the `gpadmin/admin` role:

```
$ psql -U "gpadmin/admin" -h mdw postgres
psql (8.3.23)
Type "help" for help.

postgres=# select current_user;
 current_user
---------------
 gpadmin/admin
(1 row)
```

Note: When you start `psql` on the master host, you must include the `-h <master-hostname>` option to force a TCP connection because Kerberos authentication does not work with local connections.

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.

Mapping Kerberos Principals to Greenplum Database Roles

To connect to a Greenplum Database system with Kerberos authentication enabled, a user first requests a ticket-granting ticket from the KDC server using the `kinit` utility with a password or a keytab file provided by the Kerberos admin. When the user then connects to the Kerberos-enabled Greenplum Database system, the user's Kerberos principle name will be the Greenplum Database role name, subject to transformations specified in the options field of the `gss` entry in the Greenplum Database `pg_hba.conf` file:

- If the `krb_realm=<realm>` option is present, Greenplum Database only accepts Kerberos principals who are members pf the specified realm.
- If the `include_realm=0` option is specified, the Greenplum Database role name is the Kerberos principal name without the Kerberos realm. If the `include_realm=1` option is instead specified, the Kerberos realm is not stripped from the Greenplum Database rolename. The role must have been created with the Greenplum Database `CREATE ROLE` command.
- If the `map=<map-name>` option is specified, the Kerberos principal name is compared to entries labeled with the specified `<map-name>` in the `$MASTER_DATA_DIRECTORY/pg_ident.conf` file and replaced with the Greenplum Database role name specified in the first matching entry.

A user name map is defined in the `$MASTER_DATA_DIRECTORY/pg_ident.conf` configuration file. This example defines a map named `mymap` with two entries.

```
# MAPNAME    SYSTEM-USERNAME         GP-USERNAME
mymap        /^admin@GPDB.KRB$       gpadmin
mymap        /^(.*)_gp)@GPDB.KRB$    \1
```

The map name is specified in the `pg_hba.conf` Kerberos entry in the options field:

```
host all all 0.0.0.0/0 gss include_realm=0 krb_realm=GPDB.KRB map=mymap
```

The first map entry matches the Kerberos principal admin@GPDB.KRB and replaces it with the Greenplum Database gpadmin role name. The second entry uses a wildcard to match any Kerberos principal in the GPDB-KRB realm with a name ending with the characters `_gp` and replaces it with the initial portion of the principal name. Greenplum Database applies the first matching map entry in the `pg_ident.conf` file, so the order of entries is significant.

For more information about using username maps see Username maps in the PostgreSQL documentation.

Configuring JDBC Kerberos Authentication for Greenplum Database

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 Installing the Kerberos Client on the Master Host.
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
     debug=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/gpdb-kdc
   ```

   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.

Installing and Configuring 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.

If you do not already have a KDC, follow these steps to install and configure a KDC server on a Red Hat Enterprise Linux host with a `GPDB.KRB` realm. The host name of the KDC server in this example is gpdb-kdc.

1. Install the Kerberos server and client packages:

   ```
   $ sudo yum install krb5-libs krb5-server krb5-workstation
   ```
2. Edit the /etc/krb5.conf configuration file. The following example shows a Kerberos server configured with a default `GPDB.KRB` realm.

   ```
   [logging]
    default = FILE:/var/log/krb5libs.log
    kdc = FILE:/var/log/krb5kdc.log
    admin_server = FILE:/var/log/kadmind.log
   ```

```
[libdefaults]
 default_realm = GPDB.KRB
 dns_lookup_realm = false
 dns_lookup_kdc = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = true
 default_tgs_enctypes = aes128-cts des3-hmac-sha1 des-cbc-crc des-cbc-md5
 default_tkt_enctypes = aes128-cts des3-hmac-sha1 des-cbc-crc des-cbc-md5
 permitted_enctypes = aes128-cts des3-hmac-sha1 des-cbc-crc des-cbc-md5

[realms]
 GPDB.KRB = {
  kdc = gpdb-kdc:88
  admin_server = gpdb-kdc:749
  default_domain = gpdb.krb
 }

[domain_realm]
 .gpdb.krb = GPDB.KRB
 gpdb.krb = GPDB.KRB

[appdefaults]
 pam = {
    debug = false
    ticket_lifetime = 36000
    renew_lifetime = 36000
    forwardable = true
    krb4_convert = false
 }
```

The `kdc` and `admin_server` keys in the `[realms]` section specify the host (`gpdb-kdc`) 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 `GPDB.KRB` realm in the `[realms]` and `[domain_realm]` section of the `kdc.conf` file. See the Kerberos documentation for information about the `kdc.conf` file.

3. To create the Kerberos database, run the `kdb5_util`.

```
# kdb5_util create -s
```

The `kdb5_util` create command 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.

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
```

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 Red Hat 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*.

Parent topic: Configuring Client Authentication

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
  - krb5-workstation

  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 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:

   ```
   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.

```
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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

- Configuring Kerberos on Windows for Greenplum Database Clients
- Configuring Client Authentication with Active Directory

For information about configuring Greenplum Database with Kerberos authentication, see Using Kerberos Authentication.

Parent topic: Configuring Client Authentication

Configuring 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 Configuring 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.

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\Kerberos5\krb5.ini`. 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.

```
[libdefaults]
 debug = true
 default_etypes = aes256-cts-hmac-sha1-96
 default_realm = EXAMPLE.LOCAL
 dns_lookup_realm = false
 dns_lookup_kdc = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = true

[realms]
 EXAMPLE.LOCAL = {
  kdc =bocdc.example.local
  admin_server = bocdc.example.local
 }

[domain_realm]
 .example.local = EXAMPLE.LOCAL
 example.local = EXAMPLE.LOCAL
```

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.

```
set KRB5CCNAME=%USERPROFILE%\krb5cache
kinit

"c:\Program Files (x86)\Greenplum\greenplum-clients-<version>\greenplum_clients_path.b
at"
```

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.

```
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)
Type "help" for help.
```

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 keyab 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 keyab 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 ketyab `dev1.keytab`.

```
ktpass -out dev1.keytab -princ dev1@EXAMPLE.LOCAL -mapUser dev1 -pass your_password -c
rypto 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.

```
---
VERSION: 1.0.0.1
DATABASE: warehouse
HOST: prod1.example.local
PORT: 5432

GPLOAD:
   INPUT:
    - SOURCE:
         PORT_RANGE: [18080,18080]
         FILE:
           - /Users/dev1/Downloads/test.csv
    - FORMAT: text
    - DELIMITER: ','
    - QUOTE: '"'
    - ERROR_LIMIT: 25
    - LOG_ERRORS: true
   OUTPUT:
    - TABLE: public.test
    - MODE: INSERT
   PRELOAD:
    - REUSE_TABLES: true
```

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/t
est.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 `KRB5CCNAME` and run `kinit`.

  ```
  set KRB5CCNAME=%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.

Configuring 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 Configuring Kerberos on Windows for Greenplum Database Clients.
This section contains the following information.

- Prerequisites
- Setting Up Active Directory
- Setting Up Greenplum Database 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 do not have a domain, you must configure the system to use a domain.

  This Linux `hostname` command displays the FQDN.

  ```
  hostname --fqdn
  ```

- 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.

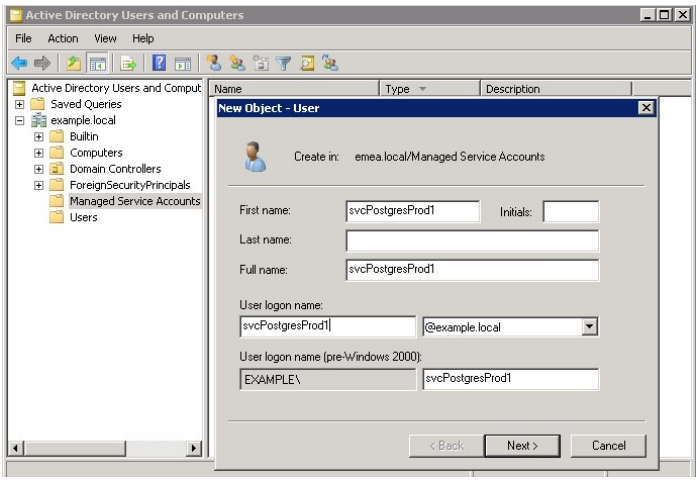
Setting Up Active Directory

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.

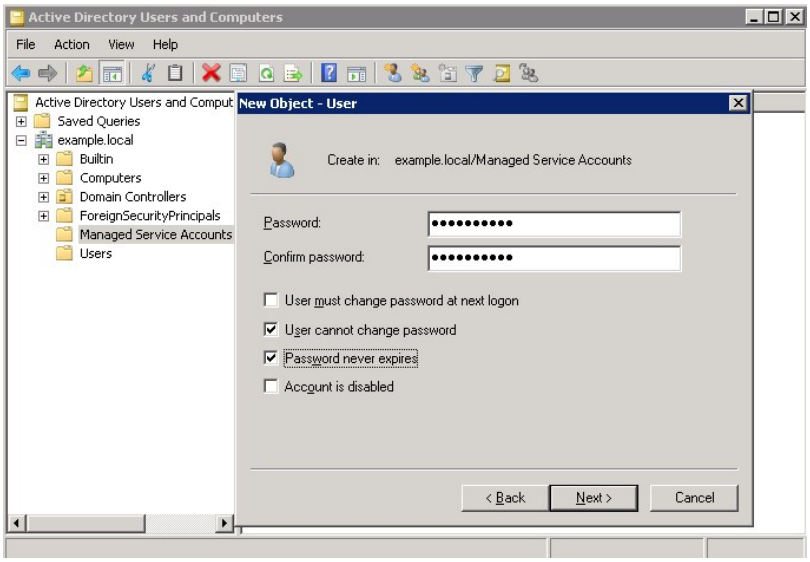
The Active Directory domain is `example.local`.

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`*.



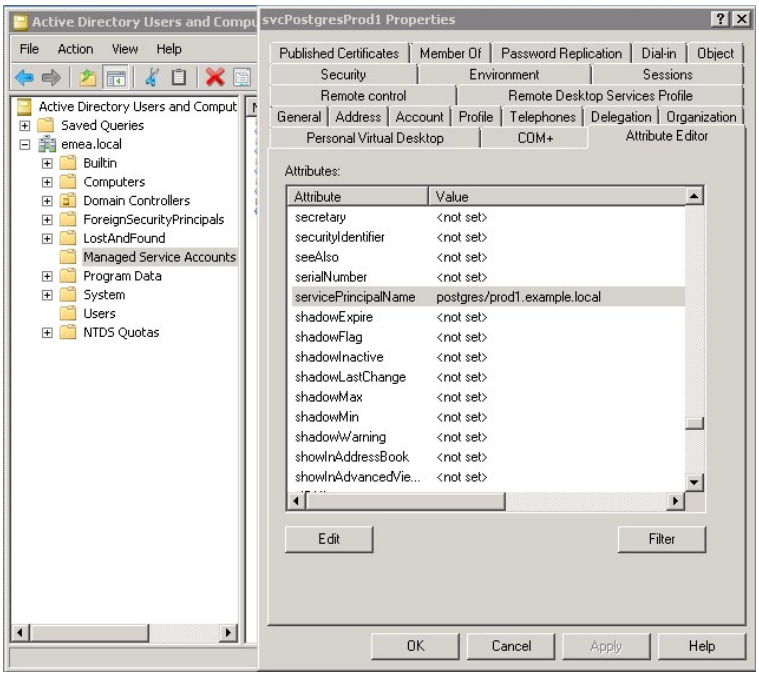
In this example, the AD password is set to never expire and cannot be changed by the user. The AD account password is only used when creating the Kerberos keytab file. There is no requirement to provide it to a database administrator.



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 ling argument.

This example command creates the keyab file `svcPostgresProd1.keytab`.

```
"c:\Program Files\Java\jre1.8.0_77\bin\ktab.exe" -a svcPostgresprod1 -k svcPostgresPro
d1.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.

Setting Up Greenplum Database 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.

```
mv 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:

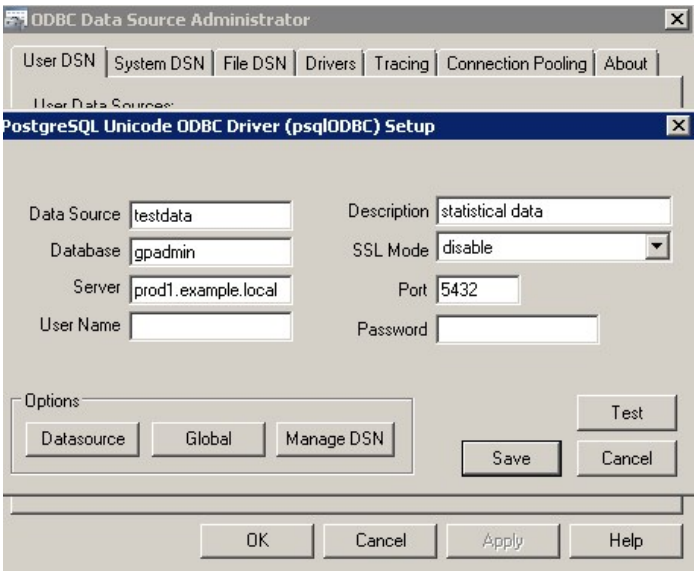
```
kinit dev1
psql -h prod1.example.local -U dev1
```

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.

```
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:

  ```
  klist -ket svcPostgresProd1.keytab
  ```

  To get the corresponding value from AD domain controller, run this command as an AD Administrator:

  ```
  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.

  ```
  2016-03-29 14:30:54.041151 PDT,"dev22","gpadmin",p15370,th2040321824,
    "172.28.9.181","49283",2016-03-29 14:30:53 PDT,1917,con32,,seg-1,,,x1917,sx1,
    "FATAL","28000","authentication failed for user ""dev22"": valid
    until timestamp expired",,,,,,,0,,"auth.c",628,
  ```

  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.

  ```
  psql -h 'hostname' postgres
  psql: GSSAPI continuation error: Unspecified GSS failure. Minor code may provid
  e 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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Managing Greenplum Database Access

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 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 1 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 1. Role Attributes

Attributes	Description
`SUPERUSER | NOSUPERUSER`	Determines if the role is a superuser. You must yourself be a superuser to create a new superuser. `NOSUPERUSER` is the default.
`CREATEDB | NOCREATEDB`	Determines if the role is allowed to create databases. `NOCREATEDB` is the default.
`CREATEROLE | NOCREATEROLE`	Determines if the role is allowed to create and manage other roles. `NOCREATEROLE` is the default.
`INHERIT | NOINHERIT`	Determines whether a role inherits the privileges of roles it is a member of. A role with the `INHERIT` attribute can automatically use whatever database privileges have been granted to all roles it is directly or indirectly a member of. `INHERIT` is the default.
`LOGIN | NOLOGIN`	Determines whether a role is allowed to log in. A role having the `LOGIN` attribute can be thought of as a user. Roles without this attribute are useful for managing database privileges (groups). `NOLOGIN` is the default.
`CONNECTION LIMIT` *connlimit*	If role can log in, this specifies how many concurrent connections the role can make. -1 (the default) means no limit.
`CREATEEXTTABLE | NOCREATEEXTTABLE`	Determines whether a role is allowed to create external tables. `NOCREATEEXTTABLE` is the default. For a role with the `CREATEEXTTABLE` attribute, the default external table `type` is `readable` and the default `protocol` is `gpfdist`. Note that external tables that use the `file` or `execute` protocols can only be created by superusers.
`PASSWORD '`*password*`'`	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`.
`ENCRYPTED | UNENCRYPTED`	Controls whether a new password is stored as a hash string in the `pg_authid` system catalog. If neither `ENCRYPTED` nor `UNENCRYPTED` is specified, the default behavior is determined by the `password_encryption` configuration parameter, which is `on` by default. If the supplied *password* string is already in hashed format, it is stored as-is, regardless of whether `ENCRYPTED` or `UNENCRYPTED` is specified. See Protecting Passwords in Greenplum Database for additional information about protecting login passwords.

Table 1. Role Attributes

Attributes	Description
`VALID UNTIL 'timestamp'`	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.
`RESOURCE QUEUE queue_name`	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.
`DENY {deny_interval | deny_point}`	Restricts access during an interval, specified by day or day and time. For more information see Time-based Authentication.

You can set these attributes when you create the role, or later using the `ALTER ROLE` command. For example:

```
=# 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:

```
=# 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 2). 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 2. Object Privileges

Object Type	Privileges

Tables, Views, Sequences	`SELECT`
	`INSERT`
	`UPDATE`
	`DELETE`
	`RULE`
	`ALL`

External Tables	`SELECT`
	`RULE`
	`ALL`

Databases	`CONNECT`
	`CREATE`
	`TEMPORARY | TEMP`
	`ALL`

Functions	`EXECUTE`
Procedural Languages	`USAGE`

Schemas	`CREATE`
	`USAGE`
	`ALL`

Custom Protocol	`SELECT`
	`INSERT`
	`UPDATE`
	`DELETE`
	`RULE`
	`ALL`

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:

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

To revoke privileges, use the `REVOKE` command. For example:

```
=# 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:

```
=# 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 is installed with an optional module 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*.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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**

- **Creating and Managing Tablespaces**
- **Creating and Managing Schemas**
- **Creating and Managing Tables**
- **Choosing the Table Storage Model**
- **Partitioning Large Tables**
- **Creating and Using Sequences**
- **Using Indexes in Greenplum Database**
- **Creating and Managing Views**

Parent topic: Greenplum Database Administrator Guide

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Defining Database Objects

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 templates, *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:

```
=> 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*:

```
$ 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:

```
=> 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:

```
=> 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:

```
=> 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):

```
=> 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:

```
=> \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*:

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

Warning: Dropping a database cannot be undone.

The `DROP DATABASE` command is not transactional.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Defining Database Objects

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.

   ```
   $ su - gpadmin
   ```

2. Create a filespace configuration file:

   ```
   $ 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:

   ```
   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:

   ```
   $ 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.

Warning: Do not move transaction files to a non-default location. Moving transaction files might cause data loss.

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 `gpfilespace` 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 `gpfilespace --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 gpfilespace

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.

   If any connections are still in progess, the `gpfilespace --movetempfiles` utility will fail.
3. Bring Greenplum Database online with no active session and run the following command:

   ```
   gpfilespace --movetempfilespace filespace_name
   ```

   The location of the temporary files is stored in the segment configuration shared memory (PMModuleState) and used whenever temporary 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:

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

Database superusers define tablespaces and grant access to database users with the `GRANT CREATE` command. For example:

```
=# 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 = space1;
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 pgts, pg_filespace pgfs,
          pg_filespace_entry pgfse
   WHERE  pgts.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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Defining Database Objects

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:

```
=> 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:

```
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:

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

Schema Search Paths

To specify an object's location in a database, use the schema-qualified name. For example:

```
=> 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:

```
=> 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:

```
=> 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:

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

Use the `SHOW` command to view the current search path. For example:

```
=> SHOW search_path;
```

Dropping a Schema

Use the `DROP SCHEMA` command to drop (delete) a schema. For example:

```
=> 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:

```
=> 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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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:

```
=> 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:

```
=> 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:

```
=> 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:

```
=> 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:

```
=> CREATE TABLE products
                         (name varchar(40),
                          prod_id integer,
                          supplier_id integer)
              DISTRIBUTED BY (prod_id);
```

```
=> CREATE TABLE random_stuff
                         (things text,
                          doodads text,
                          etc text)
              DISTRIBUTED RANDOMLY;
```

Important: 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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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*.

Parent topic: Defining Database Objects

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.

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

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:

```
=> 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:

  ```
  SELECT SUM(salary)...
  ```

  ```
  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:

  ```
  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:

```
=> 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 1. Compression Algorithms for Append-Optimized Tables

Table Orientation	Available Compression Types	Supported Algorithms
Row	Table	`ZLIB` and `QUICKLZ`[1]
Column	Column and Table	`RLE_TYPE`, `ZLIB`, and `QUICKLZ`[1]

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:

```
=> CREATE TABLE foo (a int, b text)
   WITH (appendonly=true, compresstype=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 2. Functions for compressed append-optimized table metadata

Function	Return Type	Description
get_ao_distribution(name) get_ao_distribution(oid)	Set of (dbid, tuplecount) rows	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.

Table 2. Functions for compressed append-optimized table metadata

Function	Return Type	Description
get_ao_compression_ratio(name) get_ao_compression_ratio(oid)	float8	Calculates the compression ratio for a compressed append-optimized table. If information is not available, this function returns a value of -1.

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 3. Storage Directives for Column-level Compression

Name	Definition	Values	Comment

COMPRESSTYPE	Type of compression.	zlib: deflate algorithm quicklz: fast compression RLE_TYPE: run-length encoding none: no compression	Values are not case-sensitive.
COMPRESSLEVEL	Compression level.	zlib compression: 1-9	1 is the fastest method with the least compression. 1 is the default. 9 is the slowest method with the most compression.
		QuickLZ compression: 1 – use compression	1 is the default.
		RLE_TYPE compression: 1 – 4 1 - apply RLE only 2 - apply RLE then apply zlib compression level 1 3 - apply RLE then apply zlib compression level 5 4 - apply RLE then apply zlib compression level 9	1 is the fastest method with the least compression. 4 is the slowest method with the most compression. 1 is the default.
BLOCKSIZE	The size in bytes for each block in the table	8192 – 2097152	The value must be a multiple of 8192.

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.

```
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.

```
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.

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

Example 4

In this example, `CREATE TABLE` assigns the zlib compresstype 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.

```
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)
                 )
   WITH (appendonly=true, orientation=column);
```

Example 5

This example creates an append-optimized, column-oriented table, T5. T5 has two partitions, `p1` and `p2`, each of which has subpartitions. Each subpartition has `ENCODING` clauses:

- The `ENCODING` clause for partition `p1`'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.

  ```
  CREATE TABLE T5(i int, j int, k int, l int)
      WITH (appendonly=true, orientation=column)
      PARTITION BY range(i) SUBPARTITION BY range(j)
      (
         p1 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:

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

You can then use `comptype` in a `CREATE TABLE` command to specify `quicklz` compression for a column:

```
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:

```
=> 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:

```
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:

```
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.

```
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:

```
CREATE TABLE sales2 (LIKE sales)
WITH (appendonly=true, compresstype=quicklz,
      compresslevel=1, orientation=column);
INSERT INTO sales2 SELECT * FROM sales;
DROP TABLE sales;
ALTER TABLE sales2 RENAME TO sales;
GRANT ALL PRIVILEGES ON sales TO admin;
GRANT SELECT ON sales TO guest;
```

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 1 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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Defining Database Objects

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 1. Example Multi-level Partition Design

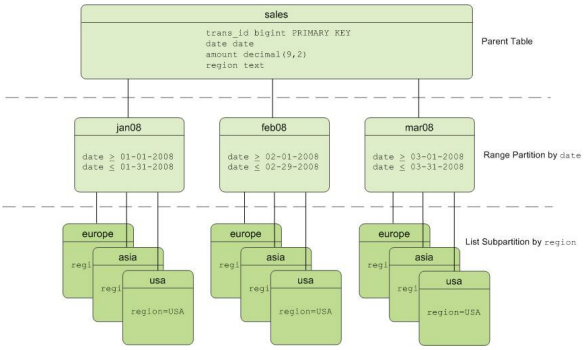


Table Partitioning in Greenplum Database

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 `SUBPARTITION 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:

```
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:

```
CREATE TABLE sales (id int, date date, amt decimal(10,2))
DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
( PARTITION Jan16 START (date '2016-01-01') INCLUSIVE ,
  PARTITION Feb16 START (date '2016-02-01') INCLUSIVE ,
  PARTITION Mar16 START (date '2016-03-01') INCLUSIVE ,
  PARTITION Apr16 START (date '2016-04-01') INCLUSIVE ,
  PARTITION May16 START (date '2016-05-01') INCLUSIVE ,
  PARTITION Jun16 START (date '2016-06-01') INCLUSIVE ,
  PARTITION Jul16 START (date '2016-07-01') INCLUSIVE ,
  PARTITION Aug16 START (date '2016-08-01') INCLUSIVE ,
  PARTITION Sep16 START (date '2016-09-01') INCLUSIVE ,
  PARTITION Oct16 START (date '2016-10-01') INCLUSIVE ,
  PARTITION Nov16 START (date '2016-11-01') INCLUSIVE ,
  PARTITION Dec16 START (date '2016-12-01') INCLUSIVE
                  END (date '2017-01-01') EXCLUSIVE );
```

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:

```
CREATE TABLE rank (id int, rank int, year int, gender
char(1), count int)
DISTRIBUTED BY (id)
PARTITION BY RANGE (year)
( START (2006) END (2016) EVERY (1),
  DEFAULT PARTITION extra );
```

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:

```
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 1:

```
CREATE TABLE sales (trans_id int, date date, amount
decimal(9,2), region text)
DISTRIBUTED BY (trans_id)
PARTITION BY RANGE (date)
SUBPARTITION BY LIST (region)
SUBPARTITION TEMPLATE
( SUBPARTITION usa VALUES ('usa'),
  SUBPARTITION asia VALUES ('asia'),
  SUBPARTITION europe VALUES ('europe'),
  DEFAULT SUBPARTITION other_regions)
  (START (date '2011-01-01') INCLUSIVE
   END (date '2012-01-01') EXCLUSIVE
   EVERY (INTERVAL '1 month'),
   DEFAULT PARTITION outlying_dates );
```

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 TEMPLATE (
               SUBPARTITION usa VALUES ('usa'),
               SUBPARTITION europe VALUES ('europe'),
               SUBPARTITION asia VALUES ('asia'),
               DEFAULT SUBPARTITION other_regions )
( START (2002) END (2012) EVERY (1),
  DEFAULT PARTITION outlying_years );
```

CAUTION:
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.

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 1. 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 onsales_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 onsales_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* system 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 also show information about partitioned tables.

- *pg_partition*- Tracks partitioned tables and their inheritance level relationships.
- *pg_partition_templates*- Shows the subpartitions created using a subpartition template.
- *pg_partition_columns* - Shows the partition key columns used in a partition design.

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. The given partition name is the `partitionname` column value in the *pg_partitions* system view. 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 partition: `PARTITION FOR (value)` or `PARTITION FOR (RANK(number))`.

For a multi-level partitioned table, you identify a specific partition to change with `ALTER PARTITION` clauses. For each partition level in the table hierarchy that is above the target partition, specify the partition that is related to the target partition in an `ALTER PARTITION` clause. For example, if you have a partitioned table that consists of three levels, year, quarter, and region, this `ALTER TABLE` command exchanges a leaf partition `region` with the table `region_new`.

```
ALTER TABLE sales ALTER PARTITION year_1 ALTER PARTITION quarter_4 EXCHANGE PARTITION regio
n WITH TABLE region_new ;
```

The two `ALTER PARTITION` clauses identify which `region` partition to exchange. Both clauses are required to identify the specific leaf partition to exchange.

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.

```
<parentname>_<level>_prt_<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 *<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).

Exchanging a partition with a partitioned table or a child partition of a partitioned table is not

supported.

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). For a multi-level partition, 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:

```
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:

```
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:

```
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.

```
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`:

```
ALTER TABLE sales ADD PARTITION "4"
  START ('2014-04-01') INCLUSIVE
  END ('2014-05-01') EXCLUSIVE ;
```

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 if 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 EXTENAL 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 EXTENAL TABLE` create a readable external that uses the same external data as the writable external data.

   ```
   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.

   ```
   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.

   ```
   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.

   ```
   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`.

```
ALTER TABLE sales RENAME PARTITION yr_1 TO  yr_1_ext ;
```

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 squence. 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.

Parent topic: Defining Database Objects

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:

```
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`:

```
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`:

```
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`:

```
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;
```

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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;
```

Parent topic: Defining Database Objects

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

```
REINDEX my_table;
```

```
REINDEX my_index;
```

Dropping an Index

The `DROP INDEX` command removes an index. For example:

```
DROP INDEX title_idx;
```

When loading data, it can be faster to drop all indexes, load, then recreate the indexes.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Defining Database Objects

Creating Views

The `CREATE VIEW` command defines a view of a query. For example:

```
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:

```
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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Distribution and Skew

Greenplum Database relies on even distribution of data across segments.

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.

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.

Parent topic: Greenplum Database Administrator Guide

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 may be caused by uneven data distribution due to the wrong choice of distribution keys or single tuple table insert or copy operations. Present at the table level, 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:

```
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 `siffraction` 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 than10% 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.

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:

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

   Example output:

```
  oid  |  datname
-------+-----------
 17088 | gpadmin
 10899 | postgres
     1 | template1
 10898 | template0
 38817 | pws
 39682 | gpperfmon
(6 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 \
    { for ( i in arr ) print "Segment node" i, arr[i], "bytes (" arr[i]/(1024**
3)" GB)"; \
    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[sdw1] 291176
448
      /data2/primary/gpseg5/.../pgsql_tmp_slice0_sort_17764_0001.0[sdw8] 924581
888
      /data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_0010.9[sdw4] 9805
82400
      /data1/primary/gpseg18/.../pgsql_tmp_slice10_sort_29425_0001.0[sdw6] 9864
47872
      /data2/primary/gpseg35/.../pgsql_tmp_slice10_sort_29602_0001.0...[sdw5] 9
99620608
      /data1/primary/gpseg26/.../pgsql_tmp_slice10_sort_28637_0001.0[sdw2] 9997
51680
```

```
      /data2/primary/gpseg9/.../pgsql_tmp_slice10_sort_3969_0001.0[sdw3] 100011
2128
      /data1/primary/gpseg13/.../pgsql_tmp_slice10_sort_24723_0001.0[sdw5] 1000
898560
      /data2/primary/gpseg28/.../pgsql_tmp_slice10_sort_28641_0001.0...[sdw8] 1
008009216
      /data1/primary/gpseg44/.../pgsql_tmp_slice10_sort_15671_0001.0[sdw5] 1008
566272
      /data1/primary/gpseg24/.../pgsql_tmp_slice10_sort_28633_0001.0[sdw4] 1009
451008
      /data1/primary/gpseg19/.../pgsql_tmp_slice10_sort_29427_0001.0[sdw7] 1011
187712
      /data1/primary/gpseg37/.../pgsql_tmp_slice10_sort_18526_0001.0[sdw8] 1573
741824
      /data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_0001.0[sdw8] 1573
741824
      /data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_0002.1[sdw8] 1573
741824
      /data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_0003.2[sdw8] 1573
741824
      /data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_0004.3[sdw8] 1573
741824
      /data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_0005.4[sdw8] 1573
741824
      /data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_0006.5[sdw8] 1573
741824
      /data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_0007.6[sdw8] 1573
741824
      /data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_0008.7[sdw8] 1573
741824
      /data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_0009.8
```

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_slice
   10_sort_15673_0002.1
   COMMAND  PID    USER    FD   TYPE DEVICE  SIZE        NODE        NAME
   postgres 15673  gpadmin 11u  REG  8,48    1073741824  64424546751 /data2/primar
   y/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
   gpadmin  15673 27471 28 12:05 ?        00:12:59 postgres: port 40003, sbaskin b
   dw
           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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Inserting, Updating, and Deleting Data

This section provides information about manipulating data and concurrent access in Greenplum Database.

This topic includes the following subtopics:

Parent topic: Greenplum Database Administrator Guide

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 1. Lock Modes in Greenplum Database

Lock Mode	Associated SQL Commands	Conflicts With
ACCESS SHARE	`SELECT`	ACCESS EXCLUSIVE
ROW SHARE	`SELECT FOR SHARE`, `SELECT FOR UPDATE`	EXCLUSIVE, ACCESS EXCLUSIVE
ROW EXCLUSIVE	`INSERT`, `COPY` `UPDATE`, `DELETE` on non-partitioned tables, see Note.	SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE
SHARE UPDATE EXCLUSIVE	`VACUUM` (without `FULL`), `ANALYZE`	SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE
SHARE	`CREATE INDEX`	ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE
SHARE ROW EXCLUSIVE		ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE
EXCLUSIVE	`SELECT FOR UPDATE` `DELETE`, `UPDATE` on partitioned tables. See Note.	ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE

Table 1. Lock Modes in Greenplum Database

Lock Mode	Associated SQL Commands	Conflicts With
ACCESS EXCLUSIVE	`ALTER TABLE`, `DROP TABLE`, `TRUNCATE`, `REINDEX`, `CLUSTER`, `VACUUM FULL`	ACCESS SHARE, ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE

Note: Greenplum Database acquires an `EXCLUSIVE` lock for `SELECT FOR UPDATE`. PostgreSQL acquires a less restrictive `ROW EXCLUSIVE` lock.

For an `UPDATE` or `DELETE` command on a partitioned table, Greenplum Database acquires an `EXCLUSIVE` lock on the root partition table. On a non-partitioned table, Greenplum Database acquires a `ROW EXCLUSIVE` lock.

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:

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

To specify only the values to insert:

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

Usually, the data values are literals (constants), but you can also use scalar expressions. For example:

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

You can insert multiple rows in a single command. For example:

```
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:

- While GPORCA supports updates to Greenplum distribution key columns, the Postgres planner does not.
- 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`

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.
- **About GPORCA**
  In Greenplum Database, the default GPORCA optmizer co-exists with the legacy query optimizer.

- **Defining Queries**
  Greenplum Database is based on the PostgreSQL implementation of the SQL standard.
- **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.
- **Using Functions and Operators**
  Description of user-defined and built-in functions and operators in Greenplum Database.
- **Working with JSON Data**
  Greenplum Database supports the `json` data type that stores JSON (JavaScript Object Notation) data.
- **Working with XML Data**
  Greenplum Database supports the `xml` data type that stores XML data.
- **Query Performance**
  Greenplum Database dynamically eliminates irrelevant partitions in a table and optimally allocates memory for different operators in a query.
- **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.
- **Query Profiling**
  Examine the query plans of poorly performing queries to identify possible performance tuning opportunities.

Parent topic: Greenplum Database Administrator Guide

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.

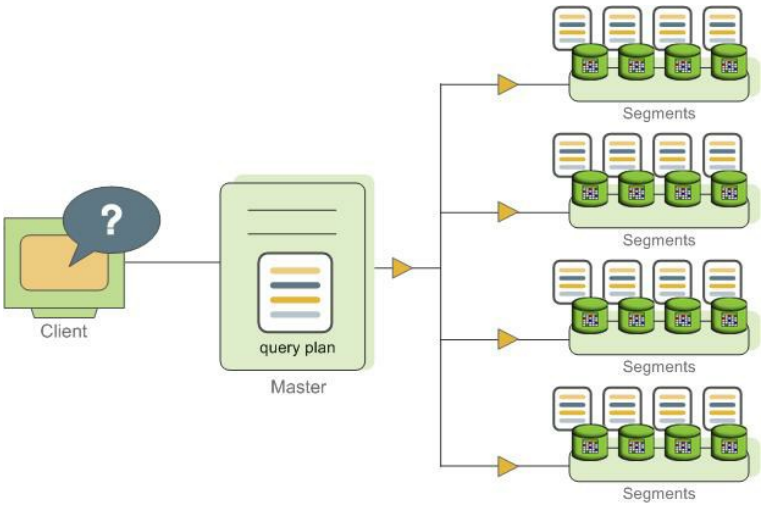
Parent topic: Querying Data

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 1. The master dispatches targeted query plans to a single segment, as shown in Figure 2. Each segment is responsible for executing local database operations on its own set of data.

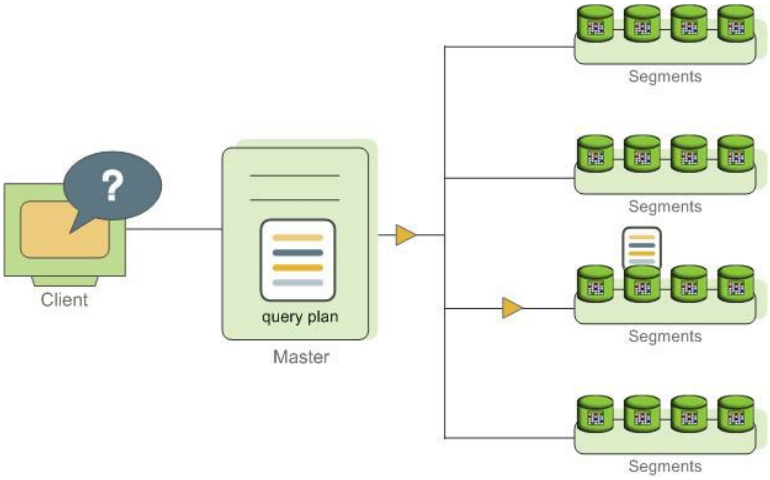
Most database operations—such as table scans, joins, aggregations, and sorts—execute across all segments in parallel. Each operation is performed on a segment database independent of the data stored in the other segment databases.

Figure 1. Dispatching the Parallel Query Plan



Certain queries may access only data on a single segment, such as single-row `INSERT`, `UPDATE`, `DELETE`, or `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 2. Dispatching a Targeted Query Plan



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 table scans, joins, and so on, Greenplum Database has an additional operation type called *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 *slices*. A slice is a portion of the plan that segments can work on independently. A query plan is sliced wherever a *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:

```
SELECT customer, amount
FROM sales JOIN customer USING (cust_id)
WHERE dateCol = '04-30-2016';
```

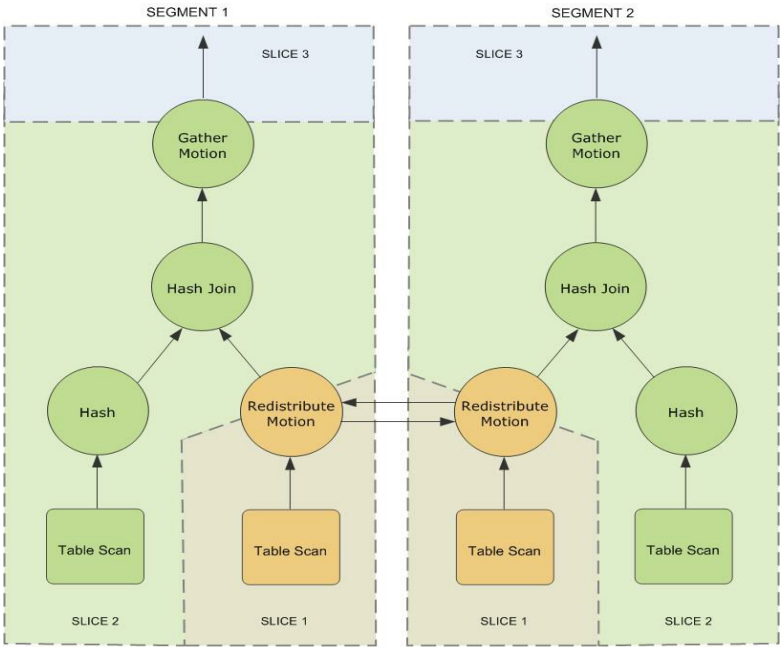
Figure 3 shows the query plan. Each segment receives a copy of the query plan and works on it in

parallel.

The query plan for this example has a *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 `cust_id`, but the sales table is distributed across the segments by `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.

Figure 3. Query Slice Plan



Understanding Parallel Query Execution

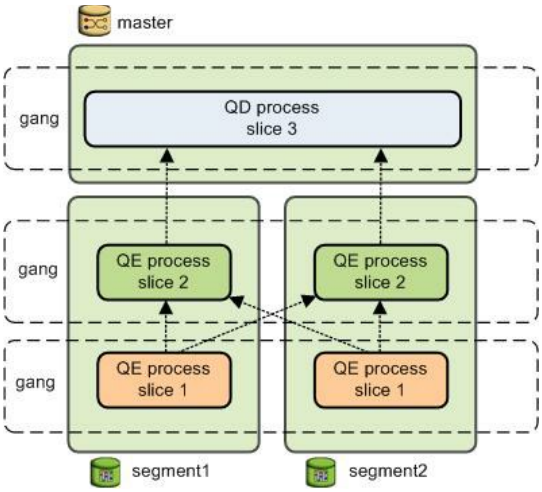
Greenplum creates a number of database processes to handle the work of a query. On the master, the query worker process is called the *query dispatcher* (QD). The QD is responsible for creating and dispatching the query plan. It also accumulates and presents the final results. On the segments, a query 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 4 shows the query worker processes on the master and two segment instances for the query plan illustrated in Figure 3.

Figure 4. Query Worker Processes



A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

About GPORCA

In Greenplum Database, the default GPORCA optmizer co-exists with the legacy query optimizer.

These sections describe GPORCA functionality and usage:

- **Overview of GPORCA**
  GPORCA extends the planning and optimization capabilities of the Greenplum Database legacy optimizer.
- **Enabling and Disabling GPORCA**
  By default, Greenplum Database uses GPORCA instead of the legacy query planner. Server configuration parameters enable or disable GPORCA.
- **Collecting Root Partition Statistics**
  For a partitioned table, GPORCA uses statistics of the table root partition to generate query plans. These statistics are used for determining the join order, for splitting and joining aggregate nodes, and for costing the query steps. In contrast, the legacy planner uses the statistics of each leaf partition.
- **Considerations when Using GPORCA**
  To execute queries optimally with GPORCA, query criteria to consider.
- **GPORCA Features and Enhancements**
  GPORCA, the Greenplum next generation query optimizer, includes enhancements for specific types of queries and operations:
- **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.
- **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.
- **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.
- **About Uniform Multi-level Partitioned Tables**

Parent topic: Querying Data

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

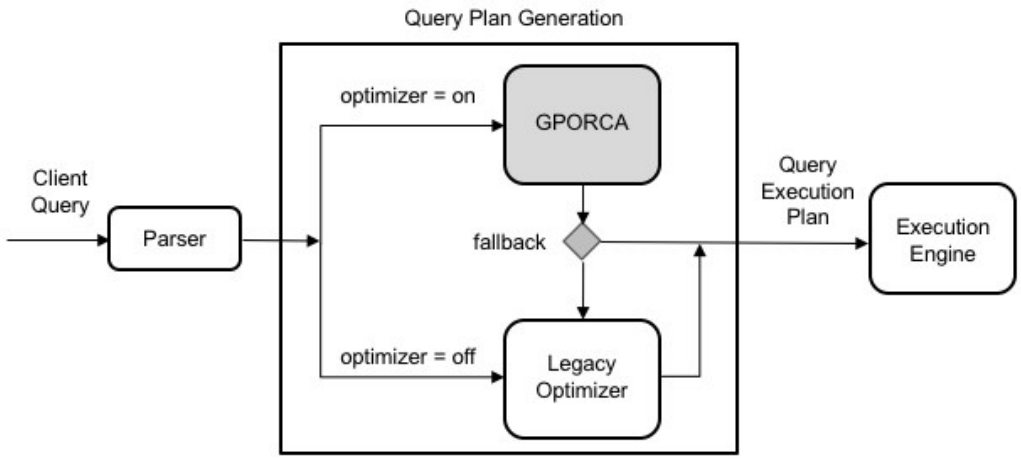
Overview of GPORCA

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.



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.
Parent topic: About GPORCA

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Enabling and Disabling GPORCA

By default, Greenplum Database uses GPORCA instead of the legacy query planner. Server configuration parameters enable or disable GPORCA.
Although GPORCA is on by default, you can configure GPORCA usage at the system, database, session, or query level using the optimizer parameter. Refer to one of the following sections if you want to change the default behavior:

- Enabling GPORCA for a System
- Enabling GPORCA for a Database
- Enabling GPORCA for a Session or a 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*.
Parent topic: About GPORCA

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`:

   ```
   $ 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.

   ```
   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*.

```
> 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:

```
> set optimizer = on ;
```

To set the parameter for a specific query, include the SET command prior to running the query.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Collecting Root Partition Statistics

For a partitioned table, GPORCA uses statistics of the table root partition to generate query plans. These statistics are used for determining the join order, for splitting and joining aggregate nodes, and for costing the query steps. In contrast, the legacy planner uses the statistics of each leaf partition.

If you execute queries on partitioned tables, you should collect statistics on the root partition and periodically update those statistics to ensure that GPORCA can generate optimal query plans. If the root partition statistics are not up-to-date or do not exist, GPORCA still performs dynamic partition elimination for queries against the table. However, the query plan might not be optimal.

Parent topic: About GPORCA

Running ANALYZE

By default, running the `ANALYZE` command on the root partition of a partitioned table samples the leaf partition data in the table, and stores the statistics for the root partition. `ANALYZE` collects statistics on the root and leaf partitions, including HyperLogLog (HLL) statistics on the leaf partitions. `ANALYZE ROOTPARTITION` collects statistics only on the root partition. The server configuration parameter `optimizer_analyze_root_partition` controls whether the `ROOTPARTITION` keyword is required to collect root statistics for the root partition of a partitioned table. See the `ANALYZE` command for information about collecting statistics on partitioned tables.

Keep in mind that `ANALYZE` always scans the entire table before updating the root partition statistics. If your table is very large, this operation can take a significant amount of time. `ANALYZE ROOTPARTITION` also uses an `ACCESS SHARE` lock that prevents certain operations, such as `TRUNCATE` and `VACUUM` operations, during execution. For these reasons, you should schedule `ANALYZE` operations periodically, or when there are significant changes to leaf partition data.

Follow these best practices for running `ANALYZE` or `ANALYZE ROOTPARTITION` on partitioned tables in your system:

- Run `ANALYZE <root_partition>` on a new partitioned table after adding initial data. Run `ANALYZE <leaf_partition>` on a new leaf partition or a leaf partition where data has changed. By default, running the command on a leaf partition updates the root partition statistics if the other leaf partitions have statistics.
- Update root partition statistics when you observe query performance regression in `EXPLAIN` plans against the table, or after significant changes to leaf partition data. For example, if you add a new leaf partition at some point after generating root partition statistics, consider running `ANALYZE` or `ANALYZE ROOTPARTITION` to update root partition statistics with the new tuples inserted from the new leaf partition.
- For very large tables, run `ANALYZE` or `ANALYZE ROOTPARTITION` only weekly, or at some interval longer than daily.
- Avoid running `ANALYZE` with no arguments, because doing so executes the command on all database tables including partitioned tables. With large databases, these global `ANALYZE` operations are difficult to monitor, and it can be difficult to predict the time needed for completion.
- Consider running multiple `ANALYZE <table_name>` or `ANALYZE ROOTPARTITION <table_name>` operations in parallel to speed the operation of statistics collection, if your I/O throughput can support the load.
- You can also use the Greenplum Database utility `analyzedb` to update table statistics. Using `analyzedb` ensures that tables that were previously analzyed are not re-analyzed if no modifications were made to the leaf partition.

GPORCA and Leaf Partition Statistics

Although creating and maintaining root partition statistics is crucial for GPORCA query performance with partitioned tables, maintaining leaf partition statistics is also important. If GPORCA cannot generate a plan for a query against a partitioned table, then the legacy planner is used and leaf partition statistics are needed to produce the optimal plan for that query.

GPORCA itself also uses leaf partition statistics for any queries that access leaf partitions directly, instead of using the root partition with predicates to eliminate partitions. For example, if you know which partitions hold necessary tuples for a query, you can directly query the leaf partition table itself; in this case GPORCA uses the leaf partition statistics.

Disabling Automatic Root Partition Statistics Collection

If you do not intend to execute queries on partitioned tables with GPORCA (setting the server configuration parameter `optimizer` to `off`), then you can disable the automatic collection of statistics on the root partition of the partitioned table. The server configuration parameter `optimizer_analyze_root_partition` controls whether the `ROOTPARTITION` keyword is required to collect root statistics for the root partition of a partitioned table. The default setting for the parameter is `on`, the `ANALYZE` command can collect root partition statistics without the `ROOTPARTITION` keyword. You can disable automatic collection of root partition statistics by setting the parameter to `off`. When the value is `off`, you must run `ANALZYE ROOTPARTITION` to collect root partition statistics.

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 `off`:

   ```
   $ gpconfig -c optimizer_analyze_root_partition -v off --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.

   ```
   gpstop -u
   ```

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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` sets the query optimization level for join ordering by specifying which types of join ordering alternatives to evaluate.
- `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.

- `gp_enable_relsize_collection` controls how GPORCA (and the legacy query optimizer) handle a table without statistics. By default, GPORCA uses a default value to estimate the number of rows if statistics are not available. When this value is `on`, GPORCA uses the estimated size of a table if there are no statistics for the table.

  This parameter is ignored for a root partition of a partitioned table. If the root partition does not have statistics, GPORCA always uses the default value. You can use `ANALZYE ROOTPARTITION` to collect statistics on the root partition. See `ANALYZE`.

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;
```

Parent topic: About GPORCA

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: About GPORCA

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 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.

  ```
  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.

```
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.

```
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.

```
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.

```
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
```

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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*.

Parent topic: About GPORCA

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

Parent topic: About GPORCA

Unsupported SQL Query Features

Certain query features are not supported with the default GPORCA optimizer. When an unsupported query is executed, Greenplum logs this notice along with the query text:

```
Feature not supported by the Pivotal Query Optimizer: UTILITY command
```

These features are unsupported when GPORCA is enabled (the default):

- Prepared statements that have parameterized values.
- Indexed expressions (an index defined as expression based on one or more columns of the table)
- GIN indexing method. GPORCA supports only B-tree, bitmap, and GiST indexes. GPORCA ignores indexes created with unsupported methods.
- 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
- Aggregate functions that take set operators as input arguments.
- percentile_* window functions (Greenplum Database does not support ordered-set aggregate functions).
- Inverse distribution functions.
- Queries that contain UNICODE characters in metadata names, such as table names, and the characters are not compatible with the host system locale.
- `SELECT`, `UPDATE`, and `DELETE` commands where a table name is qualified by the `ONLY` keyword.

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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.
Parent topic: About GPORCA

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:

```
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 wi
dth=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: Queries on master-only tables"
```

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: About GPORCA

Example

This `CREATE TABLE` command creates a uniform partitioned table.

```
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.

```
mlp_1_prt_11
   mlp_1_prt_11_2_prt_usa
   mlp_1_prt_11_2_prt_europe
   mlp_1_prt_11_2_prt_asia

mlp_1_prt_21
   mlp_1_prt_21_2_prt_usa
   mlp_1_prt_21_2_prt_europe
   mlp_1_prt_21_2_prt_asia
```

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.

```
WITH tbl AS (SELECT oid, partitionlevel AS level,
             partitiontablename AS part
         FROM pg_partitions, pg_class
         WHERE tablename = 'mlp' AND partitiontablename=relname
            AND partitionlevel=1 )
  SELECT tbl.part, consrc
    FROM tbl, pg_constraint
    WHERE tbl.oid = conrelid ORDER BY consrc;
```

Note: 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.

```
           part           |               consrc
--------------------------+-----------------------------------
 mlp_1_prt_2_2_prt_asia   | (region = 'asia'::text)
 mlp_1_prt_1_2_prt_asia   | (region = 'asia'::text)
 mlp_1_prt_2_2_prt_europe | (region = 'europe'::text)
 mlp_1_prt_1_2_prt_europe | (region = 'europe'::text)
 mlp_1_prt_1_2_prt_usa    | (region = 'usa'::text)
 mlp_1_prt_2_2_prt_usa    | (region = 'usa'::text)
```

```
 mlp_1_prt_1_2_prt_asia   | ((year >= 2006) AND (year < 2011))
 mlp_1_prt_1_2_prt_usa    | ((year >= 2006) AND (year < 2011))
 mlp_1_prt_1_2_prt_europe | ((year >= 2006) AND (year < 2011))
 mlp_1_prt_2_2_prt_usa    | ((year >= 2011) AND (year < 2016))
 mlp_1_prt_2_2_prt_asia   | ((year >= 2011) AND (year < 2016))
 mlp_1_prt_2_2_prt_europe | ((year >= 2011) AND (year < 2016))
(12 rows)
```

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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Querying Data

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 SQL Command Reference.

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:

```
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 Summary of Built-in Functions 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 4 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 2.
- Greenplum Database does not support DISTINCT with multiple input expressions.
- Greenplum Database does not support specifying an aggregate function as an argument to another aggregate function.
- Greenplum Database does not support specifying a window function as an argument to an aggregate function.

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.

Greenplum Database does not support specifying a window function as an argument to another window function.

The syntax of a window expression is:

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

Where *window_function* is one of the functions listed in Table 3, *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:

```
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. 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.

Greenplum Database supports three types of casts applied to a value expression:

- *Explicit cast* - Greenplum Database applies a cast when you explicitly specify a cast between two data types. Greenplum Database accepts two equivalent syntaxes for explicit type casts:

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

  The `CAST` syntax conforms to SQL; the syntax with `::` is historical PostgreSQL usage.

- *Assignment cast* - Greenplum Database implicitly invokes a cast in assignment contexts, when assigning a value to a column of the target data type. For example, a `CREATE CAST` command with the `AS ASSIGNMENT` clause creates a cast that is applied implicitly in the assignment context. This example assignment cast assumes that `tbl1.f1` is a column of type `text`. The `INSERT` command is allowed because the value is implicitly cast from the `integer` to `text` type.

  ```
  INSERT INTO tbl1 (f1) VALUES (42);
  ```

- *Implicit cast* - Greenplum Database implicitly invokes a cast in assignment or expression contexts. For example, a `CREATE CAST` command with the `AS IMPLICIT` clause creates an implicit cast, a cast that is applied implicitly in both the assignment and expression context. This example implicit cast assumes that `tbl1.c1` is a column of type `int`. For the calculation in the predicate, the value of `c1` is implicitly cast from `int` to a `decimal` type.

```
SELECT * FROM tbl1 WHERE tbl1.c2 = (4.3 + tbl1.c1) ;
```

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. Greenplum Database implicitly applies casts only to casts where the is defined with the cast context of assignment or implicit in the system catalogs. Other casts must be invoked with explicit casting syntax to prevent unexpected conversions from being applied without the user's knowledge.

You can display cast information with the `psql` metacommand `\dC`. Cast information is stored in the catalog table `pg_cast`, and type information is stored in the catalog table `pg_type`.

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

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

Example 2 – Correlated EXISTS subquery

```
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

```
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

```
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

```
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

```
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.

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
-----------------------------------------------
 {{{1,2},{3,4}},{{5,6},{7,8}},{{9,10},{11,12}}}
```

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%');
                           ?column?
------------------------------------------------------------
 {2011,1954,1948,1952,1951,1244,1950,2005,1949,1953,2006,31}
```

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'));
ERROR:  function getf1(record) is not unique
```

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

```
myrowtype));
 getf1
-------
    11
```

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:

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

`somefunc()` would probably not be called at all. The same is true in the following expression:

```
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:

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

The following example shows a trustworthy evaluation order:

```
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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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:

```
WITH regional_sales AS (
     SELECT region, SUM(amount) AS total_sales
     FROM orders
     GROUP BY region
  ), top_regions AS (
     SELECT region
     FROM regional_sales
     WHERE total_sales > (SELECT SUM(total_sales)/10 FROM regional_sales)
  )
```

```
SELECT region,
    product,
    SUM(quantity) AS product_units,
    SUM(amount) AS product_sales
FROM orders
WHERE region IN (SELECT region FROM top_regions)
GROUP BY region, product;
```

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`.
Note: The `RECURSIVE` keyword is a Beta feature.

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:

```
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.

```
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:

```
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:

```
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.

Parent topic: Querying Data

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Querying Data

Using Functions in Greenplum Database

Table 1. Functions in Greenplum Database

Function Type	Greenplum Support	Description	Comments
IMMUTABLE	Yes	Relies only on information directly in its argument list. Given the same argument values, always returns the same result.	
STABLE	Yes, in most cases	Within a single table scan, returns the same result for same argument values, but results change across SQL statements.	Results depend on database lookups or parameter values. `current_timestamp` family of functions is `STABLE`; values do not change within an execution.
VOLATILE	Restricted	Function values can change within a single table scan. For example: `random()`, `timeofday()`.	Any function with side effects is volatile, even if its result is predictable. For example: `setval()`.

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):

```
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:

```
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 `PREPARE`d 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 2. Built-in functions and operators

Operator/Function Category	VOLATILE Functions	STABLE Functions	Restrictions
Logical Operators			
Comparison Operators			
Mathematical Functions and Operators	random setseed		
String Functions and Operators	*All built-in conversion functions*	convert pg_client_encoding	
Binary String Functions and Operators			
Bit String Functions and Operators			
Pattern Matching			
Data Type Formatting Functions		to_char to_timestamp	
Date/Time Functions and Operators	timeofday	age current_date current_time current_timestamp localtime localtimestamp now	
Enum Support Functions			
Geometric Functions and Operators			
Network Address Functions and Operators			
Sequence Manipulation Functions	nextval() setval()		
Conditional Expressions			
Array Functions and Operators		*All array functions*	
Aggregate Functions			
Subquery Expressions			
Row and Array Comparisons			
Set Returning Functions	generate_series		

Table 2. Built-in functions and operators

Operator/Function Category	VOLATILE Functions	STABLE Functions	Restrictions
System Information Functions		*All session information functions* *All access privilege inquiry functions* *All schema visibility inquiry functions* *All system catalog information functions* *All comment information functions* *All transaction ids and snapshots*	
System Administration Functions	set_config pg_cancel_backend pg_terminate_backend pg_reload_conf pg_rotate_logfile pg_start_backup pg_stop_backup pg_size_pretty pg_ls_dir pg_read_file pg_stat_file	current_setting *All database object size functions*	**Note:** The function `pg_column_size` displays bytes required to store the value, possibly with TOAST compression.
XML Functions and function-like expressions		cursor_to_xml(cursor refcursor, count int, nulls boolean, tableforest boolean, targetns text) cursor_to_xmlschema(cursor refcursor, nulls boolean, tableforest boolean, targetns text) database_to_xml(nulls boolean, tableforest boolean, targetns text) database_to_xmlschema(nulls boolean, tableforest boolean, targetns text) database_to_xml_and_xmlschema(nulls boolean, tableforest boolean, targetns text) query_to_xml(query text, nulls boolean, tableforest boolean, targetns text) query_to_xmlschema(query text, nulls boolean, tableforest boolean, targetns text) query_to_xml_and_xmlschema(query text, nulls boolean, tableforest boolean, targetns text) schema_to_xml(schema name, nulls boolean, tableforest boolean, targetns text) schema_to_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text) schema_to_xml_and_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)	

Table 2. Built-in functions and operators

Operator/Function Category	VOLATILE Functions	STABLE Functions	Restrictions
		table_to_xml(tbl regclass, nulls boolean, tableforest boolean, targetns text) table_to_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text) 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 | CONTENT } value) xpath(text, xml) xpath(text, xml, text[]) xpath_exists(text, xml) xpath_exists(text, xml, text[]) xmlpi(name target [, content]) xmlroot(xml, version text | no value [, standalone yes|no|no value]) xmlserialize ({ DOCUMENT | CONTENT } value AS type) xml(text) text(xml) xmlcomment(xml) xmlconcat2(xml, xml)	

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 3. Window functions

Function	Return Type	Full Syntax	Description
`cume_dist()`	`double precision`	`CUME_DIST() OVER ( [PARTITION BY` expr `] ORDER BY` expr `)`	Calculates the cumulative distribution of a value in a group of values. Rows with equal values always evaluate to the same cumulative distribution value.

Table 3. Window functions

Function	Return Type	Full Syntax	Description
`dense_rank()`	`bigint`	`DENSE_RANK () OVER ( [PARTITION BY` expr `] ORDER BY` expr `)`	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.
`first_value(`*expr*`)`	same as input expr type	`FIRST_VALUE(` expr `) OVER ( [PARTITION BY` expr `] ORDER BY` expr `[ROWS|RANGE` frame_expr `] )`	Returns the first value in an ordered set of values.
`lag(`*expr* `[,`*offset*`] [,`*default*`])`	same as input *expr* type	`LAG(` *expr* `[,` *offset* `] [,` *default* `]) OVER ( [PARTITION BY` *expr* `] ORDER BY` *expr* `)`	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.
`last_value(`*expr*`)`	same as input *expr* type	`LAST_VALUE(`*expr*`) OVER ( [PARTITION BY` *expr*`] ORDER BY` *expr* `[ROWS|RANGE` *frame_expr*`] )`	Returns the last value in an ordered set of values.
`lead(`*expr* `[,`*offset*`] [,`*default*`])`	same as input *expr* type	`LEAD(`*expr* `[,`*offset*`] [,`*exprdefault*`]) OVER ( [PARTITION BY` *expr*`] ORDER BY` *expr* `)`	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.
`ntile(`*expr*`)`	`bigint`	`NTILE(`*expr*`) OVER ( [PARTITION BY` *expr*`] ORDER BY` *expr* `)`	Divides an ordered data set into a number of buckets (as defined by *expr*) and assigns a bucket number to each row.
`percent_rank()`	`double precision`	`PERCENT_RANK () OVER ( [PARTITION BY` *expr*`] ORDER BY` *expr* `)`	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).

Table 3. Window functions

Function	Return Type	Full Syntax	Description
`rank()`	`bigint`	`RANK () OVER ( [PARTITION BY expr] ORDER BY expr )`	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.
`row_number()`	`bigint`	`ROW_NUMBER () OVER ( [PARTITION BY expr] ORDER BY expr )`	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).

Advanced Aggregate Functions

The following built-in advanced aggregate functions are Greenplum extensions of the PostgreSQL database. These functions are *immutable*. Greenplum Database does not support the PostgreSQL ordered-set aggregate functions.

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 4. Advanced Aggregate Functions

Function	Return Type	Full Syntax	Description

`MEDIAN (expr)`	`timestamp, timestamptz, interval, float`	`MEDIAN (expression)`
		Example:
		`SELECT department_id, MEDIAN(salary) FROM employees GROUP BY department_id;`
			Can take a two-dimensional array as input. Treats such arrays as matrices.

`sum(array[])`	`smallint[], int[], bigint[], float[]`	`sum(array[[1,2],[3,4]])`
		Example:
		`CREATE TABLE mymatrix ( myvalue int[]); INSERT INTO mymatrix VALUES (array[[1,2], [3,4]]); INSERT INTO mymatrix VALUES (array[[0,1], [1,0]]); SELECT sum(myvalue) FROM mymatrix; sum --------------- {{1,3},{4,4}}`
			Performs matrix summation. Can take as input a two-dimensional array that is treated as a matrix.

`pivot_sum (label[], label, expr)`	`int[], bigint[], float[]`	`pivot_sum( array['A1','A2'], attr, value)`	A pivot aggregation using sum to resolve duplicate entries.

Table 4. Advanced Aggregate Functions

Function	Return Type	Full Syntax	Description
`unnest (array[])`	set of `anyelement`	`unnest( array['one', 'row', 'per', 'item'])`	Transforms a one dimensional array into rows. Returns a set of `anyelement`, a polymorphic pseudotype in PostgreSQL.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Querying Data

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 `\u`*`XXXX`*. 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 1. *json* Operators

Operator	Right Operand Type	Description	Example	Example Result
`->`	`int`	Get JSON array element (indexed from zero).	`'[{"a":"foo"}, {"b":"bar"}, {"c":"baz"}]'::json->2`	`{"c":"baz"}`
`->`	`text`	Get JSON object field by key.	`'{"a": {"b":"foo"}}'::json->'a'`	`{"b":"foo"}`
`->>`	`int`	Get JSON array element as text.	`'[1,2,3]'::json->>2`	`3`
`->>`	`text`	Get JSON object field as text.	`'{"a":1,"b":2}'::json->>'b'`	`2`
`#>`	`text[]`	Get JSON object at specified path.	`'{"a": {"b":{"c": "foo"}}}'::json#>'{a,b}'`	`{"c": "foo"}`

Table 1. *json* Operators

Operator	Right Operand Type	Description	Example	Example Result
`#>>`	`text[]`	Get JSON object at specified path as text.	`'{"a":[1,2,3],"b":[4,5,6]}'::json#>>'{a,2}'`	`3`

JSON Creation Functions

This table describes the functions that create `json` values.

Table 2. JSON Creation Functions

Function	Description	Example	Example Result
`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 3. JSON Processing Functions

Function	Return Type	Description	Example	Example Result

`json_each(json)`	`setof key text, value json`
	`setof key text, value jsonb`
		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`

Table 3. JSON Processing Functions

Function	Return Type	Description	Example	Example Result
`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":{"f3":"a", "f4":"b"}}')`	`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`

`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`.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Querying Data

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>...</chap
ter></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`:

```
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:

```
SET XML OPTION { DOCUMENT | CONTENT };
```

or simply like Greenplum Database:

```
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. This includes string representations of XML values, such as in the above examples. 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:

```
SELECT xpath('/my:a/text()', '<my:a xmlns:my="http://example.com">test</my:a>',
             ARRAY[ARRAY['my', 'http://example.com']]);

 xpath
--------
 {test}
(1 row)
```

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:

```
SELECT xpath_exists('/my:a/text()', '<my:a xmlns:my="http://example.com">test</my:a>',
                    ARRAY[ARRAY['my', 'http://example.com']]);

 xpath_exists
--------------
 t
(1 row)
```

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:

```
<tablename>
  <row>
    <columnname1>data</columnname1>
    <columnname2>data</columnname2>
  </row>

  <row>
    ...
  </row>

  ...
</tablename>
```

If `tableforest` is true, the result is an XML content fragment that looks like this:

```
<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:

```
<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:

```
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 tex
t)
```

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:

```
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 te
xt)
```

In addition, the following functions are available to produce analogous mappings of entire schemas or the entire current database:

```
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:

```
<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:

```
<dbname>

<schema1name>
  ...
</schema1name>

<schema2name>
  ...
</schema2name>

...

</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 version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.w3.org/1999/xhtml"
>

  <xsl:output method="xml"
      doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
      doctype-public="-//W3C/DTD XHTML 1.0 Strict//EN"
      indent="yes"/>

  <xsl:template match="/*">
    <xsl:variable name="schema" select="//xsd:schema"/>
    <xsl:variable name="tabletypename"
                  select="$schema/xsd:element[@name=name(current())]/@type"/>
    <xsl:variable name="rowtypename"
                  select="$schema/xsd:complexType[@name=$tabletypename]/xsd:sequence/x
sd:element[@name='row']/@type"/>

    <html>
      <head>
        <title><xsl:value-of select="name(current())"/></title>
      </head>
      <body>
        <table>
          <tr>
            <xsl:for-each select="$schema/xsd:complexType[@name=$rowtypename]/xsd:sequ
ence/xsd:element/@name">
              <th><xsl:value-of select="."/></th>
            </xsl:for-each>
          </tr>

          <xsl:for-each select="row">
            <tr>
              <xsl:for-each select="*">
                <td><xsl:value-of select="."/></td>
              </xsl:for-each>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

XML Function Reference

The functions described in this section operate on values of type `xml`. The section XML Predicatesalso 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:

```
SELECT xmlconcat('<abc/>', '<bar>foo</bar>');

      xmlconcat
----------------------
 <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:

```
SELECT xmlconcat('<?xml version="1.1"?><foo/>', '<?xml version="1.1" standalone="no"?>
<bar/>');

             xmlconcat
-----------------------------------
 <?xml version="1.1"?><foo/><bar/>
```

Function:

`xmlelement`

Synopsis:

```
xmlelement(name name [, xmlattributes(value [AS attname] [, ... ])] [, content, ...])
```

The `xmlelement` expression produces an XML element with the given name, attributes, and content.

Examples:

```
SELECT xmlelement(name foo);

 xmlelement
------------
 <foo/>

SELECT xmlelement(name foo, xmlattributes('xyz' as bar));

    xmlelement
------------------
 <foo bar="xyz"/>

SELECT xmlelement(name foo, xmlattributes(current_date as bar), 'cont', 'ent');

             xmlelement
-------------------------------------
 <foo bar="2017-01-26">content</foo>
```

Element and attribute names that are not valid XML names are escaped by replacing the offending characters by the sequence `_x`*HHHH*`_`, 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);

          xmlforest
------------------------------
 <foo>abc</foo><bar>123</bar>


SELECT xmlforest(table_name, column_name)
FROM information_schema.columns
WHERE table_schema = 'pg_catalog';

                                       xmlforest
----------------------------------------------------------------------------------------
-----
 <table_name>pg_authid</table_name><column_name>rolname</column_name>
 <table_name>pg_authid</table_name><column_name>rolsuper</column_name>
```

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 `xmlelement` 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 `xmlelement`.

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.

```
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:

```
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>');
 xml_is_well_formed_document
-----------------------------
 t
(1 row)

SELECT xml_is_well_formed_document('<pg:foo xmlns:pg="http://postgresql.org/stuff">bar
</my:foo>');
```

```
 xml_is_well_formed_document
-----------------------------
 f
(1 row)
```

The last example shows that the checks include whether namespaces are correctly matched.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Querying Data

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Querying Data

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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:

```
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:

```
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.

```
SET gp_log_dynamic_partition_pruning = on;
```

For information about GPORCA, see Querying Data.

Parent topic: 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 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.
  Note: The cost values generated by the Pivotal Query Optimizer and the Postgres Planner are not directly comparable. The two optimizers use different cost models, as well as different algorithms, to determine the cost of an execution plan. Nothing can or should be inferred by comparing cost values between the two optimizers.

  In addition, the cost generated for any given optimizer is valid only for comparing plan alternatives for a given single query and set of statistics. Different queries can generate plans with different costs, even when keeping the optimizer a constant.

  To summarize, the cost is essentially an internal number used by a given optimizer, and nothing should be inferred by examining only the cost value displayed in the `EXPLAIN` plans.
- **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 en
d, 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.

```
                       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 width=1)
 Settings:  optimizer=on
 Optimizer status: PQO version 1.584
(5 rows)
```

```
explain select count(*) from part;

                       QUERY PLAN
-------------------------------------------------------------------------------
--
 Aggregate  (cost=3519.05..3519.06 rows=1 width=8)
   ->  Gather Motion 2:1  (slice1; segments: 2)  (cost=3518.99..3519.03 rows=1 width=8
)
         ->  Aggregate  (cost=3518.99..3519.00 rows=1 width=8)
               ->  Seq Scan on part  (cost=0.00..3018.79 rows=100040 width=1)
 Settings:  optimizer=on
 Optimizer status: legacy query optimizer
(5 rows)
```

For this query, the server configuration parameter `OPTIMIZER` is `off`.

```
explain select count(*) from part;

                       QUERY PLAN
-------------------------------------------------------------------------------
--
 Aggregate  (cost=3519.05..3519.06 rows=1 width=8)
   ->  Gather Motion 2:1  (slice1; segments: 2)  (cost=3518.99..3519.03 rows=1 width=8
)
         ->  Aggregate  (cost=3518.99..3519.00 rows=1 width=8)
               ->  Seq Scan on part  (cost=0.00..3018.79 rows=100040 width=1)
 Settings: optimizer=off
 Optimizer status: legacy query optimizer
(5 rows)
```

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 ANALYZE` 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 `join_collapse_limit=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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.
- **Accessing External Data with PXF**
  Data managed by your organization may already reside in external sources. The Greenplum

Platform 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.

- **Accessing HDFS Data with gphdfs (Deprecated)**
  Greenplum Database leverages the parallel architecture of a Hadoop Distributed File System to read and write data files efficiently using the `gphdfs` protocol.
- **Using the Greenplum Parallel File Server (gpfdist)**
  The gpfdist protocol is used in a `CREATE EXTERNAL TABLE` SQL command to access external data served by the Greenplum Database `gpfdist` file server utility. When external data is served by gpfdist, all segments in the Greenplum Database system can read or write external table data in parallel.

Parent topic: Greenplum Database Administrator Guide

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 about 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 hosts 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.
- `gphdfs://` accesses files on a Hadoop Distributed File System (HDFS). See gphdfs:// Protocol (Deprecated).
  Note: The `gphdfs` external table protocol is deprecated and will be removed in the next major release of Greenplum Database.
- `s3://` accesses files in an Amazon S3 bucket. See s3:// Protocol.
- The `pxf://` protocol accesses external HDFS files and HBase and Hive tables using the Greenplum Platform Extension Framework (PXF). See pxf:// Protocol.

Note:

The `gphdfs://` (deprecated), `s3://`, and `pxf://` protocols are custom data access protocols, where the `file://`, `gpfdist://`, and `gpfdists://` protocols are implemented internally in Greenplum Database. The custom and internal protocols differ in these ways:

- Custom protocols must be registered using the `CREATE PROTOCOL` command. The `gphdfs://` protocol (deprecated) is preregistered when you install Greenplum Database. Registering the PXF extension in a database creates the `pxf://` protocol. (See Accessing External Data with PXF.) You can optionally register the `s3://` protocol. (See Configuring and Using S3 External Tables.) Internal protocols are always present and cannot be unregistered.
- When a custom protocol is registered, a row is added to the `pg_extprotocol` catalog table to specify the handler functions that implement the protocol. The protocol's shared libraries must have been installed on all Greenplum Database hosts. The internal protocols are not represented in the `pg_extprotocol` table and have no additional libraries to install.
- To grant users permissions on custom protocols, you use `GRANT [SELECT | INSERT |`

ALL] ON PROTOCOL. To allow (or deny) users permissions on the internal protocols, you use CREATE ROLE or ALTER ROLE to add the CREATEEXTTABLE (or NOCREATEEXTTABLE) attribute to each user's role.

External tables access external files from within the database as if they are regular database tables. External tables defined with the gpfdist/gpfdists, gphdfs (deprecated), and s3 protocols utilize Greenplum parallelism by using the resources of all Greenplum Database segments to load or unload data. The gphdfs protocol (deprecated) 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 gpdists protocols.
   - Verify that you have already set up the required one-time configuration for the gphdfs protocol (deprecated).
   - 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
  - Reading external table data in parallel from multiple Greenplum database segment instances, to optimize large load operations
  - Filter pushdown. If a query contains a WHERE clause, it may be passed to the external data source. Refer to the gp_external_enable_filter_pushdown server configuration parameter discussion for more information. Note that this feature is currently supported only with the pxf protocol (see pxf:// Protocol).

  Readable external tables allow only SELECT operations.
- 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.
- **gpfdist:// Protocol**
  The `gpfdist://` protocol is used in a URI to reference a running `gpfdist` instance.
- **gpfdists:// Protocol**
  The `gpfdists://` protocol is a secure version of the `gpfdist:// protocol`.
- **gphdfs:// Protocol (Deprecated)**
  The `gphdfs://` protocol specifies an external file path on a Hadoop Distributed File System (HDFS).
- **pxf:// Protocol**
  You can use the Greenplum Platform Extension Framework (PXF) `pxf://` protocol to access data residing on external Hadoop systems (HDFS, Hive, HBase), object store systems (Azure, Google Cloud Storage, Minio, S3), and SQL databases.
- **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.
- **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://` (deprecated) protocols.
- **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.
- **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.
- **Examples for Creating External Tables**
  These examples show how to define external data with different protocols. Each `CREATE EXTERNAL TABLE` command can contain only one protocol.

Parent topic: Working with External Data

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 Database 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:

```
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:

```
SELECT * FROM pg_max_external_files;
```

Parent topic: Defining External Tables

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. For readable external tables, `gpfdist` uncompresses `gzip` (`.gz`) and `bzip2` (`.bz2`) files automatically. For writable external tables, data is compressed using `gzip` if the target file has a `.gz` extension. 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.

Note: Compression is not supported for readable and writeable external tables when the `gpfdist` utility runs on Windows platforms.

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.

`gpfdist` supports data transformations. You can write a transformation process to convert external data from or to a format that is not directly supported with Greenplum Database external tables.

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.

Parent topic: Defining 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 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.

The server configuration parameter `verify_gpfdists_cert` controls whether SSL certificate authentication is enabled when Greenplum Database communicates with the `gpfdist` utility to either read data from or write data to an external data source. You can set the parameter value to `false` to disable authentication when testing the communication between the Greenplum Database external table and the `gpfdist` utility that is serving the external data. If the value is `false`, these SSL exceptions are ignored:

- The self-signed SSL certificate that is used by `gpfdist` is not trusted by Greenplum Database.
- The host name contained in the SSL certificate does not match the host name that is running `gpfdist`.

Warning: Disabling SSL certificate authentication exposes a security risk by not validating the `gpfdists` SSL certificate.

Parent topic: Defining External Tables

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

gphdfs:// Protocol (Deprecated)

The `gphdfs://` protocol specifies an external file path on a Hadoop Distributed File System (HDFS).

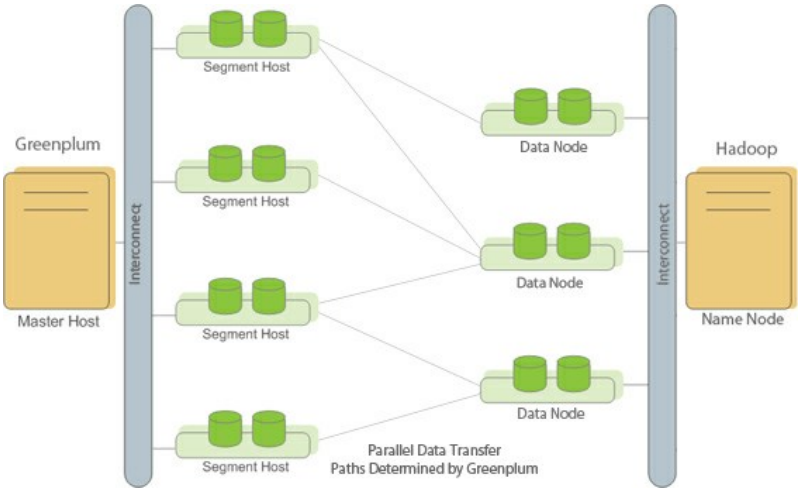
Note: The `gphdfs` external table protocol is deprecated and will be removed in the next major release of Greenplum Database. Consider using the Greenplum Platform Extension Framework (PXF) `pxf` external table protocol to access data stored in a Hadoop file system.

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 1. External Table Located on a Hadoop Distributed File System



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.

Parent topic: Defining External Tables

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

pxf:// Protocol

You can use the Greenplum Platform Extension Framework (PXF) `pxf://` protocol to access data residing on external Hadoop systems (HDFS, Hive, HBase), object store systems (Azure, Google Cloud Storage, Minio, S3), and SQL databases.

The PXF `pxf` protocol is packaged as a Greenplum Database extension. The `pxf` protocol supports reading from external data stores. You can also write text, binary, and parquet-format data with the

`pxf` protocol.

When you use the `pxf` protocol to query an external data store, you specify the directory, file, or table that you want to access. PXF requests the data from the data store and delivers the relevant portions in parallel to each Greenplum Database segment instance serving the query.

You must explicitly initialize and start PXF before you can use the `pxf` protocol to read or write external data. You must also enable PXF in each database in which you want to allow users to create external tables to access external data, and grant permissions on the `pxf` protocol to those Greenplum Database users.

For detailed information about configuring and using PXF and the `pxf` protocol, refer to Accessing External Data with PXF.

Parent topic: Defining External Tables

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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:

```
CREATE OR REPLACE FUNCTION write_to_s3() RETURNS integer AS
   '$libdir/gps3ext.so', 's3_export' LANGUAGE C STABLE;
```

```
CREATE OR REPLACE FUNCTION read_from_s3() RETURNS integer AS
   '$libdir/gps3ext.so', 's3_import' LANGUAGE C STABLE;
```

b. 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:

```
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_prefix*`N/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:

```
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_fil
e_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/abcdefff
```

- 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. For information about S3 buckets and folders, see the Amazon S3 documentation 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.

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.

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.

chunksize

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.

gpcheckcloud_newline

When downloading files from an S3 location, the `gpcheckcloud` utility appends a new line character to last line of a file if the last line of a file does not have an EOL (end of line) character. The default character is `\n` (newline). The value can be `\n`, `\r` (carriage return), or `\n\r` (newline/carriage return).

Adding an EOL character prevents the last line of one file from being concatenated with the first line of next file.

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.

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.

verifycert

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`.

- `verifycert=false` - Ignores authentication errors and allows encrypted communication over HTTPS.
- `verifycert=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/ co
nfig=/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 c
onfig=/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/ re
gion=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.

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.
- 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 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

-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.

-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`.

-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.

-t

Sends a template configuration file to `STDOUT`. You can capture the output and create an `s3` configuration file to connect to Amazon S3.

-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.

```
gpcheckcloud -t > ./mytest_s3.config
```

This example attempts to upload a local file, test-data.csv to an S3 bucket location using the `s3` configuration file `s3.mytestconf`:

```
gpcheckcloud -u ./test-data.csv "s3://s3-us-west-2.amazonaws.com/test1/abc config=s3.m
ytestconf"
```

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`.

```
gpcheckcloud -c "s3://s3-us-west-2.amazonaws.com/test1/abc config=s3.mytestconf"
```

Download all files from the S3 bucket location and send the output to `STDOUT`.

```
gpcheckcloud -d "s3://s3-us-west-2.amazonaws.com/test1/abc config=s3.mytestconf"
```

Parent topic: Defining External Tables

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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://` (deprecated) 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.

Parent topic: Defining External Tables

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Defining External Tables

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.

Parent topic: Defining External Tables

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 '|');
```

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.

```
=# 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**
- **Example 2—Multiple gpfdist instances**
- **Example 3—Multiple gpfdists instances**
- **Example 4—Single gpfdist instance with error logging**
- **Example 5—TEXT Format on a Hadoop Distributed File Server**
- **Example 6—Multiple files in CSV format with header rows**
- **Example 7—Readable External Web Table with Script**
- **Example 8—Writable External Table with gpfdist**
- **Example 9—Writable External Web Table with Script**
- **Example 10—Readable and Writable External Tables with XML Transformations**

Parent topic: Defining External Tables

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

```
=# CREATE EXTERNAL TABLE ext_expenses ( name text,
    date date, amount float4, category text, desc1 text )
    LOCATION ('gpfdist://etlhost-1:8081/*')
FORMAT 'TEXT' (DELIMITER '|');
```

Parent topic: Examples for Creating External Tables

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

```
=# CREATE EXTERNAL TABLE ext_expenses ( name text,
   date date,  amount float4, category text, desc1 text )
   LOCATION ('gpfdist://etlhost-1:8081/*.txt',
             'gpfdist://etlhost-2:8081/*.txt')
   FORMAT 'TEXT' ( DELIMITER '|' NULL ' ') ;
```

Parent topic: Examples for Creating External Tables

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

```
=# CREATE EXTERNAL TABLE ext_expenses ( name text,
   date date,  amount float4, category text, desc1 text )
   LOCATION ('gpfdists://etlhost-1:8081/*.txt',
             'gpfdists://etlhost-2:8082/*.txt')
   FORMAT 'TEXT' ( DELIMITER '|' NULL ' ') ;
```

Parent topic: Examples for Creating External Tables

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

```
=# CREATE EXTERNAL TABLE ext_expenses ( name text,
   date date, amount float4, category text, desc1 text )
   LOCATION ('gpfdist://etlhost-1:8081/*.txt',
             'gpfdist://etlhost-2:8082/*.txt')
   FORMAT 'TEXT' ( DELIMITER '|' NULL ' ')
   LOG ERRORS SEGMENT REJECT LIMIT 5;
```

To create the readable ext_expenses table from CSV-formatted text files:

```
=# CREATE EXTERNAL TABLE ext_expenses ( name text,
   date date,  amount float4, category text, desc1 text )
   LOCATION ('gpfdist://etlhost-1:8081/*.txt',
             'gpfdist://etlhost-2:8082/*.txt')
   FORMAT 'CSV' ( DELIMITER ',' )
   LOG ERRORS SEGMENT REJECT LIMIT 5;
```

Parent topic: Examples for Creating External Tables

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Example 5—TEXT Format on a Hadoop Distributed File Server

Note: The `gphdfs` external table protocol is deprecated and will be removed in the next major release of Greenplum Database.

Creates a readable external table, *ext_expenses*, using the `gphdfs` protocol. The column delimiter is a pipe (|).

```
=# 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 with gphdfs (Deprecated).

Parent topic: Examples for Creating External Tables

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

```
=# CREATE EXTERNAL TABLE ext_expenses ( name text,
   date date,  amount float4, category text, desc1 text )
   LOCATION ('file://filehost/data/international/*',
             'file://filehost/data/regional/*',
             'file://filehost/data/supplement/*.csv')
   FORMAT 'CSV' (HEADER);
```

Parent topic: Examples for Creating External Tables

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Example 7—Readable External Web Table with Script

Creates a readable external web table that executes a script once per segment host:

```
=# CREATE EXTERNAL WEB TABLE log_output (linenum int,
    message text)
   EXECUTE '/var/load_scripts/get_log_data.sh' ON HOST
   FORMAT 'TEXT' (DELIMITER '|');
```

Parent topic: Examples for Creating External Tables

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

```
=# CREATE WRITABLE EXTERNAL TABLE sales_out (LIKE sales)
   LOCATION ('gpfdist://etl1:8081/sales.out')
   FORMAT 'TEXT' ( DELIMITER '|' NULL ' ')
   DISTRIBUTED BY (txn_id);
```

Parent topic: Examples for Creating External Tables

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Example 9—Writable External Web Table with Script

Creates a writable external web table, `campaign_out`, that pipes output data recieved by the segments to an executable script, `to_adreport_etl.sh`:

```
=# CREATE WRITABLE EXTERNAL WEB TABLE campaign_out
    (LIKE campaign)
    EXECUTE '/var/unload_scripts/to_adreport_etl.sh'
    FORMAT 'TEXT' (DELIMITER '|');
```

Parent topic: Examples for Creating External Tables

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 External Data with gpfdist and gpload.

Parent topic: Examples for Creating External Tables

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Accessing External Data with PXF

Data managed by your organization may already reside in external sources. The Greenplum Platform 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 Hadoop and Object Store connectors. These connectors enable you to read external data stored in text, Avro, JSON, RCFile, Parquet, SequenceFile, and ORC formats. You can use the JDBC connector to access an external SQL database.

Note: PXF supports filter pushdown in the Hive, HBase, and JDBC connectors.

The Greenplum Platform 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 PXF, refer to the Greenplum Platform Extension Framework (PXF) documentation.

Parent topic: Working with External Data
Parent topic: Loading and Unloading Data

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Accessing HDFS Data with gphdfs (Deprecated)

Greenplum Database leverages the parallel architecture of a Hadoop Distributed File System to read and write data files efficiently using the `gphdfs` protocol.

Note: The `gphdfs` external table protocol is deprecated and will be removed in the next major release of Greenplum Database. Consider using the Greenplum Platform Extension Framework (PXF) `pxf` external table protocol to access data stored in a Hadoop file system.

There are three steps to using the gphdfs protocol with HDFS:

- **One-time gphdfs Protocol Installation (Deprecated)**
- **Grant Privileges for the gphdfs Protocol (Deprecated)**
- **Specify gphdfs Protocol in an External Table Definition (Deprecated)**
- **gphdfs Support for Avro Files (Deprecated)**
- **gphdfs Support for Parquet Files (Deprecated)**
- **HDFS Readable and Writable External Table Examples (Deprecated)**
- **Reading and Writing Custom-Formatted HDFS Data with gphdfs (Deprecated)**
- **Using Amazon EMR with Greenplum Database installed on AWS (Deprecated)**

Parent topic: Working with External Data

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

One-time gphdfs Protocol Installation (Deprecated)

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.

   ```
   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 1. Server Configuration Parameters for Hadoop Targets

Configuration Parameter	Description	Default Value	Set Classifications
gp_hadoop_target_version	The Hadoop target. Choose one of the following: cdh hadoop hdp mpr	hadoop	master session reload
gp_hadoop_home	This parameter specifies the installation directory for Hadoop.	NULL	master session reload

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 'hdp'
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.

Parent topic: Accessing HDFS Data with gphdfs (Deprecated)

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 `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.

```
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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Grant Privileges for the gphdfs Protocol (Deprecated)

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`).

      ```
      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).

Parent topic: Accessing HDFS Data with gphdfs (Deprecated)

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Specify gphdfs Protocol in an External Table Definition (Deprecated)

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:

```
LOCATION ('gphdfs://mycluster/path/filename.txt')
```

A cluster without HA specifies the hostname and port of the name node in the `LOCATION` clause:

```
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:

  ```
  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:

  ```
  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 a 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.

Parent topic: Accessing HDFS Data with gphdfs (Deprecated)

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 1. Compression Options

Compression Option	Values	Default Value
compress	true or false	false
compression_type	BLOCK or RECORD	RECORD For AVRO format, compression_type must be block if compress is true.
codec	Codec class name	GzipCodec for text format and DefaultCodec for gphdfs_export format. For AVRO format, the value is either deflate (the default) or snappy
codec_level (for AVRO format and deflate codec only)	integer between 1 and 9	6 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.

Place compression options in the query portion of the URI.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

gphdfs Support for Avro Files (Deprecated)

Note: The gphdfs external table protocol is deprecated and will be removed in the next major release of Greenplum Database. You can use the Greenplum Platform Extension Framework (PXF) to access Avro-format data.

You can use the Greenplum Database gphdfs protocol to access Avro files on a Hadoop file system (HDFS).

Parent topic: Accessing HDFS Data with gphdfs (Deprecated)

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

```
{"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:

```
{ "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/

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.

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`.

```
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.

```
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 1. Avro External Table Parameters

Parameter	Value	Readable/Writable	Default Value
schema	*URL_to_schema_file*	Read and Write	None. For a readable external table • The specified schema overrides the schema in the Avro file. See Avro Schema Overrides • If not specified, Greenplum Database uses the Avro file schema. For a writable external table • Uses the specified schema when creating the Avro file. • If not specified, Greenplum Database creates a schema according to the external table definition.
namespace	*avro_namespace*	Write only	`public.avro` If specified, a valid Avro *namespace*.
compress	`true` or `false`	Write only	`false`
compression_type	`block`	Write only	Optional. For `avro` format, `compression_type` must be `block` if `compress` is `true`.

Table 1. Avro External Table Parameters

Parameter	Value	Readable/Writable	Default Value
codec	`deflate` or `snappy`	Write only	`deflate`
codec_level (`deflate` codec only)	integer between 1 and 9	Write only	`6` 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 2. Avro Primitive Data Type Support for Readable External Tables

Avro Data Type	Greenplum Database Data Type
null	Supported only in a Avro union data type. See Data Conversion when Writing Avro Files.
boolean	boolean
int	int or smallint
long	bigint
float	real
double	double
bytes	bytea
string	text

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`: `short result = (short)IntValue;`.

This table lists the Avro complex data types and the and the Greenplum Database type it is converted to.

Table 3. Avro Complex Data Type Support for Readable External Tables

Avro Data Type	Greenplum Database Data Type
enum	int The integer represents the zero-based position of the symbol in the schema.
array	array 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.
maps	Not supported
union	The first non-null data type.
fixed	bytea
record	XML data

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 `sting`.
- `age` data is converted to Greenplum Database `int`.
- `clist` records are converted to `XML`.

```
{"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"]}
             ]}
          },
         {"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 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& quot;>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 paramter `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.

```
{
 "type":"record",
 "name":"tav2",
 "namespace":"public.avro",
  "doc":"",
  "fields":[
   {"name":"id","type":["null","int"],"doc":""},
   {"name":"name","type":["null","string"],"doc":""},
   {"name":"age","type":["null","long"],"doc":""},
   {"name":"birth","type":["null","string"],"doc":""}
        ]
}
```

This updated schema contains a comment field.

```
{
 "type":"record",
 "name":"tav2",
 "namespace":"public.avro",
  "doc":"",
  "fields":[
   {"name":"id","type":["null","int"],"doc":""},
```

```
  {"name":"name","type":["null","string"],"doc":""},
  {"name":"birth","type":["null","string"],"doc":""},
  {"name":"age","type":["null","long"],"doc":""},
  {"name":"comment","type":["null","string"],"doc":""}
      ]
}
```

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":"",
 "fields":[
  {"name":"id","type":["null","int"],"doc":""},
  {"name":"name","type":["null","string"],"doc":""},
  {"name":"birth","type":["null","string"],"doc":""},
  {"name":"comment","type":["null","string"],"doc":""}
      ]
}
```

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 4. Avro Data Types Generated by Greenplum Database

Greenplum Database Data Type	Avro Union Data Type Definition
boolean	`["boolean", "null"]`
int	`["int", "null"]`
bigint	`["long", "null"]`
smallint	`["int", "null"]`
real	`["float", "null"]`
double	`["double", "null"]`
bytea	`["bytes", "null"]`
text	`["string", "null"]`

Table 4. Avro Data Types Generated by Greenplum Database

Greenplum Database Data Type	Avro Union Data Type Definition
array	`[{array}, "null"]` 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`.

```
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.

```
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.

```
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';
```

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

gphdfs Support for Parquet Files (Deprecated)

Note: The `gphdfs` external table protocol is deprecated and will be removed in the next major release of Greenplum Database. You can use the Greenplum Platform Extension Framework (PXF) to access Parquet-format data.

You can use the Greenplum Database `gphdfs` protocol to access Parquet files on a Hadoop file system (HDFS).

Parent topic: Accessing HDFS Data with gphdfs (Deprecated)

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.

For information about the Parquet file format, see the Parquet documentation http://parquet.apache.org/documentation/latest/.

For an overview of columnar data storage and the Parquet file format, see https://blog.twitter.com/2013/dremel-made-simple-with-parquet.

Required Parquet Jar Files

The `gphdfs` protocol supports Parquet versions 1.7.0 and later. For each version, the required Parquet jar files are included in a bundled jar file `parquet-hadoop-bundle-<version>.jar`.

Earlier Parquet versions not use the Java class names `org.apache.parquet` and are not supported. The `gphdfs` protocol expects the Parquet Java class names to be `org.apache.parquet.xxx`.

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 jar files with the class name `org.apache.parquet` can be downloaded and installed on the Greenplum Database hosts.

For information about downloading the Parquet jar files, see https://mvnrepository.com/artifact/org.apache.parquet/parquet-hadoop-bundle

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`.

```
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:

```
message test {
    repeated byte_array binary_field;
    required int32 int32_field;
    optional int64 int64_field;
    required boolean boolean_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.

```
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 identifies how Greenplum database converts the Parquet data type if the Parquet schema definition does not contain an annotation.

Table 1. Data Type Conversion when Reading a Parquet File

Parquet Data Type	Greenplum Database Data Type
boolean	boolean
int32	int or smallint
int64	long
int96	bytea
float	real
double	double
byte_array	bytea
fixed_len_byte_array	bytea

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 2. Data Type (with Annotation) Conversion when Reading Parquet File

Parquet Schema Data Type and Annotation	Greenplum Database Data Type
binary with `json` or `utf8` annotation	text
binary and the Greenplum Database column data type is text	text
int32 with `int_16` annotation	smallint
int32, int64, fixed_len_byte_array, or binary with `decimal` annotation	decimal
`repeated`	array column - The data type is converted according to Table 1
`optional`, `required`	Data type is converted according to Table 1

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`.

```
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.

```
<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`.

```
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.

```
<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.

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 3. Data Type Conversion when Reading a Hive Generated Parquet File

Hive Data Type	Greenplum Database Data Type
tinyint	int
smallint	int
int	int
bigint	bigint
decimal	numeric
float	real
double	float
boolean	boolean
string	text
char	text or char
varchar	text or varchar
timestamp	bytea
binary	bytea
array	xml
map	xml
struct	xml

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.

  ```
  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.

```
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 the Hive generated Parquet schema for a Hive column `col1` of data type `array[int]`.

  ```
  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 4. Parquet Format External Table location Parameters

Option	Values	Readable/Writable	Default Value
schema	*URL_to_schema*	Write only	None. If not specified, the `gphdfs` protocol creates a schema according to the external table definition.
pagesize	> 1024 Bytes	Write only	1 MB
rowgroupsize	> 1024 Bytes	Write only	8 MB
parquetversion *or* pqversion	`v1`, `v2`	Write only	`v1`
codec	`UNCOMPRESSED`, `GZIP`, `LZO`, `snappy`	Write only	`UNCOMPRESSED`
dictionaryenable[1]	`true`, `false`	Write only	false

Table 4. Parquet Format External Table location Parameters

Option	Values	Readable/Writable	Default Value
dictionarypagesize[1]	> 1024 Bytes	Write only	512 KB

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 uses 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 5. Schema Data Type Conversion when Writing a Parquet File

Greenplum Database Data Type	Parquet Schema Data Type
boolean	optional boolean
smallint	optional int32 with annotation `int_16`
int	optional int32
bigint	optional int64
real	optional float
double	optional double
numeric or decimal	binary with annotation `decimal`
bytea	optional binary
array column	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`
Others	binary with annotation `utf8`

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.

```
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.

```
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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

HDFS Readable and Writable External Table Examples (Deprecated)

Note: The `gphdfs` external table protocol is deprecated and will be removed in the next major release of Greenplum Database.

The following code defines a readable external table for an HDFS file named `filename.txt` on port 8081.

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

The following code defines a set of readable external tables that have a custom format located in the same HDFS directory on port 8081.

```
=# CREATE EXTERNAL TABLE ext_expenses (
        name text,
        date date,
        amount float4,
        category text,
        desc1 text )
   LOCATION ('gphdfs://hdfshost-1:8081/data/custdat*.dat')
   FORMAT 'custom' (formatter='gphdfs_import');
```

The following code defines an HDFS directory for a writable external table on port 8081 with all compression options specified.

```
=# CREATE WRITABLE EXTERNAL TABLE ext_expenses (
        name text,
        date date,
        amount float4,
        category text,
        desc1 text )
   LOCATION ('gphdfs://hdfshost-1:8081/data/?compress=true&compression_type=RECORD
   &codec=org.apache.hadoop.io.compress.DefaultCodec')
   FORMAT 'custom' (formatter='gphdfs_export');
```

Because the previous code uses the default compression options for `compression_type` and `codec`, the following command is equivalent.

```
=# CREATE WRITABLE EXTERNAL TABLE ext_expenses (
        name text,
        date date,
        amount float4,
        category text,
        desc1 text )
   LOCATION    ('gphdfs://hdfshost-1:8081/data?compress=true')
   FORMAT 'custom' (formatter='gphdfs_export');
```

Parent topic: Accessing HDFS Data with gphdfs (Deprecated)

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Reading and Writing Custom-Formatted HDFS Data with gphdfs (Deprecated)

Note: The `gphdfs` external table protocol is deprecated and will be removed in the next major release of Greenplum Database.

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-1.2-javadoc.tar` and open `index.html`. The Javadoc documents the following packages:

```
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 `hdp-gnet-1.2.0.0.jar` if you use the HDP

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.

Parent topic: Accessing HDFS Data with gphdfs (Deprecated)

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

```
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"));
  job.waitForCompletion(true);
}
```

Run CREATE EXTERNAL TABLE

The Hadoop location corresponds to the output path in the MapReduce job.

```
=# CREATE EXTERNAL TABLE demodata
   LOCATION ('gphdfs://hdfshost-1:8081/demo/data/MRTest1')
   FORMAT 'custom' (formatter='gphdfs_import');
```

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.

   ```
   =# 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.

Sample MapReduce Code

```
/*
 * DEMO Map/Reduce class test2
 * -- Convert GPDBFormat back to TEXT
 */
public static class Map_test2 extends Mapper<LongWritable, GPDBWritable,
  Text, NullWritable> {
  public void map(LongWritable key, GPDBWritable value, Context context )
    throws IOException {
```

```
    try {
      context.write(new Text(value.toString()), NullWritable.get());
    } catch (Exception e) { throw new IOException (e.getMessage()); }
  }
}

public static void runTest2() throws Exception{
Configuration conf = new Configuration(true);
 Job job = new Job(conf, "test2");
 job.setJarByClass(demoMR.class);
 job.setInputFormatClass(GPDBInputFormat.class);
 job.setOutputKeyLClass (Text.class);
 job.setOutputValueClass(NullWritable.class);
 job.setOutputFormatClass(TextOutputFormat.class);
 job.setMapperClass(Map_test2.class);
     GPDBInputFormat.setInputPaths (job,
     new Path("/demo/data/writeFromGPDB_42"));
 GPDBOutputFormat.setOutputPath(job, new Path("/demo/data/MRTest2"));
 job.waitForCompletion(true);

}
```

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Using Amazon EMR with Greenplum Database installed on AWS (Deprecated)

Note: The `gphdfs` external table protocol is deprecated and will be removed in the next major release of Greenplum Database.

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 (Deprecated), 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 1.

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 Hadooop version information that can be used to configure Greenplum Database.

Table 1. EMR Hadoop Configuration Information

EMR Version	EMR Apache Hadoop Version	EMR Hadoop Master Port	gp_hadoop_target_version	Hadoop Version on Greenplum Database Segment Hosts
4.0	2.6	8020	`hadoop2`	Apache Hadoop 2.x
3.9	2.4	9000	`hadoop2`	Apache Hadoop 2.x
3.8	2.4	9000	`hadoop2`	Apache Hadoop 2.x
3.3	2.4	9000	`hadoop2`	Apache Hadoop 2.x

Parent topic: Accessing HDFS Data with gphdfs (Deprecated)

Using the Greenplum Parallel File Server (gpfdist)

The gpfdist protocol is used in a `CREATE EXTERNAL TABLE` SQL command to access external data served by the Greenplum Database `gpfdist` file server utility. When external data is served by gpfdist, all segments in the Greenplum Database system can read or write external table data in parallel.

This topic describes the setup and management tasks for using gpfdist with external tables.

Parent topic: Working with External Data

About gpfdist and External Tables

The `gpfdist` file server utility is located in the `$GPHOME/bin` directory on your Greenplum Database master host and on each segment host. When you start a `gpfdist` instance you specify a listen port and the path to a directory containing files to read or where files are to be written. For example, this command runs `gpfdist` in the background, listening on port 8801, and serving files in the `/home/gpadmin/external_files` directory:

```
$ gpfdist -p 8801 -d /home/gpadmin/external_files &
```

The `CREATE EXTERNAL TABLE` command `LOCATION` clause connects an external table definition to one or more `gpfdist` instances. If the external table is readable, the `gpfdist` server reads data records from files from in specified directory, packs them into a block, and sends the block in a response to a Greenplum Database segment's request. The segments unpack rows they receive and distribute them according to the external table's distribution policy. If the external table is a writable table, segments send blocks of rows in a request to gpfdist and gpfdist writes them to the external file.

External data files can contain rows in CSV format or any delimited text format supported by the `FORMAT` clause of the `CREATE EXTERNAL TABLE` command. In addition, `gpfdist` can be configured with a YAML-formatted file to transform external data files between a supported text format and another format, for example XML or JSON. See <ref> for an example that shows how to use `gpfdist` to read external XML files into a Greenplum Database readable external table.

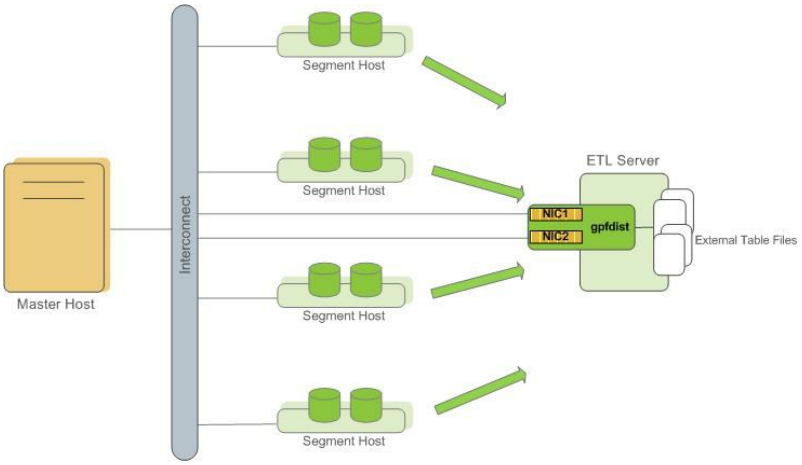
For readable external tables, `gpfdist` uncompresses `gzip` (`.gz`) and `bzip2` (`.bz2`) files automatically. You can use the wildcard character (*) or other C-style pattern matching to denote multiple files to read. External files are assumed to be relative to the directory specified when you started the `gpfdist` instance.

About gpfdist Setup and Performance

You can run `gpfdist` instances on multiple hosts and you can run multiple `gpfdist` instances on each host. This allows you to deploy `gpfdist` servers strategically so that you can attain fast data load and unload rates by utilizing all of the available network bandwidth and Greenplum Database's parallelism.

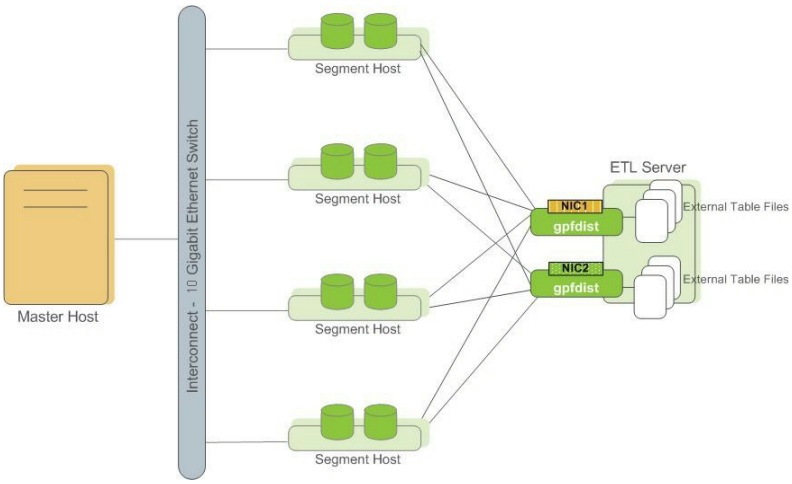
- Allow network traffic to use all ETL host network interfaces simultaneously. Run one instance of gpfdist for each interface on the ETL host, then declare the host name of each NIC in the `LOCATION` clause of your external table definition (see Examples for Creating External Tables).

Figure 1. 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 gpfdist servers.

Figure 2. 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:

```
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:

```
$ gpfdist -d /var/load_files -p 8081 -l /home/gpadmin/log &
```

For multiple gpfdist instances on the same ETL host (see Figure 1), use a different base directory and port for each instance. For example:

```
$ 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:

```
$ ps -ef | grep gpfdist
```

Then kill the process, for example (where 3456 is the process ID in this example):

```
$ 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):

```
$ 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.

If the `gpfdist` utility hangs with no read or write activity occurring, you can generate a core dump the next time a hang occurs to help debug the issue. Set the environment variable `GPFDIST_WATCHDOG_TIMER` to the number of seconds of no activity to wait before `gpfdist` is forced to exit. When the environment variable is set and `gpfidst` hangs, the utility aborts after the specified number of seconds, creates a core dump, and sends abort information to the log file.

This example sets the environment variable on a Linux system so that `gpfidst` exits after 300 seconds (5 minutes) of no activity.

```
export GPFDIST_WATCHDOG_TIMER=300
```

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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, Amazon S3, and web servers.

- The `COPY` SQL command transfers data between an external text file on the master host, or

multiple text files on segment hosts, and a Greenplum Database table.

- Readable 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 `gpload` utility automates the steps of a load task using `gpfdist` and a YAML-formatted control file.
- You can create readable and writable external tables with the Greenplum Platform Extension Framework (PXF), and use these tables to load data into, or offload data from, Greenplum Database. For information about using PXF, refer to Accessing External Data with PXF.
- The Greenplum-Kafka Integration provides high-speed, parallel data transfer from Kafka to Greenplum Database. For information about using these tools, refer to the Greenplum-Kafka Integration documentation.
- The Greenplum Streaming Server is an ETL tool and API that you can use to load data into Greenplum Database. For information about using this tool, refer to the Greenplum Streaming Server documentation.
- 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 Greenplum-Informatica Connector provides high speed data transfer from an Informatica PowerCenter cluster to a Pivotal Greenplum Database cluster for batch and streaming ETL operations. For information about using the Greenplum-Informatica Connector, refer to the documentation at https://greenplum-informatica.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**
- **Loading and Writing Non-HDFS Custom Data**

- **Handling Load Errors**
- **Loading Data with gpload**
- **Accessing External Data with PXF**
  Data managed by your organization may already reside in external sources. The Greenplum Platform 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.
- **Loading Kafka Data with the Greenplum-Kafka Integration**
- **Loading Data with the Greenplum Streaming Server**
- **Using the Greenplum-Spark Connector**
- **Using the Greenplum-Informatica Connector**
- **Transforming External Data with gpfdist and gpload**
  The `gpfdist` parallel file server allows you to set up transformations that enable Greenplum Database external tables to read and write files in formats that are not supported with the `CREATE EXTERNAL TABLE` command's `FORMAT` clause. An *input* transformation reads a file in the foreign data format and outputs rows to `gpfdist` in the CSV or other text format specified in the external table's `FORMAT` clause. An *output* transformation receives rows from `gpfdist` in text format and converts them to the foreign data format.
- **Loading Data with COPY**
- **Running COPY in Single Row Error Isolation Mode**
- **Optimizing Data Load and Query Performance**
- **Unloading Data from Greenplum Database**
- **Formatting Data Files**
- **Example Custom Data Access Protocol**

Parent topic: Greenplum Database Administrator Guide

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`:

```
=# INSERT INTO expenses_travel
    SELECT * from ext_expenses where category='travel';
```

To load all data into a new database table:

```
=# CREATE TABLE expenses AS SELECT * from ext_expenses;
```

Parent topic: Loading and Unloading Data

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Loading and Writing Non-HDFS Custom Data

Greenplum Database supports `TEXT` and `CSV` formats for importing and exporting data through external tables. You can load and save data in other formats by defining a custom format or custom protocol or by setting up a transformation with the `gpfdist` parallel file server.

For information about importing custom data from HDFS, see Reading and Writing Custom-

Formatted HDFS Data with gphdfs (Deprecated).

- **Using a Custom Format**
- **Using a Custom Protocol**

Parent topic: Loading and Unloading Data

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Using a Custom Format

You specify a custom data format in the `FORMAT` clause of `CREATE EXTERNAL TABLE`.

```
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**
- **Examples: Read Fixed-Width Data**

Parent topic: Loading and Writing Non-HDFS Custom Data

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

```
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.

  ```
  line_delim=E'\n'
  line_delim=E'\r'
  line_delim=E'\r\n'
  line_delim='abc'
  ```

Parent topic: Using a Custom Format

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Examples: Read Fixed-Width Data

The following examples show how to read fixed-width data.

Example 1 – Loading a table with all fields defined

```
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

```
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

```
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 \r\n line delimiter

```
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');
```

Parent topic: Using a Custom Format

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Using a Custom Protocol

Greenplum Database 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, you could connect Greenplum Database in parallel to another system directly, and stream data from one to the other without the need to materialize the 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;
   ```

   The format of the optional validator function is:

   ```
   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>;
```

Parent topic: Loading and Writing Non-HDFS Custom Data

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 `count` 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**
- **Capture Row Formatting Errors and Declare a Reject Limit**
- **Viewing Bad Rows in the Error Log**
- **Identifying Invalid CSV Files in Error Table Data**
- **Moving Data between Tables**

Parent topic: Loading and Unloading Data

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*:

```
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 :

```
SELECT gp_truncate_error_log('ext_expenses');
```

Parent topic: Handling Load Errors

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

```
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.

Parent topic: Handling Load Errors

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 1. Error Log Format

column	type	description
cmdtime	timestamptz	Timestamp when the error occurred.
relname	text	The name of the external table or the target table of a `COPY` command.
filename	text	The name of the load file that contains the error.
linenum	int	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.

Table 1. Error Log Format

column	type	description
bytenum	int	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.
errmsg	text	The error message text.
rawdata	text	The raw data of the rejected row.
rawbytes	bytea	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.

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*.

Parent topic: Handling Load Errors

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.

Parent topic: Handling Load Errors

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)

Parent topic: Handling Load Errors

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Note: `gpfdist` and `gpload` are compatible only with the Greenplum Database major version in which they are shipped. For example, a `gpfdist` utility that is installed with Greenplum Database 4.x cannot be used with Greenplum Database 5.x or 6.x.

Note: `MERGE` and `UPDATE` operations are not supported if the target table column name is a reserved keyword, has capital letters, or includes any character that requires quotes (" ") to identify the column.

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
```

Parent topic: Loading and Unloading Data

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Accessing External Data with PXF

Data managed by your organization may already reside in external sources. The Greenplum Platform 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 Hadoop and Object Store connectors. These connectors enable you to read external data stored in text, Avro, JSON, RCFile, Parquet, SequenceFile, and ORC formats. You can use the JDBC connector to access an external SQL database.

Note: PXF supports filter pushdown in the Hive, HBase, and JDBC connectors.

The Greenplum Platform 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 PXF, refer to the Greenplum Platform Extension Framework (PXF) documentation.

Parent topic: Working with External Data
Parent topic: Loading and Unloading Data

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Transforming External Data with gpfdist and gpload

The `gpfdist` parallel file server allows you to set up transformations that enable Greenplum Database external tables to read and write files in formats that are not supported with the `CREATE EXTERNAL TABLE` command's `FORMAT` clause. An *input* transformation reads a file in the foreign data format and outputs rows to `gpfdist` in the CSV or other text format specified in the external table's `FORMAT` clause. An *output* transformation receives rows from `gpfdist` in text format and converts them to the foreign data format.

Note: `gpfdist` and `gpload` are compatible only with the Greenplum Database major version in which they are shipped. For example, a `gpfdist` utility that is installed with Greenplum Database 4.x cannot be used with Greenplum Database 5.x or 6.x.

This topic describes the tasks to set up data transformations that work with `gpfdist` to read or write external data files with formats that Greenplum Database does not support.

- About gpfdist Transformations
- Determine the Transformation Schema
- Write a Transformation
- Write the gpfdist Configuration File
- Transfer the Data
- Configuration File Format

- XML Transformation Examples

Parent topic: Loading and Unloading Data

About gpfdist Transformations

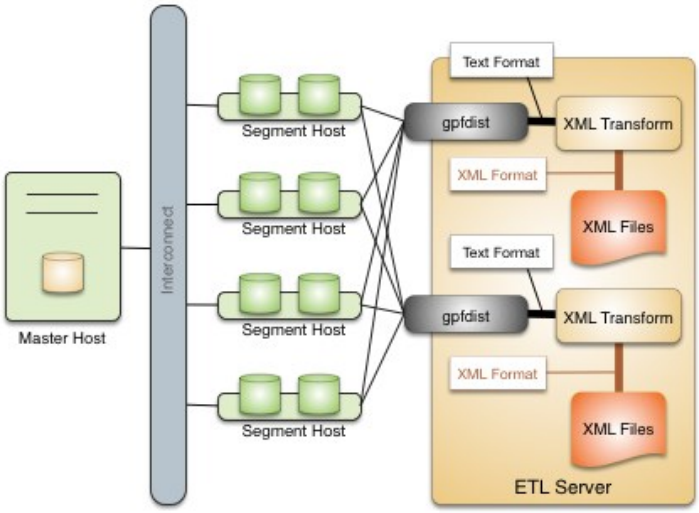
To set up a transformation for a data format, you provide an executable command that `gpfdist` can call with the name of the file containing data. For example, you could write a shell script that runs an XSLT transformation on an XML file to output rows with columns delimited with a vertical bar (`|`) character and rows delimited with linefeeds.

Transformations are configured in a YAML-formatted configuration file passed to `gpfdist` on the command line.

If you want to load the external data into a table in the Greenplum database, you can use the `gpload` utility to automate the tasks to create an external table, run `gpfdist`, and load the transformed data into the database table.

Accessing data in external XML files from within the database is a common example requiring transformation. The following diagram shows *gpfdist* performing a transformation on XML files on an ETL server.

Figure 1. External Tables using XML Transformations



Following are the high-level steps to set up a `gpfdist` transformation for external data files. The process is illustrated with an XML example.

1. Determine the transformation schema.
2. Write a transformation.
3. Write the gpfdist configuration file.
4. Transfer the data.

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 source files 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 XML file that contains price records. Each price record contains two fields: an item number and a price.

```
<?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 of this transformation is to import all the data into a Greenplum Database readable external table with an integer `itemnumber` column and a decimal `price` column.

Write a Transformation

The transformation specifies what to extract from the data. You can use any authoring environment and language appropriate for your project. For XML transformations choose from technologies 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 two-column delimited text format.

```
708421|19.99
708466|59.25
711121|24.99
```

The following STX transform, called *input_ transform.stx*, performs the data transformation.

```
<?xml version="1.0"?>
<stx:transform version="1.0"
   xmlns:stx="http://stx.sourceforge.net/2002/ns"
   pass-through="none">
  <!-- declare variables -->
  <stx:variable name="itemnumber"/>
  <stx:variable name="price"/>
  <!-- match and output prices as columns delimited by | -->
  <stx:template match="/prices/pricerecord">
    <stx:process-children/>
    <stx:value-of select="$itemnumber"/>
<stx:text>|</stx:text>
    <stx:value-of select="$price"/>      <stx:text>
</stx:text>
  </stx:template>
  <stx:template match="itemnumber">
    <stx:assign name="itemnumber" select="."/>
  </stx:template>
  <stx:template match="price">
    <stx:assign name="price" select="."/>
  </stx:template>
</stx:transform>
```

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 File

The gpfdist configuration is specified as a YAML 1.1 document. It contains rules that gpfdist uses to select a transformation to apply when loading or extracting data.

This example gpfdist configuration contains the following items that are required for the *prices.xml* transformation scenario:

- 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 transformation 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 transformation name `prices_input` in the following example will be referenced later when creating the table in SQL.

```
---
VERSION: 1.0.0.1
TRANSFORMATIONS:
  prices_input:
    TYPE:     input
    COMMAND:  /bin/bash input_transform.sh %filename%
```

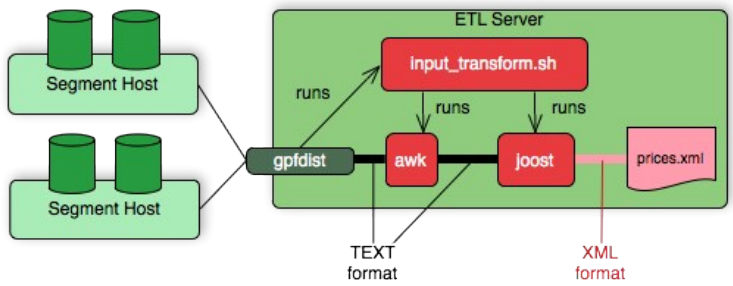
The `COMMAND` setting uses a wrapper script called `input_transform.sh` with a `%filename%` placeholder. When gpfdist runs the `prices_input` transform, it invokes `input_transform.sh` with `/bin/bash` and replaces the `%filename%` placeholder with the path to the input file to transform. The wrapper 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.

```
#!/bin/bash
# input_transform.sh - sample input transformation,
# demonstrating use of Java and Joost STX to convert XML into
# text to load into Greenplum Database.
# java arguments:
#   -jar joost.jar         joost STX engine
#   -nodecl                  don't generate a <?xml?> declaration
#   $1                        filename to process
#   input_transform.stx    the STX transformation
#
# the AWK step eliminates a blank line joost emits at the end
java \
    -jar joost.jar \
    -nodecl \
    $1 \
    input_transform.stx \
```

```
| awk 'NF>0
```

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.



Transfer the Data

Create the target database tables with SQL statements based on the appropriate schema.

There are no special requirements for Greenplum Database tables that hold loaded data. In the prices example, the following command creates the `prices` table, where the data is to be loaded.

```
CREATE TABLE prices (
  itemnumber integer,
  price       decimal
)
DISTRIBUTED BY (itemnumber);
```

Next, use one of the following approaches to transform the data with gpfdist.

- `gpload` supports only input transformations, but in many cases is easier to implement.
- `gpfdist` with `INSERT INTO SELECT FROM` supports both input and output transformations, but exposes details that `gpload` automates for you.

Transforming with gpload

The Greenplum Database `gpload` utility orchestrates a data load operation using the `gpfdist` parallel file server and a YAML-formatted configuration file. `gpload` automates these tasks:

- Creates a readable external table in the database.
- Starts `gpfdist` instances with the configuration file that contains the transformation.
- Runs `INSERT INTO` *`table_name`* `SELECT FROM` *`external_table`* to load the data.
- Removes the external table definition.

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`.

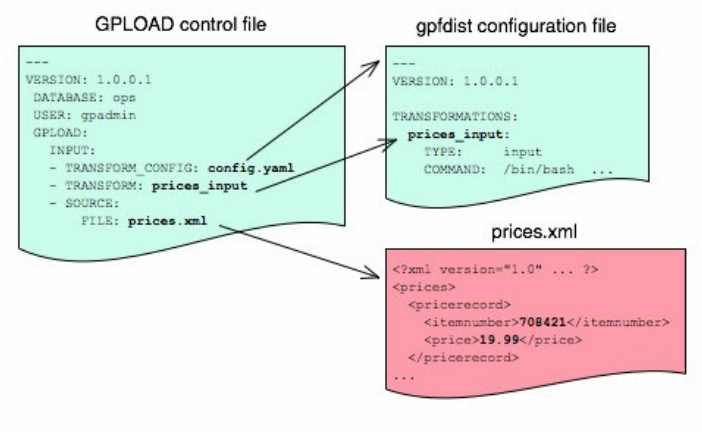
```
---
VERSION: 1.0.0.1
DATABASE: ops
USER: gpadmin
GPLOAD:
  INPUT:
```

```
    - 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 gpfdist and INSERT INTO SELECT FROM

With this load method, you perform each of the tasks that `gpload` automates. You start `gpfdist`, create an external table, load the data, and clean up by dropping the table and stopping `gpfdist`.

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 `gpfdist` configuration file.

The following query then loads the 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.

```
---
```

```
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.

STDERR

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 an 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 repo `github.com://greenplum-db/gpdb` in the `gpMgmt/demo/gpfdist_transform` directory. 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 '|');
```

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 version="1.0" encoding="UTF-8"?>
<Return returnVersion="2006v2.0"
   xmlns="https://www.irs.gov/efile"
   xmlns:efile="https://www.irs.gov/efile"
   xsi:schemaLocation="https://www.irs.gov/efile"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <ReturnHeader binaryAttachmentCount="1">
     <ReturnId>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>
       <EIN>011248772</EIN>
       ... more data ...
     </Filer>
     <Preparer>
       <Name>Percy Polar</Name>
       ... more data ...
     </Preparer>
     <TaxYear>2005</TaxYear>
   </ReturnHeader>
   ... more data ..
```

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:

```
AAAAAAAAAAAAAAAAAAAA|<Return returnVersion="2006v2.0"...
```

Load the data into Greenplum Database.

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/.

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 ...
```

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 (|).

```
W-12|6507/7-A-42|xr31|Deep Drill #5|Deep Drilling Co.|John
Doe|John.Doe@example.com|
```

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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Loading and Unloading Data

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 `NOT NULL`, `CHECK`, or `UNIQUE` constraint, the operation fails and no data loads.

Specifying `SEGMENT REJECT LIMIT` runs the `COPY` operation in single row error isolation mode. Specify the acceptable number of error rows on each segment, after which the entire `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 `COPY` operation does not reach the error limit, Greenplum loads all correctly-formatted rows and discards the error rows. Use the `LOG ERRORS` clause to capture data formatting errors internally in Greenplum Database. For example:

```
=> COPY country FROM '/data/gpdb/country_data'
   WITH DELIMITER '|' LOG ERRORS
   SEGMENT REJECT LIMIT 10 ROWS;
```

See Viewing Bad Rows in the Error Log for information about investigating error rows.

Parent topic: Loading and Unloading Data

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Optimizing Data Load and Query Performance

Use the following tips to help optimize your data load and subsequent query performance.

- Drop indexes before loading data into existing tables.

  Creating an index on pre-existing data is faster than updating it incrementally as each row is loaded. You can temporarily increase the `maintenance_work_mem` server configuration parameter to help speed up `CREATE INDEX` commands, though load performance is affected. Drop and recreate indexes only when there are no active users on the system.

- Create indexes last when loading data into new tables. Create the table, load the data, and create any required indexes.

- Run `ANALYZE` after loading data. If you significantly altered the data in a table, run `ANALYZE` or `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.

- Run `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 `VACUUM` command to recover the wasted space.

Parent topic: Loading and Unloading Data

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 (`COPY`).

- **Defining a File-Based Writable External Table**
- **Defining a Command-Based Writable External Web Table**
- **Unloading Data Using a Writable External Table**
- **Unloading Data Using COPY**

Parent topic: Loading and Unloading Data

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Defining a File-Based Writable External Table

Writable external tables that output data to files use the Greenplum parallel file server program, gpfdist, or the Hadoop Distributed File System interface, `gphdfs` (deprecated).

Use the `CREATE WRITABLE EXTERNAL TABLE` command to define the external table and specify the location and format of the output files. See Using the Greenplum Parallel File Server (gpfdist) for instructions on setting up gpfdist for use with an external table and Accessing HDFS Data with gphdfs (Deprecated) for instructions on setting up `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)**
- **Example 2—Hadoop file server (gphdfs (Deprecated))**

Parent topic: Unloading Data from Greenplum Database

Example 1—Greenplum file server (gpfdist)

```
=# CREATE WRITABLE EXTERNAL TABLE unload_expenses
   ( LIKE expenses )
   LOCATION ('gpfdist://etlhost-1:8081/expenses1.out',
             'gpfdist://etlhost-2:8081/expenses2.out')
 FORMAT 'TEXT' (DELIMITER ',')
 DISTRIBUTED BY (exp_id);
```

Parent topic: Defining a File-Based Writable External Table

Example 2—Hadoop file server (gphdfs (Deprecated))

```
=# 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.

Parent topic: Defining a File-Based Writable External Table

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:

```
=# CREATE WRITABLE EXTERNAL WEB TABLE output (output text)
    EXECUTE 'export PATH=$PATH:/home/gpadmin
            /programs;
    myprogram.sh'
    FORMAT 'TEXT'
    DISTRIBUTED RANDOMLY;
```

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 1. External Table EXECUTE Variables

Variable	Description
$GP_CID	Command count of the transaction executing the external table statement.
$GP_DATABASE	The database in which the external table definition resides.
$GP_DATE	The date on which the external table command ran.
$GP_MASTER_HOST	The host name of the Greenplum master host from which the external table statement was dispatched.
$GP_MASTER_PORT	The port number of the Greenplum master instance from which the external table statement was dispatched.
$GP_QUERY_STRING	The SQL command (DML or SQL query) executed by Greenplum Database.
$GP_SEG_DATADIR	The location of the data directory of the segment instance executing the external table command.
$GP_SEG_PG_CONF	The location of the `postgresql.conf` file of the segment instance executing the external table command.
$GP_SEG_PORT	The port number of the segment instance executing the external table command.
$GP_SEGMENT_COUNT	The total number of primary segment instances in the Greenplum Database system.
$GP_SEGMENT_ID	The ID number of the segment instance executing the external table command (same as `dbid` in `gp_segment_configuration`).
$GP_SESSION_ID	The database session identifier number associated with the external table statement.

Table 1. External Table EXECUTE Variables

Variable	Description
$GP_SN	Serial number of the external table scan node in the query plan of the external table statement.
$GP_TIME	The time the external table command was executed.
$GP_USER	The database user executing the external table statement.
$GP_XID	The transaction ID of the external table statement.

- **Disabling EXECUTE for Web or Writable External Tables**

Parent topic: Unloading Data from Greenplum Database

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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
```

Parent topic: Defining a Command-Based Writable External Web Table

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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;
```

Parent topic: Unloading Data from Greenplum Database

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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';
```

Parent topic: Unloading Data from Greenplum Database

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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**
- **Formatting Columns**
- **Representing NULL Values**
- **Escaping**
- **Character Encoding**

Parent topic: Loading and Unloading Data

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Formatting Data Files

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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:

```
=# CREATE EXTERNAL TABLE ext_table (name text, date date)
LOCATION ('gpfdist://<hostname>/filename.txt)
FORMAT 'TEXT' (DELIMITER '|');
```

Parent topic: Formatting Data Files

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Formatting Data Files

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**
- **Escaping in CSV Formatted Files**

Parent topic: Formatting Data 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 hexidecimal 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 hexidecimal (`\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.

Parent topic: Escaping

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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"""
```

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.
Parent topic: Escaping

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Note: If you *change* the `ENCODING` value in an existing `gpload` control file, you must manually drop any external tables that were creating using the previous `ENCODING` configuration. `gpload` does not drop and recreate external tables to use the new `ENCODING` if `REUSE_TABLES` is set to `true`.

Parent topic: Formatting Data Files

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;
```

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Example Custom Data Access Protocol

The following is the API for the Greenplum Database custom data access protocol. The example protocol implementation gpextprotocal.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.

```
/* ---- 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 gpextprotocal.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**

Parent topic: Loading and Unloading Data

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 gpextprotocal.c cc -shared -o gpextprotocal.so gpextprotocal.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 `gpextprotocal.so`. Refer to the GCC manual for more information on the `cc` options.

The header files that are declared as include files in `gpextprotocal.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.

- **gpextprotocal.c**

Parent topic: Example Custom Data Access Protocol

gpextprotocal.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);
}
```

Parent topic: Installing the External Table Protocol

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

- **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.
- **Common Causes of Performance Issues**
  This section explains the troubleshooting processes for common performance issues and potential solutions to these issues.
- **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.
- **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.
- **Investigating a Performance Problem**
  This section provides guidelines for identifying and troubleshooting performance problems in a Greenplum Database system.

Parent topic: Greenplum Database Administrator Guide

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Managing Performance

Understanding the Performance Factors

Several key performance factors influence database performance. Understanding these factors helps identify performance opportunities and avoid problems:

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

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

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Common Causes of Performance Issues

This section explains the troubleshooting processes for common performance issues and potential solutions to these issues.

Parent topic: Managing Performance

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 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*

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 changing `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 1. Maximum Limits of Greenplum Database

Dimension	Limit
Database Size	Unlimited
Table Size	Unlimited, 128 TB per partition per segment
Row Size	1.6 TB (1600 columns * 1 GB)
Field Size	1 GB
Rows per Table	281474976710656 (2^48)
Columns per Table/View	1600
Indexes per Table	Unlimited
Columns per Index	32

Table 1. Maximum Limits of Greenplum Database

Dimension	Limit
Table-level Constraints per Table	Unlimited
Table Name Length	63 Bytes (Limited by *name* data type)

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).

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

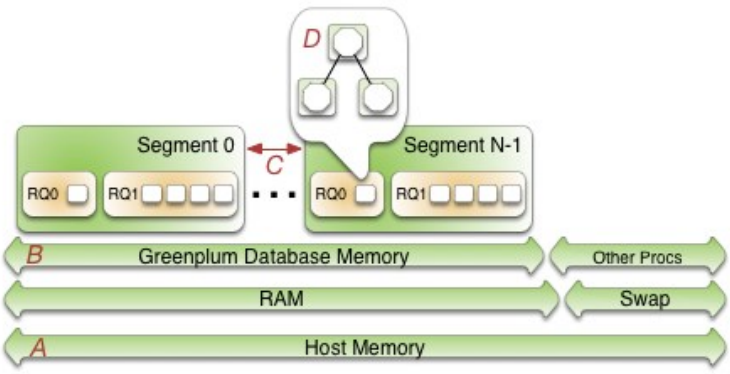
For best query throughput, the memory configuration should be managed carefully. There are memory configuration options at every level in Greenplum Database, from operating system parameters, to managing resources with resource queues and resource groups, to setting the amount of memory allocated to 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 when resource queue-based resource management is active.

Figure 1. 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.

Configuring vm.overcommit_ratio when Resource Group-Based Resource Management is Active

When resource group-based resource management is active, tune the operating system `vm.overcommit_ratio` as necessary. If your memory utilization is too low, increase the value; if your memory or swap usage is too high, decrease the setting.

Configuring vm.overcommit_ratio when Resource Queue-Based Resource Management is Active

To calculate a safe value for `vm.overcommit_ratio` when resource queue-based resource management is active, first determine the total memory available to Greenplum Database processes, `gp_vmem_rq`, with this formula:

```
gp_vmem_rq = ((SWAP + RAM) - (7.5GB + 0.05 * 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. When resource queue-based resource management is active, use `gp_vmem_rq` to calculate the `vm.overcommit_ratio` value with this formula:

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

Configuring Segment Memory when Resource Group-Based Resource Management is Active

When resource group-based resource management is active, the amount of memory allocated to each segment on a segment host is the memory available to Greenplum Database multiplied by the *gp_resource_group_memory_limit* server configuration parameter and divided by the number of active primary segments on the host. Use the following formula to calculate segment memory when using resource groups for resource management.

```
rg_perseg_mem = ((RAM * (vm.overcommit_ratio / 100) + SWAP) * gp_resource_group_memory
_limit) / num_active_primary_segments
```

Resource groups expose additional configuration parameters that enable you to further control and refine the amount of memory allocated for queries.

Configuring Segment Memory when Resource Queue-Based Resource Management is Active

When resource queue-based resource management is active, 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_rq` value that you calculated earlier.

```
gp_vmem_protect_limit = gp_vmem_rq / 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.

gp_vmem_protect_limit does not affect segment memory when using resource groups for Greenplum Database resource management.

Resource queues expose additional configuration parameters that enable you to further control and refine the amount of memory allocated for queries.

Example Memory Configuration Calculations

This section provides example memory calculations for resource queues and resource groups for a Greenplum Database system with the following specifications:

- 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

Resource Group Example

When resource group-based resource management is active in Greenplum Database, the usable memory available on a host is a function of the amount of RAM and swap space configured for the system, as well as the `vm.overcommit_ratio` system parameter setting:

```
total_node_usable_memory = RAM * (vm.overcommit_ratio / 100) + Swap
                         = 256GB * (50/100) + 64GB
                         = 192GB
```

Assuming the default `gp_resource_group_memory_limit` value (.7), the memory allocated to a Greenplum Database host with the example configuration is:

```
total_gp_memory = total_node_usable_memory * gp_resource_group_memory_limit
                = 192GB * .7
                = 134.4GB
```

The memory available to a Greenplum Database segment on a segment host is a function of the memory reserved for Greenplum on the host and the number of active primary segments on the host. On cluster startup:

```
gp_seg_memory = total_gp_memory / number_of_active_primary_segments
              = 134.4GB / 8
              = 16.8GB
```

Note that when 3 mirror segments switch to primary segments, the per-segment memory is still 16.8GB. Total memory usage on the segment host may approach:

```
total_gp_memory_with_primaries = 16.8GB * 11 = 184.8GB
```

Resource Queue Example

The `vm.overcommit_ratio` calculation for the example system when resource queue-based resource management is active in Greenplum Database follows:

```
gp_vmem_rq = ((SWAP + RAM) - (7.5GB + 0.05 * RAM)) / 1.7
        = ((64 + 256) - (7.5 + 0.05 * 256)) / 1.7
        = 176

vm.overcommit_ratio = (RAM - (0.026 * gp_vmem_rq)) / RAM
                    = (256 - (0.026 * 176)) / 256
                    = .982
```

You would set `vm.overcommit_ratio` of the example system to 98.

The `gp_vmem_protect_limit` calculation when resource queue-based resource management is active in Greenplum Database:

```
gp_vmem_protect_limit = gp_vmem_rq / maximum_acting_primary_segments
                      = 176 / 11
                      = 16GB
                      = 16384MB
```

You would set the `gp_vmem_protect_limit` server configuration parameter on the example system to 16384.

Parent topic: Managing Performance

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.

Important: 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, and on SuSE 12 SP2/SP3 systems with kernel version 4.4.73-5.1 or newer.

If you use RedHat 6 and the performance with resource groups is acceptable for your use case, upgrade your kernel to version 2.6.32-696 or higher to benefit from other fixes to the cgroups implementation.

SuSE 11 does not have a kernel version that resolves this issue; resource groups are still considered to be a Beta feature on this platform. Resource groups are not supported on SuSE 11 for production use.

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.

Metric	Resource Queues	Resource Groups
Concurrency	Managed at the query level	Managed at the transaction level
CPU	Specify query priority	Specify percentage of CPU resources; uses Linux Control Groups
Memory	Managed at the queue and operator level; users can over-subscribe	Managed at the transaction level, with enhanced allocation and tracking; users cannot over-subscribe
Memory Isolation	None	Memory is isolated between resource groups and between transactions within the same resource group
Users	Limits are applied only to non-admin users	Limits are applied to `SUPERUSER` and non-admin users alike
Queueing	Queue only when no slot available	Queue when no slot is available or not enough available memory
Query Failure	Query may fail immediately if not enough memory	Query may fail after reaching transaction fixed memory limit when no shared resource group memory exists and the transaction requests more memory
Limit Bypass	Limits are not enforced for `SUPERUSER` roles and certain operators and functions	Limits are not enforced on `SET`, `RESET`, and `SHOW` commands

Metric	Resource Queues	Resource Groups
External Components	None	Manage PL/Container CPU and memory resources

- **Using Resource Groups**
- **Using Resource Queues**

Parent topic: Managing Performance

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Using Resource Groups

You use resource groups to set and enforce CPU, memory, and concurrent transaction limits in Greenplum Database. After you define a resource group, you can then assign the group to one or more Greenplum Database roles, or to an external component such as PL/Container, in order to control the resources used by those roles or components.

When you assign a resource group to a role (a role-based resource group), the resource limits that you define for the group apply to all of the roles to which you assign the group. For example, the memory limit for a resource group identifies the maximum memory usage for all running transactions submitted by Greenplum Database users in all roles to which you assign the group.

Similarly, when you assign a resource group to an external component, the group limits apply to all running instances of the component. For example, if you create a resource group for a PL/Container external component, the memory limit that you define for the group specifies the maximum memory usage for all running instances of each PL/Container runtime to which you assign the group.

This topic includes the following subtopics:

Parent topic: Managing Resources

Understanding Role and Component Resource Groups

Greenplum Database supports two types of resource groups: groups that manage resources for roles, and groups that manage resources for external components such as PL/Container.

The most common application for resource groups is to manage the number of active queries that different roles may execute concurrently in your Greenplum Database cluster. You can also manage the amount of CPU and memory resources that Greenplum allocates to each query.

Resource groups for roles use Linux control groups (cgroups) for CPU resource management. Greenplum Database tracks virtual memory internally for these resource groups using a memory auditor referred to as `vmtracker`.

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 for roles, 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.

You can also use resource groups to manage the CPU and memory resources of external components such as PL/Container. Resource groups for external components use Linux cgroups to manage both the total CPU and total memory resources for the component.

Note: Containerized deployments of Greenplum Database, such as Greenplum for Kubernetes, might create a hierarchical set of nested cgroups to manage host system resources. The nesting of cgroups affects the Greenplum Database resource group limits for CPU percentage, CPU cores, and memory (except for Greenplum Database external components). The Greenplum Database resource group system resource limit is based on the quota for the parent group.

For example, Greenplum Database is running in a cgroup demo, and the Greenplum Database cgroup is nested in the cgroup demo. If the cgroup demo is configured with a CPU limit of 60% of system CPU resources and the Greenplum Database resource group CPU limit is set 90%, the Greenplum Database limit of host system CPU resources is 54% (0.6 x 0.9).

Nested cgroups do not affect memory limits for Greenplum Database external components such as PL/Container. Memory limits for external components can only be managed if the cgroup that is used to manage Greenplum Database resources is not nested, the cgroup is configured as a top-level cgroup.

For information about configuring cgroups for use by resource groups, see Configuring and Using Resource Groups.

Resource Group Attributes and Limits

When you create a resource group, you:

- Specify the type of resource group by identifying how memory for the group is audited.
- Provide a set of limits that determine the amount of CPU and memory resources available to the group.

Resource group attributes and limits:

Limit Type	Description
MEMORY_AUDITOR	The memory auditor in use for the resource group. `vmtracker` (the default) is required if you want to assign the resource group to roles. Specify `cgroup` to assign the resource group to an external component.
CONCURRENCY	The maximum number of concurrent transactions, including active and idle transactions, that are permitted in the resource group.
CPU_RATE_LIMIT	The percentage of CPU resources available to this resource group.
CPUSET	The CPU cores to reserve for this resource group.

Limit Type	Description
MEMORY_LIMIT	The percentage of reserved memory resources available to this resource group.
MEMORY_SHARED_QUOTA	The percentage of reserved memory to share across transactions submitted in this resource group.
MEMORY_SPILL_RATIO	The memory usage threshold for memory-intensive transactions. When a transaction reaches this threshold, it spills to disk.

Note: Resource limits are not enforced on `SET`, `RESET`, and `SHOW` commands.

Memory Auditor

The `MEMORY_AUDITOR` attribute specifies the type of resource group by identifying the memory auditor for the group. A resource group that specifies the `vmtracker` `MEMORY_AUDITOR` identifies a resource group for roles. A resource group specifying the `cgroup` `MEMORY_AUDITOR` identifies a resource group for external components.

The default `MEMORY_AUDITOR` is `vmtracker`.

The `MEMORY_AUDITOR` that you specify for a resource group determines if and how Greenplum Database uses the limit attributes to manage CPU and memory resources:

Limit Type	Resource Group for Roles	Resource Group for External Components
CONCURRENCY	Yes	No; must be zero (0)
CPU_RATE_LIMIT	Yes	Yes
CPUSET	Yes	Yes
MEMORY_LIMIT	Yes	Yes
MEMORY_SHARED_QUOTA	Yes	Component-specific
MEMORY_SPILL_RATIO	Yes	Component-specific

Note: For queries managed by resource groups that are configured to use the `vmtracker` memory auditor, Greenplum Database supports the automatic termination of queries based on the amount of memory the queries are using. See the server configuration parameter `runaway_detector_activation_percent`.

Transaction Concurrency Limit

The `CONCURRENCY` limit controls the maximum number of concurrent transactions permitted for a resource group for roles.
Note: The `CONCURRENCY` limit is not applicable to resource groups for external components and must be set to zero (0) for such groups.

Each resource group for roles 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 for roles 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.

You can set the server configuration parameter `gp_resource_group_bypass` to bypass a resource group concurrency limit.

CPU Limits

You configure the share of CPU resources to reserve for a resource group on a segment host by assigning specific CPU core(s) to the group, or by identifying the percentage of segment CPU resources to allocate to the group. Greenplum Database uses the `CPUSET` and `CPU_RATE_LIMIT` resource group limits to identify the CPU resource allocation mode. You must specify only one of these limits when you configure a resource group.

You may employ both modes of CPU resource allocation simultaneously in your Greenplum Database cluster. You may also change the CPU resource allocation mode for a resource group at runtime.

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 host. This limit governs the maximum CPU usage of all resource groups on a segment host regardless of the CPU allocation mode configured for the group. The remaining unreserved 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.

Assigning CPU Resources by Core

You identify the CPU cores that you want to reserve for a resource group with the `CPUSET` property. The CPU cores that you specify must be available in the system and cannot overlap with any CPU cores that you reserved for other resource groups. (Although Greenplum Database uses the cores that you assign to a resource group exclusively for that group, note that those CPU cores may also be used by non-Greenplum processes in the system.)

Specify a comma-separated list of single core numbers or number intervals when you configure `CPUSET`. You must enclose the core numbers/intervals in single quotes, for example, '1,3-4'.

When you assign CPU cores to `CPUSET` groups, consider the following:

- A resource group that you create with `CPUSET` uses the specified cores exclusively. If there are no running queries in the group, the reserved cores are idle and cannot be used by queries in other resource groups. Consider minimizing the number of `CPUSET` groups to avoid wasting system CPU resources.
- Consider keeping CPU core 0 unassigned. CPU core 0 is used as a fallback mechanism in the following cases:
    - `admin_group` and `default_group` require at least one CPU core. When all CPU cores are reserved, Greenplum Database assigns CPU core 0 to these default groups. In this situation, the resource group to which you assigned CPU core 0 shares the core with `admin_group` and `default_group`.
    - If you restart your Greenplum Database cluster with one node replacement and the node does not have enough cores to service all `CPUSET` resource groups, the groups are automatically assigned CPU core 0 to avoid system start failure.
- Use the lowest possible core numbers when you assign cores to resource groups. If you replace a Greenplum Database node and the new node has fewer CPU cores than the original, or if you back up the database and want to restore it on a cluster with nodes with fewer CPU cores, the operation may fail. For example, if your Greenplum Database cluster has 16 cores, assigning cores 1-7 is optimal. If you create a resource group and assign CPU core 9 to this group, database restore to an 8 core node will fail.

Resource groups that you configure with `CPUSET` have a higher priority on CPU resources. The

maximum CPU resource usage percentage for all resource groups configured with `CPUSET` on a segment host is the number of CPU cores reserved divided by the number of all CPU cores, multiplied by 100.

When you configure `CPUSET` for a resource group, Greenplum Database disables `CPU_RATE_LIMIT` for the group and sets the value to -1.

Note: You must configure `CPUSET` for a resource group *after* you have enabled resource group-based resource management for your Greenplum Database cluster.

Assigning CPU Resources by Percentage

The Greenplum Database node CPU percentage is divided equally among each segment on the Greenplum node. Each resource group that you configure with a `CPU_RATE_LIMIT` reserves the specified percentage of the segment CPU for resource management.

The minimum `CPU_RATE_LIMIT` percentage you can specify for a resource group is 1, the maximum is 100.

The sum of `CPU_RATE_LIMIT`s specified for all resource groups that you define in your Greenplum Database cluster must not exceed 100.

The maximum CPU resource usage for all resource groups configured with a `CPU_RATE_LIMIT` on a segment host is the minimum of:

- The number of non-reserved CPU cores divided by the number of all CPU cores, multiplied by 100, and
- The `gp_resource_group_cpu_limit` value.

CPU resource assignment for resource groups configured with a `CPU_RATE_LIMIT` 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_LIMIT`s. 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 that you create with a `CPU_RATE_LIMIT` of 20.

When you configure `CPU_RATE_LIMIT` for a resource group, Greenplum Database disables `CPUSET` for the group and sets the value to -1.

Memory Limits

When resource groups are enabled, memory usage is managed at the Greenplum Database node, segment, and resource group levels. You can also manage memory at the transaction level with a resource group for roles.

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 host. The default `gp_resource_group_memory_limit` value is .7 (70%).

The memory resource available on a Greenplum Database node is further divided equally among each segment on the node. When resource group-based resource management is active, the amount of memory allocated to each segment on a segment host is the memory available to Greenplum Database multiplied by the `gp_resource_group_memory_limit` server configuration parameter and divided by the number of active primary segments on the host:

```
rg_perseg_mem = ((RAM * (vm.overcommit_ratio / 100) + SWAP) * gp_resource_group_memory
_limit) / num_active_primary_segments
```

Each resource group may reserve a percentage of the segment memory for resource management. You identify this percentage via the `MEMORY_LIMIT` value that you specify when you create the resource group. The minimum `MEMORY_LIMIT` percentage you can specify for a resource group is

0, the maximum is 100. When `MEMORY_LIMIT` is 0, Greenplum Database reserves no memory for the resource group, but uses resource group global shared memory to fulfill all memory requests in the group. Refer to Global Shared Memory for more information about resource group global shared memory.

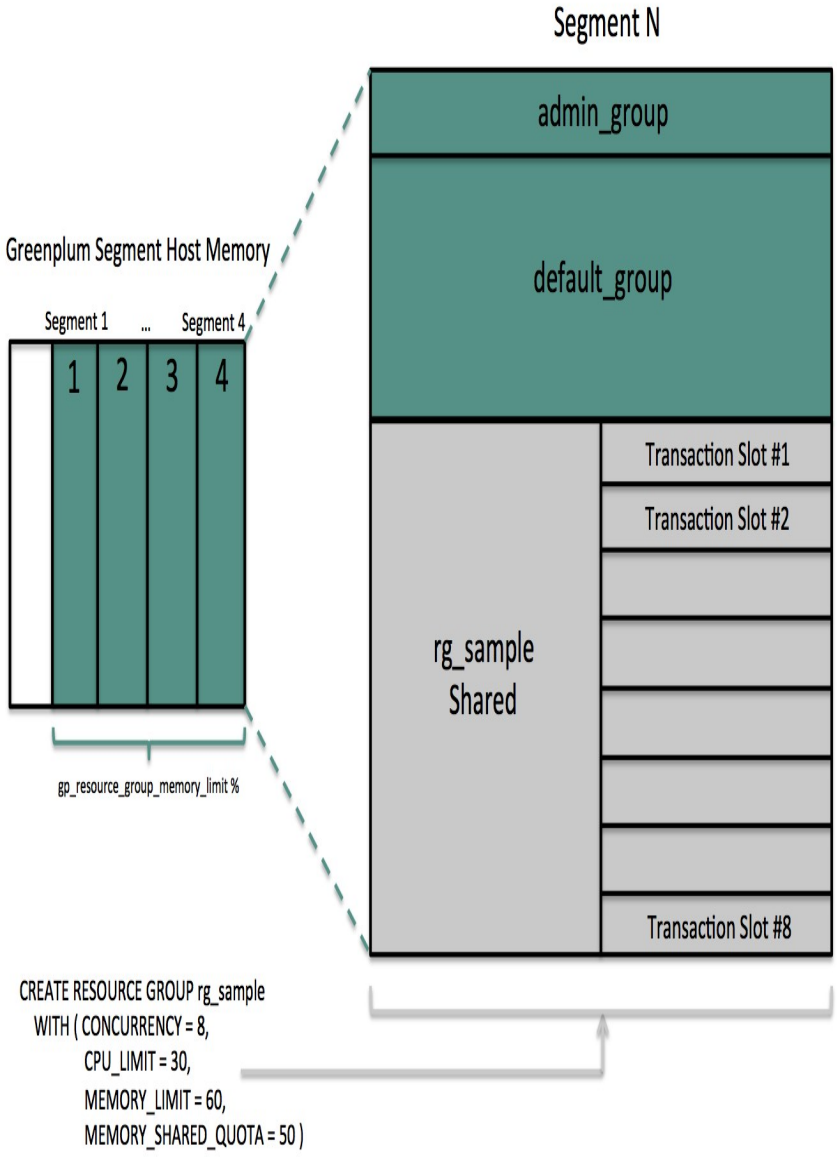
The sum of `MEMORY_LIMIT`s specified for all resource groups that you define in your Greenplum Database cluster must not exceed 100.

Additional Memory Limits for Role-based Resource Groups

If resource group memory is reserved for roles (non-zero `MEMORY_LIMIT`), the memory is further divided into fixed and shared components. The `MEMORY_SHARED_QUOTA` value that 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` that 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 a resource group for roles. If fixed memory is reserved by a resource group (non-zero `MEMORY_LIMIT`), it is divided into `CONCURRENCY` number of transaction slots. Each slot is allocated a fixed, equal amount of the resource group memory. Greenplum Database guarantees this fixed memory to each transaction.
Figure 1. 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.

Global Shared Memory

The sum of the `MEMORY_LIMIT`s configured for all resource groups (including the default `admin_group` and `default_group` groups) identifies the percentage of reserved resource group

memory. If this sum is less than 100, Greenplum Database allocates any unreserved memory to a resource group global shared memory pool.

Resource group global shared memory is available only to resource groups that you configure with the `vmtracker` memory auditor.

When available, Greenplum Database allocates global shared memory to a transaction after first allocating slot and resource group shared memory (if applicable). Greenplum Database allocates resource group global shared memory to transactions on a first-come first-served basis.

Note: Greenplum Database tracks, but does not actively monitor, transaction memory usage in resource groups. If the memory usage for a resource group exceeds its fixed memory allotment, a transaction in the resource group fails when *all* of these conditions are met:

- No available resource group shared memory exists.
- No available global shared memory exists.
- The transaction requests additional memory.

Greenplum Database uses resource group memory more efficiently when you leave some memory (for example, 10-20%) unallocated for the global shared memory pool. The availability of global shared memory also helps to mitigate the failure of memory-consuming or unpredicted queries.

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 this threshold is reached, a transaction spills to disk. Greenplum Database uses the `MEMORY_SPILL_RATIO` to determine the initial memory to allocate to a transaction.

You can specify an integer percentage value from 0 to 100 inclusive for `MEMORY_SPILL_RATIO`. The default `MEMORY_SPILL_RATIO` is 20.

When `MEMORY_SPILL_RATIO` is 0, Greenplum Database uses the `statement_mem` server configuration parameter value to control initial query operator memory.

Note: When you set `MEMORY_LIMIT` to 0, `MEMORY_SPILL_RATIO` must also be set to 0.

You can selectively set the `MEMORY_SPILL_RATIO` on a per-query basis at the session level with the `memory_spill_ratio` server configuration parameter.

memory_spill_ratio and Low Memory Queries

A low `statement_mem` setting (for example, in the 10MB range) has been shown to increase the performance of queries with low memory requirements. Use the `memory_spill_ratio` and `statement_mem` server configuration parameters to override the setting on a per-query basis. For example:

```
SET memory_spill_ratio=0;
SET statement_mem='10 MB';
```

About Using Reserved Resource Group Memory vs. Using Resource Group Global Shared Memory

When you do not reserve memory for a resource group (`MEMORY_LIMIT` and `MEMORY_SPILL_RATIO` are set to 0):

- It increases the size of the resource group global shared memory pool.
- The resource group functions similarly to a resource queue, using the `statement_mem` server configuration parameter value to control initial query operator memory.
- Any query submitted in the resource group competes for resource group global shared memory on a first-come, first-served basis with queries running in other groups.
- There is no guarantee that Greenplum Database will be able to allocate memory for a query running in the resource group. The risk of a query in the group encountering an out of memory (OOM) condition increases when there are many concurrent queries consuming memory from the resource group global shared memory pool at the same time.

To reduce the risk of OOM for a query running in an important resource group, consider reserving some fixed memory for the group. While reserving fixed memory for a group reduces the size of the resource group global shared memory pool, this may be a fair tradeoff to reduce the risk of encountering an OOM condition in a query running in a critical resource group.

Other Memory Considerations

Resource groups for roles track all Greenplum Database memory allocated via the `palloc()` function. Memory that you allocate using the Linux `malloc()` function is not managed by these resource groups. To ensure that resource groups for roles are accurately tracking memory usage, avoid using `malloc()` to allocate large amounts of memory in custom Greenplum Database user-defined functions.

Using Greenplum Command Center to Manage Resource Groups

Using Pivotal Greenplum Command Center, an administrator can create and manage resource groups, change roles' resource groups, and create workload management rules.

Workload management rules are defined in Command Center and stored in Greenplum Database. When a transaction is submitted, Greenplum Database calls the workload management database extension to evaluate and apply the rules.

Workload management assignment rules assign transactions to different resource groups based on user-defined criteria. If no assignment rule is matched, Greenplum Database assigns the transaction to the role's default resource group. Workload management idle session kill rules set the maximum number of seconds that sessions managed by a resource group can remain idle before they are terminated.

Refer to the Greenplum Command Center documentation for more information about creating and managing resource groups and workload management rules.

Configuring and Using Resource Groups

Important: 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, and on SuSE 12 SP2/SP3 systems with kernel version 4.4.73-5.1 or newer.

If you use RedHat 6 and the performance with resource groups is acceptable for your use case, upgrade your kernel to version 2.6.32-696 or higher to benefit from other fixes to the cgroups implementation.

SuSE 11 does not have a kernel version that resolves this issue; resource groups are still considered to be a Beta feature on this platform. Resource groups are not supported on SuSE 11 for production use.

Prerequisite

Greenplum Database resource groups use Linux Control Groups (cgroups) to manage CPU resources. Greenplum Database also uses cgroups to manage memory for resource groups for external components. With cgroups, Greenplum isolates the CPU and external component memory usage of your Greenplum processes from other processes on the node. This allows Greenplum to support CPU and external component memory 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`:

   ```
   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 you plan to use resource groups to manage PL/Container instances, you must add a memory block to the `gpdb.conf` file. Be sure to add the block before the closing curly-brace (`}`):

   ```
        cpuacct {
        }
        memory {
     }
    }
   ```

5. If you plan to assign specific CPU cores to resource groups, you must add a cpuset block to

the `gpdb.conf` file. Be sure to add the block before the closing curly-brace (`}`):

```
    cpuacct {
    }
    cpuset {
  }
}
```

6. 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 that 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:

     ```
     sudo yum install libcgroup-tools
     sudo cgconfigparser -l /etc/cgconfig.d/gpdb.conf
     ```

   - Redhat/CentOS 6.x systems:

     ```
     sudo yum install libcgroup
     sudo service cgconfig start
     ```

   - SuSE 11+ systems:

     ```
     sudo zypper install libcgroup-tools
     sudo cgconfigparser -l /etc/cgconfig.d/gpdb.conf
     ```

7. Identify the `cgroup` directory mount point for the node:

   ```
   grep cgroup /proc/mounts
   ```

   The first line of output identifies the `cgroup` mount point.

8. Verify that you set up the Greenplum Database cgroups configuration correctly by running the following commands. Replace <cgroup_mount_point> with the mount point that you identified in the previous step:

   ```
   ls -l <cgroup_mount_point>/cpu/gpdb
   ls -l <cgroup_mount_point>/cpuacct/gpdb
   ls -l <cgroup_mount_point>/cpuset/gpdb
   ls -l <cgroup_mount_point>/memory/gpdb
   ```

   If these directories exist and are owned by `gpadmin:gpadmin`, you have successfully configured cgroups for Greenplum Database CPU resource management.

9. To automatically recreate Greenplum Database required cgroup hierarchies and parameters when your system is restarted, configure your system to enable the Linux cgroup service daemon `cgconfig.service` (Redhat/CentOS 7.x and SuSE 11+) or `cgconfig` (Redhat/CentOS 6.x) at node start-up. For example, configure one of the following cgroup service commands in your preferred service auto-start tool:
   - Redhat/CentOS 7.x and SuSE11+ systems:

     ```
     sudo systemctl enable cgconfig.service
     ```

     To start the service immediately (without having to reboot) enter:

     ```
     sudo systemctl start cgconfig.service
     ```

   - Redhat/CentOS 6.x systems:

     ```
     sudo chkconfig cgconfig on
     ```

You may choose a different method to recreate the Greenplum Database resource group cgroup hierarchies.

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. Similarly, CPU and memory usage by an external component is governed by the CPU and memory limits configured for the resource group assigned to the component.

Greenplum Database creates two default resource groups for roles named `admin_group` and `default_group`. When you enable resources 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:

Limit Type	admin_group	default_group
CONCURRENCY	10	20
CPU_RATE_LIMIT	10	30
CPUSET	-1	-1
MEMORY_LIMIT	10	30
MEMORY_SHARED_QUOTA	50	50
MEMORY_SPILL_RATIO	20	20
MEMORY_AUDITOR	vmtracker	vmtracker

Keep in mind that the `CPU_RATE_LIMIT` and `MEMORY_LIMIT` values for the default resource groups `admin_group` and `default_group` contribute to the total percentages on a segment host. You may find that you need to adjust these limits for `admin_group` and/or `default_group` as you create and add new resource groups to your Greenplum Database deployment.

Creating Resource Groups

When you create a resource group for a role, you provide a name, a CPU resource allocation mode, and a 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 for a role, you must provide `CPU_RATE_LIMIT` or `CPUSET` and `MEMORY_LIMIT` limit values. These limits identify the percentage of Greenplum Database resources to allocate to this resource group. You specify a `MEMORY_LIMIT` to reserve a fixed amount of memory for the resource group. If you specify a `MEMORY_LIMIT` of 0, Greenplum Database uses global shared memory to fulfill all memory requirements for the 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. `rgroup1` utilizes the default `MEMORY_AUDITOR` `vmtracker` and the default `CONCURRENCY` setting of 20.

When you create a resource group for an external component, you must provide `CPU_RATE_LIMIT` or `CPUSET` and `MEMORY_LIMIT` limit values. You must also provide the `MEMORY_AUDITOR` and explicitly set `CONCURRENCY` to zero (0). For example, to create a resource group named *rgroup_extcomp* for which you reserve CPU core 1 and assign a memory limit of 15:

```
=# CREATE RESOURCE GROUP rgroup_extcomp WITH (MEMORY_AUDITOR=cgroup, CONCURRENCY=0,
     CPUSET='1', MEMORY_LIMIT=15);
```

The `ALTER RESOURCE GROUP` command updates the limits of a resource group. To change the limits of a resource group, specify the new values that you want for the group. For example:

```
=# ALTER RESOURCE GROUP rg_role_light SET CONCURRENCY 7;
=# ALTER RESOURCE GROUP exec SET MEMORY_LIMIT 25;
=# ALTER RESOURCE GROUP rgroup1 SET CPUSET '2,4';
```

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 for a role, the group cannot be assigned to any role, nor can there be any transactions active or waiting in the resource group. Dropping a resource group for an external component in which there are running instances kills the running instances.

To drop a resource group:

```
=# DROP RESOURCE GROUP exec;
```

Configuring Automatic Query Termination

When resource groups have a global shared memory pool, the server configuration parameter `runaway_detector_activation_percent` sets the percent of utilized global shared memory that triggers the termination of queries that are managed by resource groups that are configured to use the `vmtracker` memory auditor, such as `admin_group` and `default_group`.

Resource groups have a global shared memory pool when the sum of the `MEMORY_LIMIT` attribute values configured for all resource groups is less than 100. For example, if you have 3 resource groups configured with `MEMORY_LIMIT` values of 10 , 20, and 30, then global shared memory is 40% = 100% - (10% + 20% + 30%).

For information about global shared memory, see Global Shared Memory.

Assigning a Resource Group to a Role

When you create a resource group with the default `MEMORY_AUDITOR` `vmtracker`, 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:

```
=# 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.

Note: You cannot assign a resource group that you create for an external component to a role.

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:

```
=# 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:

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

The `gp_toolkit.gp_resgroup_config` view does not display the resource group memory auditor or CPU core assignment attributes. To view the these attributes along with the other resource group limits, run the following query:

```
=# SELECT g.oid AS groupid, g.rsgname AS groupname,
     t1.value AS concurrency, t1.proposed AS proposed_concurrency,
     t2.value AS cpu_rate_limit, t3.value AS memory_limit,
     t3.proposed AS proposed_memory_limit,
     t4.value AS memory_shared_quota,
     t4.proposed AS proposed_memory_shared_quota,
     t5.value AS memory_spill_ratio,
     t5.proposed AS proposed_memory_spill_ratio,
     CASE
       WHEN t6.value IS NULL THEN 'vmtracker'::text
       WHEN t6.value = '0'::text THEN 'vmtracker'::text
       WHEN t6.value = '1'::text THEN 'cgroup'::text
       ELSE 'unknown'::text
     END AS memory_auditor,
     t7.value AS cpuset
```

```
  FROM pg_resgroup g
    JOIN pg_resgroupcapability t1 ON g.oid = t1.resgroupid AND t1.reslimittype = 1
    JOIN pg_resgroupcapability t2 ON g.oid = t2.resgroupid AND t2.reslimittype = 2
    JOIN pg_resgroupcapability t3 ON g.oid = t3.resgroupid AND t3.reslimittype = 3
    JOIN pg_resgroupcapability t4 ON g.oid = t4.resgroupid AND t4.reslimittype = 4
    JOIN pg_resgroupcapability t5 ON g.oid = t5.resgroupid AND t5.reslimittype = 5
    LEFT JOIN pg_resgroupcapability t6 ON g.oid = t6.resgroupid AND t6.reslimittype =
6
    LEFT JOIN pg_resgroupcapability t7 ON g.oid = t7.resgroupid AND t7.reslimittype =
7;
```

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:

```
=# 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:

```
=# 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:

```
=# SELECT current_query, waiting, rsgname, rsgqueueduration
     FROM pg_stat_activity;
```

`pg_stat_activity` displays information about the user/role that initiated a query. A query that uses an external component such as PL/Container is composed of two parts: the query operator that runs in Greenplum Database and the UDF that runs in a PL/Container instance. Greenplum Database processes the query operators under the resource group assigned to the role that initiated the query. A UDF running in a PL/Container instance runs under the resource group assigned to the PL/Container runtime. The latter is not represented in the `pg_stat_activity` view; Greenplum Database does not have any insight into how external components such as PL/Container manage memory in running instances.

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.

```
=# 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:

```
 rolname | rsgname  | procpid | waiting |     current_query     | datname
---------+----------+---------+---------+-----------------------+---------
  sammy  | rg_light |  31861  |    f    | <IDLE> in transaction | testdb
  billy  | rg_light |  31905  |    t    | SELECT * FROM topten; | testdb
```

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:

```
=# 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.

Resource Group Frequently Asked Questions

CPU

- **Why is CPU usage lower than the `CPU_RATE_LIMIT` configured for the resource group?**

  You may run into this situation when a low number of queries and slices are running in the resource group, and these processes are not utilizing all of the cores on the system.

- **Why is CPU usage for the resource group higher than the configured `CPU_RATE_LIMIT`?**
  This situation can occur in the following circumstances:
  - A resource group may utilize more CPU than its `CPU_RATE_LIMIT` when other resource groups are idle. In this situation, Greenplum Database allocates the CPU resource of an idle resource group to a busier one. This resource group feature is called CPU burst.
  - The operating system CPU scheduler may cause CPU usage to spike, then drop down. If you believe this might be occurring, calculate the average CPU usage within a given period of time (for example, 5 seconds) and use that average to determine if CPU usage is higher than the configured limit.

Memory

- **Why did my query return an "out of memory" error?**

  A transaction submitted in a resource group fails and exits when memory usage exceeds its fixed memory allotment, no available resource group shared memory exists, and the transaction requests more memory.

- **Why did my query return a "memory limit reached" error?**

  Greenplum Database automatically adjusts transaction and group memory to the new settings when you use `ALTER RESOURCE GROUP` to change a resource group's memory and/or concurrency limits. An "out of memory" error may occur if you recently altered resource group attributes and there is no longer a sufficient amount of memory available for a currently running query.

- **Why does the actual memory usage of my resource group exceed the amount configured for the group?**

  The actual memory usage of a resource group may exceed the configured amount when one or more queries running in the group is allocated memory from the global shared

memory pool. (If no global shared memory is available, queries fail and do not impact the memory resources of other resource groups.)

When global shared memory is available, memory usage may exceed the configured amount when a transaction spills to disk. Greenplum Database statements continue to request memory when they start to spill to disk because:

- Spilling to disk requires extra memory to work.
- Other operators may continue to request memory.

Memory usage grows in spill situations; when global shared memory is available, the resource group may eventually use up to 200-300% of its configured group memory limit.

Concurrency

- **Why is the number of running transactions lower than the `CONCURRENCY` limit configured for the resource group?**

  Greenplum Database considers memory availability before running a transaction, and will queue the transaction if there is not enough memory available to serve it. If you use `ALTER RESOURCE GROUP` to increase the `CONCURRENCY` limit for a resource group but do not also adjust memory limits, currently running transactions may be consuming all allotted memory resources for the group. When in this state, Greenplum Database queues subsequent transactions in the resource group.

- **Why is the number of running transactions in the resource group higher than the configured `CONCURRENCY` limit?**

  The resource group may be running `SET` and `SHOW` commands, which bypass resource group transaction checks.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Using Resource Queues

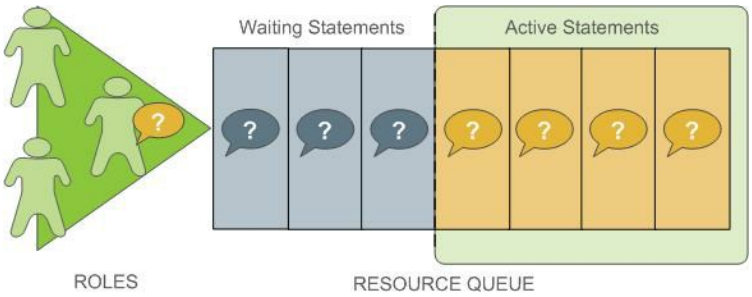
Use Greenplum Database resource queues to prioritize and allocate resources to queries according to business requirements and to prevent queries from starting when resources are unavailable.

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 1. Resource Queue Process



Resource queues define classes of queries with similar resource requirements. Administratrors 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: GPORCA and the legacy Greenplum Database query optimizer utilize different query costing models and may compute different costs for the same query. The Greenplum Database resource queue resource management scheme neither differentiates nor aligns costs between GPORCA and the legacy optimizer; it uses the literal cost value returned from the optimizer to throttle queries.

  When resource queue-based resource management is active, use the `MEMORY_LIMIT` and `ACTIVE_STATEMENTS` limits for resource queues rather than configuring cost-based limits. Even when using GPORCA, Greenplum Database may fall back to using the legacy query optimizer for certain queries, so using cost-based limits can lead to unexpected results.

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.

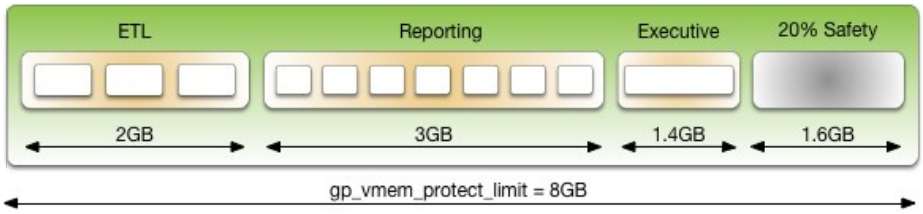
Also, an SQL statement that is run during the execution of an `EXPLAIN ANALYZE` command is excluded from resource queues.

Parent topic: Managing Resources

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:
Figure 2. 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.

Resource Queue Name	Active Statements	Memory Limit	Memory per Query
ETL	3	2GB	667MB
Reporting	7	3GB	429MB
Executive	1	1.4GB	1.4GB

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 increae `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.

statement_mem and Low Memory Queries

A low `statement_mem` setting (for example, in the 1-3MB range) has been shown to increase the performance of queries with low memory requirements. Use the `statement_mem` server configuration parameter to override the setting on a per-query basis. For example:

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

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.

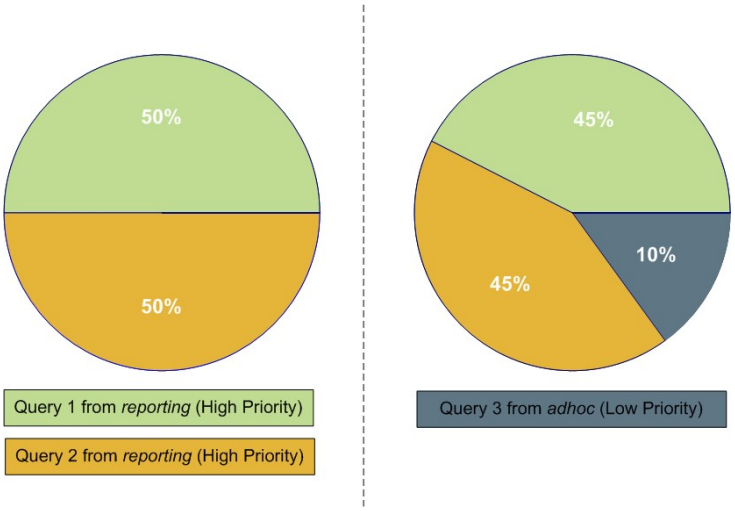
The comparative size or complexity of the queries does not affect the allotment of CPU. If a simple, low-cost query is running simultaneously with a large, complex query, and their priority settings are the same, they will be allocated the same share of available CPU resources. When a new query becomes 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 3. CPU share readjusted according to priority

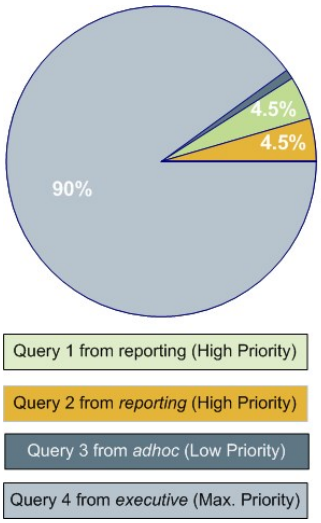


Note:

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 4. 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:

1. Configure resource managment. See Configuring Resource Management.
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, `shmmax` 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:

```
$ gpconfig --show gp_vmem_protect_limit
```

6. For example, to set one value on all segment instances and a different value on the master:

```
$ gpconfig -c gp_resqueue_priority_cpucores_per_segment -v 2 -m 8
```

7. Restart Greenplum Database to make the configuration changes effective:

```
$ 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:

```
=# 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):

```
=# 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:

```
=> 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:

```
=# 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:

```
=# 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:

```
=# 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:

```
=# 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:

```
=# 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:

```
=# 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:

```
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:

```
=# 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:

```
=# 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:

```
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:

```
=# 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:

```
=# 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:

```
=# 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:

```
=# 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:

```
=# 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:

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

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:

```
=# 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:

```
=# 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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Managing Performance

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)

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 that 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 groups, queries that are waiting will also show in *pg_locks*. To see how many queries are waiting to run in a resource group, use the *gp_resgroup_status* system catalog view. For example:

```
SELECT * FROM gp_toolkit.gp_resgroup_status;
```

Similarly, if you use resource queues, queries that are waiting in a queue also show in *pg_locks*. To see how many queries are waiting to run from a resource queue, use the *gp_resqueue_status* system catalog view. For example:

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

Checking Query Status and System Utilization

You can use system monitoring utilities such as `ps`, `top`, `iostat`, `vmstat`, `netstat` and so on to monitor database activity on the hosts in your Greenplum Database array. These tools can help identify Greenplum Database processes (`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 `gpssh` allows you to run these system monitoring commands on several hosts simultaneously.

You can create and use the Greenplum Database *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 Viewing Session Memory Usage Information.

You can enable a dedicated database, `gpperfmon`, in which data collection agents running on each segment host save query and system utilization metrics. Refer to the `gperfmon_install` management utility reference in the *Greenplum Database Management Utility Reference Guide* for help creating the `gpperfmon` database and managing the agents. See documentation for the tables and views in the `gpperfmon` database in the *Greenplum Database Reference Guide*.

The optional Greenplum Command Center web-based user interface graphically displays query and system utilization metrics saved in the `gpperfmon` database. See the Greenplum Command Center Documentation web site for procedures to enable Greenplum Command Center.

Troubleshooting Problem Queries

If a query performs poorly, look at its query plan to help identify problems. The `EXPLAIN` command shows the query plan for a given query. See 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.

Investigating Error Messages

Greenplum Database log messages are written to files in the `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: `gpdb-`*`YYYY-MM-DD_hhmmss.csv`*. To locate the log files on the master host:

```
$ cd $MASTER_DATA_DIRECTORY/pg_log
```

Log lines have the format of:

```
timestamp | user | database | statement_id | con#cmd#
|:-LOG_LEVEL: log_message
```

You may want to focus your search for `WARNING`, `ERROR`, `FATAL` or `PANIC` log level messages. You can use the Greenplum utility `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 on the GPMT page.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.
- **Client Utility Reference**
  Describes the command-line client utilities provided with Greenplum Database.
- **Additional Supplied Modules**
  This section describes additional modules available in the Greenplum Database `contrib` subdirectory.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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:`\[2620:0:170:610::11\]` or `'[2620:0:170:610::11]'`.

Greenplum Database provides the following management utility programs:

analyzedb	gplogfilter	gpmfr
gpactivatestandby	gpmapreduce	gpmovemirrors
gpaddmirrors		gpperfmon_install
gpbackup		gppkg
gpcheck		gprecoverseg
gpcheckcat		gpreload
gpcheckperf		gprestore
gpconfig		gp_restore (deprecated)
gpcopy		gpscp
gpcrondump		gpseginstall
gpdbrestore		gpssh
gpdeletesystem		gpssh-exkeys
gp_dump (deprecated)		gpstart
gpexpand		gpstate
gpfdist		gpstop
gpfilespace		gpsys1
gpinitstandby		gptransfer (deprecated)
gpinitsystem		pgbouncer
gpkafka		pgbouncer.ini
gpkafka check		pgbouncer-admin
gpkafka load		pxf
gpkafka-v2.yaml		pxf cluster
gpkafka.yaml		
gpload		

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 1. Greenplum Database Backend Server Programs

Program Name	Description	Use Instead
`initdb`	This program is called by `gpinitsystem` when initializing a Greenplum Database array. It is used internally to create the individual segment instances and the master instance.	`gpinitsystem`
`ipcclean`	Not used in Greenplum Database	N/A
`pg_basebackup`	This program makes a binary copy of a single database instance. Greenplum Database uses it for tasks such as creating a standby master instance, or recovering a mirror segment when a full copy is needed. Do not use this utility to back up Greenplum Database segment instances because it does not produce MPP-consistent backups.	`gpinitstandby`, `gprecoverseg`
`pg_controldata`	Not used in Greenplum Database	`gpstate`

Table 1. Greenplum Database Backend Server Programs

Program Name	Description	Use Instead
`pg_ctl`	This program is called by `gpstart` and `gpstop` 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.	`gpstart`, `gpstop`
`pg_resetxlog`	DO NOT USE **Warning:** 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.	N/A
`postgres`	The `postgres` executable is the actual PostgreSQL server process that processes queries.	The main `postgres` process (postmaster) creates other `postgres` subprocesses and `postgres` session as needed to handle client connections.
`postmaster`	`postmaster` starts the `postgres` database server listener process that accepts client connections. In Greenplum Database, a `postgres` database listener process runs on the Greenplum master Instance and on each Segment Instance.	In Greenplum Database, you use `gpstart` and `gpstop` to start all postmasters (`postgres` processes) in the system at once in the correct order and with the correct options.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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):

Driver	Serial Number / License Key
DataDirect Connect XE for ODBC 7.1 drivers (32-bit drivers)	1076681728
DataDirect Connect64 XE for ODBC 7.1 drivers (64-bit drivers)	1076681984

Parent topic: DataDirect ODBC Drivers for Pivotal Greenplum

Supported Client Platforms

DataDirect Connect64 XE for ODBC drivers for Greenplum support the following 64-bit client platforms:

- AIX 64: 7.1, 6.1, 5.3 Fixpack 5 or higher
- HP-UX IPF: 11i v3.0 (B.11.3X), 11i v2.0 (B.11.23)
- Linux Itanium: Red Hat Enterprise Linux (RHEL) 7.x, 6.x, RHEL 5.x, RHEL 4.x
- Linux x64: RHEL 7.x RHEL 6.x, RHEL 5.x, RHEL 4.x, SUSE Linux Enterprise Server (SLES) 12, SLES 11, SLES 10, Ubuntu 16.04
- 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)

- Windows x64: Windows 8, Windows 10, Windows Server 20016

DataDirect Connect XE for ODBC drivers for Greenplum support the following 32-bit client platforms:

- AIX 32: 7.1, 6.1, 5.3 Fixpack 5 or higher
- HP-UX IPF: 11i v3.0 (B.11.3X), 11i v2.0 (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, Ubuntu 16.04, Ubuntu 14.04
- Solaris on SPARC: 11 and 11 Express (Solaris 5.11), 10 (Solaris 5.10), 9 (Solaris 5.9), 8 (Solaris 5.8)
- Windows: Windows 8, Windows 10, Windows Server 20016

Parent topic: DataDirect ODBC Drivers for Pivotal Greenplum

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_CONNECT_ODBC_7.1.6.HOTFIX_LINUX_32.tar.Z
    - PROGRESS_DATADIRECT_CONNECT64_ODBC_7.1.6.HOTFIX_AIX_64.tar.Z
    - PROGRESS_DATADIRECT_CONNECT_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
    ```

    The files are extracted to the current directory.
3. Execute the installer:

    ```
    $ 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
    ---------------------------------------------------------------------
    Progress DataDirect Connect (R) and Connect XE for ODBC 7.1 SP5
    for UNIX operating systems
    ---------------------------------------------------------------------

    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          :
    ```

6. Enter the required registration information at each prompt:

Prompt	Enter
Name:	Name to associate with the registration.
Company:	Your company name.
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.

   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/Connect6
4_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 fil
es.


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**

- **Testing the Driver Connection on Linux**

Parent topic: DataDirect ODBC Drivers for Pivotal Greenplum

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.

   ```
   $ vi $ODBCINI
   ```

   You must edit the following entries to add values that match your system:

Entry	Description
Database	Pivotal Greenplum database name.
HostName	Master host name.
PortNumber	Master host port number.
LogonID	Greenplum Database user.
Password	Password.

4. Verify the driver version:

   ```
   $ cd /opt/Progress/DataDirect/Connect64_for_ODBC_71/bin
   $ ./ddtestlib ddgplm27.so
   Load of ddgplm27.so successful, qehandle is 0x19B4C60
   File version: 07.16.0270 (B00357, U0237)
   ```

Parent topic: Installing on Linux Systems

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:

   ```
   $ 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:

   ```
   Enter SQL statements (Press ENTER to QUIT)
   SQL> select version();
   ```

```
version
PostgreSQL 8.3.23 (Greenplum Database 5.0.0 build commit:8c709516061cff5476c03d
6e2da99aae42722ae1) on x86_64-pc-linux-gnu, compiled by GCC gcc (GCC) 6.2.0 com
piled on Sep  1 2017 22:39:53


Enter SQL statements (Press ENTER to QUIT)
SQL>
```

3. Press the ENTER key to exit the example application.

Parent topic: Installing on Linux Systems

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_CONNECT_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**
- **Configuring and Testing the Driver on Windows**

Parent topic: DataDirect ODBC Drivers for Pivotal Greenplum

Verifying the Version on Windows

To verify your driver version:

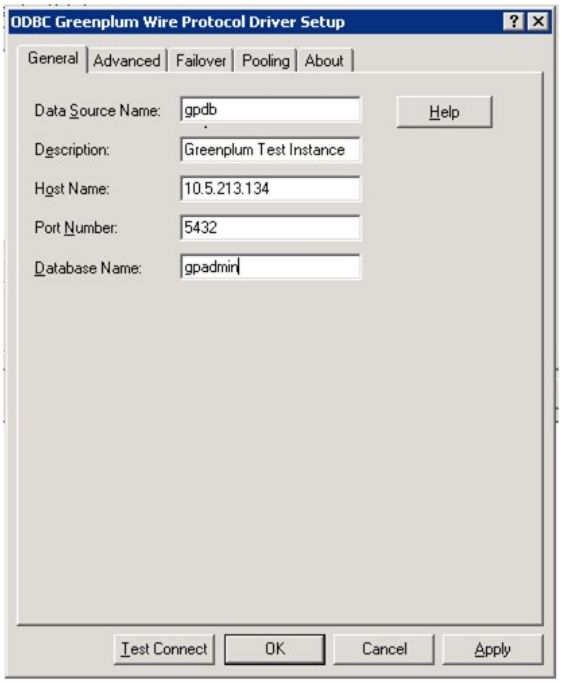
1. Select **Start > All Programs > DataDirect > ODBC Administrator** to open the Windows ODBC Administrator.
2. Click the **Drivers** tab, and scroll down to DataDirect <version> Greenplum Wire Protocol. Ensure that you see the expected version number.

Parent topic: Installing on Windows Systems

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**.
4. Select **DataDirect 7.1 Greenplum Wire Protocol** and click **Finish**.
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

Parent topic: Installing on Windows Systems

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

Parent topic: DataDirect ODBC Drivers for Pivotal Greenplum

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: DataDirect JDBC Driver for Pivotal Greenplum

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.

Parent topic: DataDirect JDBC Driver for Pivotal Greenplum

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:

   ```
   $ cd /opt/Progress/DataDirect/Connect_for_JDBC_51/lib
   ```

2. Execute the data source class to display the version information.

   For Linux/Unix systems:

   ```
   $ java -classpath greenplum.jar com.pivotal.jdbc.GreenplumDriver
   ```

```
[Pivotal][Greenplum JDBC Driver]Driver Version: 5.1.4.000212 (F000427.U000206)
```

For Windows systems:

```
java -classpath .;.\greenplum.jar com.pivotal.jdbc.GreenplumDriver
[Pivotal][Greenplum JDBC Driver]Driver Version: 5.1.4.000212 (F000427.U000206)
```

Parent topic: DataDirect JDBC Driver for Pivotal Greenplum

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.

Property	Description
Driver File Name	greenplum.jar
Data Source Class	com.pivotal.jdbc.GreenplumDriver
Connection URL	jdbc:pivotal:greenplum://host:port;DatabaseName=<name>
Driver Defaults	FetchTWFSasTime=true MaxLongVarcharSize=8190 MaxNumericPrecision=28 MaxNumericScale=6 PrepareThreshold=0 ResultSetMetadataOptions=1 SupportsCatalogs=true

Parent topic: DataDirect JDBC Driver for Pivotal Greenplum

Configuring Prepared Statement Execution

The DataDirect JDBC driver version 5.1.4.000270 (F000450.U000214) introduced support for the `PrepareThreshold` connection property. This property specifies the number of prepared statement executions to be performed before the driver switches to using server-side prepared statements.

The `PrepareThreshold` default value is `0`, always use server-side prepare for prepared statements. This setting preserves the behavior of previous versions of the JDBC driver.

When the `PrepareThreshold` value is greater than 1, it specifies on which execution of a prepared statement the driver starts using server-side prepared statements.

Note: `statement.executeBatch()` always uses server-side prepare for prepared statements. This matches the behavior of the PostgreSQL open source JDBC driver.

Refer to PrepareThreshold in the DataDirect documentation for additional information about this connection property.

Limitation

When the `PrepareThreshold` value is greater than one and the prepared statement includes parameterized operations, the driver does not send any SQL prepare calls during `connection.prepareStatement()`. The driver instead sends the query all at once, at execution time. This requires that the driver determine the data types of every column *before* it sends the query to the server. While the driver can make this determination for many data types, it cannot for the JDBC types that can be mapped to multiple Greenplum data types:

- BIT VARYING
- BOOLEAN

- JSON
- TIME WITH TIME ZONE
- UUIDCOL

To work around this limitation, set `PrepareThreshold` to `0` when a prepared statement uses parameterized values with any of the above data types. And use `ResultSet.getMetaData()` to determine if any of the above types are used in a query in advance of submitting the prepared statement.

Note: GPORCA does not support prepared statements that have parameterized values, and will fall back to using the Postgres Planner.

Parent topic: DataDirect JDBC Driver for Pivotal Greenplum

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

Parent topic: DataDirect JDBC Driver for Pivotal Greenplum

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

SQL Command Reference

The following SQL commands are available in Greenplum Database:

- ALTER DATABASE
- ALTER DOMAIN
- ALTER EXTENSION
- ALTER EXTERNAL TABLE
- ALTER FILESPACE
- 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 TABLE
- ALTER TABLESPACE
- ALTER TYPE
- ALTER USER
- ALTER VIEW
- ANALYZE
- BEGIN
- CHECKPOINT
- CLOSE
- CLUSTER
- COMMENT
- COMMIT
- COPY
- CREATE AGGREGATE
- CREATE CAST
- CREATE CONVERSION
- CREATE DATABASE
- CREATE DOMAIN
- CREATE EXTENSION
- CREATE EXTERNAL 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 TABLE
- CREATE TABLE AS
- CREATE TABLESPACE
- CREATE TYPE
- CREATE USER
- CREATE VIEW
- DEALLOCATE
- DECLARE
- DELETE
- DISCARD
- DO
- DROP AGGREGATE
- DROP CAST
- DROP CONVERSION
- DROP DATABASE
- DROP DOMAIN
- DROP EXTENSION
- DROP EXTERNAL TABLE
- DROP FILESPACE
- 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 TABLE
- DROP TABLESPACE
- DROP TYPE
- DROP USER
- 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

Parent topic: Greenplum Database Reference Guide

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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]

ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
   RENAME TO new_name

ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
   OWNER TO new_owner

ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
   SET SCHEMA new_schema
```

See ALTER FUNCTION for more information.

ALTER GROUP

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 RENAME 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

where action is one of:

  ADD [COLUMN] column_name data_type [ DEFAULT default_expr ]
      [column_constraint [ ... ]]
      [ ENCODING ( storage_directive [,...] ) ]
  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
```

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] [table [ (column [, ...] ) ]]

ANALYZE [VERBOSE] {root_partition|leaf_partition} [ (column [, ...] )]

ANALYZE [VERBOSE] ROOTPARTITION {ALL | root_partition [ (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

CLUSTER
```

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 |
  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'
```

See COMMENT for more information.

COMMIT

Commits the current transaction.

```
COMMIT [WORK | TRANSACTION]
```

See COMMIT for more information.

COPY

Copies data between a file and a table.

```
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.

```
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.

```
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.

```
CREATE [DEFAULT] CONVERSION name FOR source_encoding TO
     dest_encoding FROM funcname
```

See CREATE CONVERSION for more information.

CREATE DATABASE

Creates a new database.

```
CREATE DATABASE name [ [WITH] [OWNER [=] dbowner]
                     [TEMPLATE [=] template]
                     [ENCODING [=] encoding]
                     [TABLESPACE [=] tablespace]
                     [CONNECTION LIMIT [=] connlimit ] ]
```

See CREATE DATABASE for more information.

CREATE DOMAIN

Defines a new domain.

```
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 [TEMPORARY | TEMP] TABLE table_name
    ( column_name data_type [, ...] | LIKE other_table )
     LOCATION ('file://seghost[:port]/path/file' [, ...])
       | ('gpfdist://filehost[:port]/file_pattern[#transform=trans_name]'
           [, ...]
       | ('gpfdists://filehost[:port]/file_pattern[#transform=trans_name]'
           [, ...])
       | ('gphdfs://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 [PERSISTENTLY]] SEGMENT REJECT LIMIT count
      [ROWS | PERCENT] ]

CREATE [READABLE] EXTERNAL WEB [TEMPORARY | TEMP] 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] )]
          | 'CUSTOM' (Formatter=<formatter specifications>)
     [ ENCODING 'encoding' ]
     [ [LOG ERRORS [PERSISTENTLY]] SEGMENT REJECT LIMIT count
       [ROWS | PERCENT] ]

CREATE WRITABLE EXTERNAL [TEMPORARY | TEMP] TABLE table_name
    ( column_name data_type [, ...] | LIKE other_table )
     LOCATION('gpfdist://outputhost[:port]/filename[#transform=trans_name]'
          [, ...])
      | ('gpfdists://outputhost[:port]/file_pattern[#transform=trans_name]'
          [, ...])
      | ('gphdfs://hdfs_host[:port]/path')
      FORMAT 'TEXT'
               [( [DELIMITER [AS] 'delimiter']
               [NULL [AS] 'null string']
               [ESCAPE [AS] 'escape' | 'OFF'] )]
          | 'CSV'
               [([QUOTE [AS] 'quote']
               [DELIMITER [AS] 'delimiter']
               [NULL [AS] 'null string']
               [FORCE QUOTE column [, ...]] | * ]
               [ESCAPE [AS] 'escape'] )]
           | 'AVRO'
           | 'PARQUET'

           | 'CUSTOM' (Formatter=<formatter specifications>)
    [ ENCODING 'write_encoding' ]
    [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]

CREATE WRITABLE EXTERNAL [TEMPORARY | TEMP] TABLE table_name
    ( column_name data_type [, ...] | LIKE other_table )
     LOCATION('s3://S3_endpoint[:port]/bucket_name/[S3_prefix] [region=S3-region] [con
fig=config_file]')
      [ON MASTER]
      FORMAT 'TEXT'
               [( [DELIMITER [AS] 'delimiter']
               [NULL [AS] 'null string']
               [ESCAPE [AS] 'escape' | 'OFF'] )]
```

```
          | 'CSV'
               [([QUOTE [AS] 'quote']
               [DELIMITER [AS] 'delimiter']
               [NULL [AS] 'null string']
               [FORCE QUOTE column [, ...]] | * ]
               [ESCAPE [AS] 'escape'] )]

CREATE WRITABLE EXTERNAL WEB [TEMPORARY | TEMP] TABLE table_name
    ( column_name data_type [, ...] | LIKE other_table )
    EXECUTE 'command' [ON ALL]
    FORMAT 'TEXT'
               [( [DELIMITER [AS] 'delimiter']
               [NULL [AS] 'null string']
               [ESCAPE [AS] 'escape' | 'OFF'] )]
          | 'CSV'
               [([QUOTE [AS] 'quote']
               [DELIMITER [AS] 'delimiter']
               [NULL [AS] 'null string']
               [FORCE QUOTE column [, ...]] | * ]
               [ESCAPE [AS] 'escape'] )]
           | 'CUSTOM' (Formatter=<formatter specifications>)
    [ ENCODING 'write_encoding' ]
    [ DISTRIBUTED BY (column, [ ... ] ) | 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 ] [, ...] ] )
      [ 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 DEFINER
    | COST execution_cost
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition'
    | AS 'obj_file', 'link_symbol' } ...
    [ WITH ({ DESCRIBE = describe_function
           } [, ...] ) ]
```

See CREATE FUNCTION for more information.

CREATE GROUP

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.

```
CREATE RESOURCE GROUP name WITH (group_attribute=value [, ... ])
```

See CREATE RESOURCE GROUP for more information.

CREATE RESOURCE QUEUE

Defines a new resource queue.

```
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).

```
CREATE ROLE name [[WITH] option [ ... ]]
```

See CREATE ROLE for more information.

CREATE RULE

Defines a new rewrite rule.

```
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.

```
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.

```
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]
   AS query
   [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 (
    INPUT = input_function,
    OUTPUT = output_function
    [, RECEIVE = receive_function]
    [, SEND = send_function]
    [, TYPMOD_IN = type_modifier_input_function ]
    [, TYPMOD_OUT = type_modifier_output_function ]
    [, INTERNALLENGTH = {internallength | VARIABLE}]
    [, PASSEDBYVALUE]
    [, ALIGNMENT = alignment]
    [, STORAGE = storage]
    [, DEFAULT = default]
    [, ELEMENT = element]
    [, DELIMITER = delimiter] )

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.

```
DROP [PROCEDURAL] LANGUAGE [IF EXISTS] name [CASCADE | RESTRICT]
```

See DROP LANGUAGE for more information.

DROP OPERATOR

Removes an operator.

```
DROP OPERATOR [IF EXISTS] name ( {lefttype | NONE} ,
    {righttype | NONE} ) [CASCADE | RESTRICT]
```

See DROP OPERATOR for more information.

DROP OPERATOR CLASS

Removes an operator class.

```
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.

```
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.

```
DROP OWNED BY name [, ...] [CASCADE | RESTRICT]
```

See DROP OWNED for more information.

DROP PROTOCOL

Removes a external table data access protocol from a database.

```
DROP PROTOCOL [IF EXISTS] name
```

See DROP PROTOCOL for more information.

DROP RESOURCE GROUP

Removes a resource group.

```
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] tablespacename
```

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 { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN DATA WRAPPER fdwname [, ...]
    TO { [ GROUP ] rolename | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN SERVER servername [, ...]
    TO { [ GROUP ] 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.

```
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.

```
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.

```
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.

```
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.

```
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.

```
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.

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`:

```
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`:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

ALTER DATABASE

Changes the attributes of a database.

Synopsis

```
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.

connlimit
: 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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

ALTER DOMAIN name SET SCHEMA new_schema
```

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

ALTER EXTENSION

Change the definition of an extension that is registered in a Greenplum database.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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` or `ALTER 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::in
teger
```

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

ALTER FILESPACE

Changes the definition of a filespace.

Synopsis

```
ALTER FILESPACE name RENAME TO newname

ALTER FILESPACE name OWNER TO newowner
```

Description

`ALTER FILESPACE` changes the definition of a filespace.

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*

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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
RESET configuration_parameter
RESET ALL
```

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

ALTER GROUP

Changes a role name or membership.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

ALTER OPERATOR FAMILY

Changes the definition of an operator family.

Synopsis

```
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 `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.

newowner
: 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 `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

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 by 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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

ALTER RESOURCE GROUP

Changes the limits of a resource group.

Synopsis

```
ALTER RESOURCE GROUP name SET group_attribute value
```

where *group_attribute* is one of:

```
CONCURRENCY integer
CPU_RATE_LIMIT integer
CPUSET tuple
MEMORY_LIMIT integer
MEMORY_SHARED_QUOTA integer
MEMORY_SPILL_RATIO integer
```

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 that you create for roles to control the maximum number of active concurrent statements in that group. You can also reset the memory or CPU resources 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.

When you alter the CPU resource management mode or limit of a resource group, the new mode or limit is immediately applied.

When you alter a memory limit of a resource group that you create for roles, 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 the current resource usage exceeds the new memory limit value, or if there are running transactions in other resource groups that hold some of the resource, then Greenplum Database defers assigning the new limit until resource usage falls within the range of the new value.

When you increase the memory limit of a resource group that you create for external components, the new resource limit is phased in as system memory resources become available. If you decrease the memory limit of a resource group that you create for external components, the behavior is component-specific. For example, if you decrease the memory limit of a resource group that you create for a PL/Container runtime, queries in a running container may fail with an out of memory error.

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 resource groups that you assign to roles. 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 for a resource group that you create for roles 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`s of all resource groups defined in the Greenplum Database cluster must not exceed 100.

  If you alter the `CPU_RATE_LIMIT` of a resource group in which you previously configured a `CPUSET`, `CPUSET` is disabled, the reserved CPU cores are returned to Greenplum Database, and `CPUSET` is set to -1.

CPUSET *tuple*
: The CPU cores to reserve for this resource group. The CPU cores that you specify in *tuple* must be available in the system and cannot overlap with any CPU cores that you specify for other resource groups.

  tuple is a comma-separated list of single core numbers or core intervals. You must enclose *tuple* in single quotes, for example, '1,3-4'.

  If you alter the `CPUSET` value of a resource group for which you previously configured a `CPU_RATE_LIMIT`, `CPU_RATE_LIMIT` is disabled, the reserved CPU resources are returned to Greenplum Database, and `CPU_RATE_LIMIT` is set to -1.

You can alter `CPUSET` for a resource group only after you have enabled resource group-based resource management for your Greenplum Database cluster.

MEMORY_LIMIT *integer*
: The percentage of Greenplum Database memory resources to reserve for this resource group. The minimum memory percentage for a resource group is 0. The maximum is 100. When `MEMORY_LIMIT` is 0, Greenplum Database reserves no memory for the resource group, but uses global shared memory to fulfill all memory requests in the group. If `MEMORY_LIMIT` is 0, `MEMORY_SPILL_RATIO` must also be 0.

The sum of the `MEMORY_LIMIT`s of all resource groups defined in the Greenplum Database cluster must not exceed 100. If this sum is less than 100, Greenplum Database allocates any unreserved memory to a resource group global shared memory pool.

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. You can specify an integer percentage value from 0 to 100 inclusive. The default `MEMORY_SPILL_RATIO` value is 20. When `MEMORY_SPILL_RATIO` is 0, Greenplum Database uses the `statement_mem` server configuration parameter value to control initial query operator memory.

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;
```

Update the memory spill ratio for a resource group:

```
ALTER RESOURCE GROUP rgroup4 SET MEMORY_SPILL_RATIO 25;
```

Reserve CPU core 1 for a resource group:

```
ALTER RESOURCE GROUP rgroup5 SET CPUSET '1';
```

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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_attributes` 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

name
: The name of the resource queue whose limits are to be altered.

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`.

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`).

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`.

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.

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`.

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

GPORCA and the legacy Greenplum Database query optimizer utilize different query costing models and may compute different costs for the same query. The Greenplum Database resource queue resource management scheme neither differentiates nor aligns costs between GPORCA and the legacy optimizer; it uses the literal cost value returned from the optimizer to throttle queries.

When resource queue-based resource management is active, use the `MEMORY_LIMIT` and `ACTIVE_STATEMENTS` limits for resource queues rather than configuring cost-based limits. Even when using GPORCA, Greenplum Database may fall back to using the legacy query optimizer for certain queries, so using cost-based limits can lead to unexpected results.

Examples

Change the active query limit for a resource queue:

```
ALTER RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=20);
```

Change the memory limit for a resource queue:

```
ALTER RESOURCE QUEUE myqueue WITH (MEMORY_LIMIT='2GB');
```

Reset the maximum and minimum query cost limit for a resource queue to no limit:

```
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:

```
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:

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

Remove the `MAX_COST` and `MEMORY_LIMIT` limits from a resource queue:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

ALTER ROLE

Changes a database role (user or group).

Synopsis

```
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
    | 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'
    | [ DENY deny_point ]
    | [ DENY BETWEEN deny_point AND deny_point]
    | [ DROP DENY FOR deny_point ]
```

Description

`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. You cannot assign a resource group that you create for an external component to a role. See `CREATE RESOURCE GROUP` for additional information.
- **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 `RESET ALL` to clear all role-specific settings. See `SET` and Server Configuration Parameters for information about user-settable configuration parameters.

group_name
: The name of the resource group to assign to this role. Specifying the *group_name* `NONE` removes the role's current resource group assignment and assigns a default resource group based on the role's capability. `SUPERUSER` roles are assigned the `admin_group` resource group, while the `default_group` resource group is assigned to non-admin roles.
  You cannot assign a resource group that you create for an external component to a role.

queue_name
: The name of the resource queue to which the user-level role is to be assigned. Only roles with `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 `NONE`. A role can only belong to one resource queue.

SUPERUSER | NOSUPERUSER
CREATEDB | NOCREATEDB
CREATEROLE | NOCREATEROLE
CREATEUSER | NOCREATEUSER

`CREATEUSER` and `NOCREATEUSER` are obsolete, but still accepted, spellings of `SUPERUSER` and `NOSUPERUSER`. Note that they are not equivalent to the `CREATEROLE and NOCREATEROLE` clauses.

CREATEEXTTABLE | NOCREATEEXTTABLE [(attribute='value')]

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
LOGIN | NOLOGIN
CONNECTION LIMIT connlimit
PASSWORD password
ENCRYPTED | UNENCRYPTED
VALID UNTIL 'timestamp'

These clauses alter role attributes originally set by `CREATE ROLE`.

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 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 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:

```
ALTER ROLE daria WITH PASSWORD 'passwd123';
```

Change a password expiration date:

```
ALTER ROLE scott VALID UNTIL 'May 4 12:00:00 2015 +1';
```

Make a password valid forever:

```
ALTER ROLE luke VALID UNTIL 'infinity';
```

Give a role the ability to create other roles and new databases:

```
ALTER ROLE joelle CREATEROLE CREATEDB;
```

Give a role a non-default setting of the `maintenance_work_mem` parameter:

```
ALTER ROLE admin SET maintenance_work_mem = 100000;
```

Assign a role to a resource queue:

```
ALTER ROLE sammy RESOURCE QUEUE poweruser;
```

Give a role permission to create writable external tables:

```
ALTER ROLE load CREATEEXTTABLE (type='writable');
```

Alter a role so it does not allow login access on Sundays:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

ALTER SEQUENCE

Changes the definition of a sequence generator.

Synopsis

```
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
```

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*

OWNED BY NONE

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

ALTER TABLE

Changes the definition of a table.

Synopsis

```
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

where action is one of:

  ADD [COLUMN] column_name data_type [ DEFAULT default_expr ]
```

```
    [column_constraint [ ... ]]
    [ ENCODING ( storage_directive [,...] ) ]
  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:

```
  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`. The `ENCODING` clause is valid only for append-optimized, column-oriented tables.
- **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 1 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 `NOTNULL` 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 ×100) × (2 ×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.

new_name
: New name for the table.

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, use `ALTER PARTITION` clauses to specify which subpartition in the hierarchy you want to alter. For each partition level in the table hierarchy that is above the target partition, specify the partition that is related to the target partition in an `ALTER PARTITION` clause.

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 if the partitioned table contains a column with a check constraint or a `NOT NULL` constraint.

Exchanging a partition with a partitioned table or a child partition of a partitioned table is not supported.

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. In a multi-level partition, only a range partition can be split, not a list partition, and 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. In a multi-level partition, only a range partition can be split, not a list partition, and 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.

partition_name

The given name of a partition. The given partition name is the `partitionname` column value in the *pg_partitions* system view.

FOR (RANK(number))

For range partitions, the rank of the partition in the range.

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. 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` (`ALTER TABLE ONLY ... DROP COLUMN`) 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:

```
ALTER TABLE distributors RENAME COLUMN address TO city;
```

Rename an existing table:

```
ALTER TABLE distributors RENAME TO suppliers;
```

Add a not-null constraint to a column:

```
ALTER TABLE distributors ALTER COLUMN street SET NOT NULL;
```

Add a check constraint to a table:

```
ALTER TABLE distributors ADD CONSTRAINT zipchk CHECK
(char_length(zipcode) = 5);
```

Move a table to a different schema:

```
ALTER TABLE myschema.distributors SET SCHEMA yourschema;
```

Add a new partition to a partitioned table:

```
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:

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

Rename a partition:

```
ALTER TABLE sales RENAME PARTITION FOR ('2016-01-01') TO
jan08;
```

Drop the first (oldest) partition in a range sequence:

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

Exchange a table into your partition design:

```
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:

```
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:

```
ALTER TABLE sales SPLIT PARTITION FOR ('2016-01-01')
AT ('2016-01-16')
INTO (PARTITION jan081to15, PARTITION jan0816to31);
```

For a multi-level partitioned table that consists of three levels, year, quarter, and region, exchange a leaf partition `region` with the table `region_new`.

```
ALTER TABLE sales ALTER PARTITION year_1 ALTER PARTITION quarter_4 EXCHANGE PARTITION
region WITH TABLE region_new ;
```

In the previous command, the two `ALTER PARTITION` clauses identify which `region` partition to exchange. Both clauses are required to identify the specific partition to exchange.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

See Also

`CREATE TABLESPACE`, `DROP TABLESPACE`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

ALTER USER

Changes the definition of a database role (user).

Synopsis

```
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 connlimit
    | [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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

ALTER VIEW

Changes the definition of a view.

Synopsis

```
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*

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

ANALYZE

Collects statistics about a database.

Synopsis

```
ANALYZE [VERBOSE] [table [ (column [, ...] ) ]]

ANALYZE [VERBOSE] {root_partition|leaf_partition} [ (column [, ...] )]
```

```
ANALYZE [VERBOSE] ROOTPARTITION {ALL | root_partition [ (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. For information about the table statistics that are collected, see Notes.

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.

For partitioned tables, ANALYZE collects additional statistics, HyperLogLog (HLL) statistics, on the leaf child partitions. HLL statistics are used are used to derive number of distinct values (NDV) for queries against partitioned tables.

- When aggregating NDV estimates across multiple leaf child partitions, HLL statistics generate a more accurate NDV estimates than the standard table statistics.
- When updating HLL statistics, ANALYZE operations are required only on leaf child partitions that have changed. For example, ANALYZE is required if the leaf child partition data has changed, or if the leaf child partition has been exchanged with another table. For more information about updating partitioned table statistics, see Notes.

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 or ANALYZE ROOTPARTITION command. For information about collecting statistics on partitioned tables and when the ROOTPARTITION keyword is required, see Notes. For information about GPORCA, see Overview of GPORCA.

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

{ *root_partition* | *leaf_partition* } [(*column* [, ...])]

Collect statistics for partitioned tables including HLL statistics. HLL statistics are collected only on leaf child partitions.

ANALYZE *root_partition*, collects statistics on all leaf child partitions and the root partition.

ANALYZE *leaf_partition*, collects statistics on the leaf child partition.

By default, if you specify a leaf child partition, and all other leaf child partitions have statistics, ANALYZE updates the root partition statistics. If not all leaf child partitions have statistics, ANALYZE logs information about the leaf child partitions that do not have statistics. For information about when root partition statistics are collected, see Notes.

ROOTPARTITION [ALL]

Collect statistics only on the root partition of partitioned tables based on the data in the partitioned table. If possible, ANALYZE uses leaf child partition statistics to generate root partition statistics. Otherwise, ANALYZE collects the statistics by sampling leaf child partition data. Statistics are not collected on the leaf child partitions, the data is only sampled. HLL statistics are not collected.

For information about when the ROOTPARTITION keyword is required, see Notes.

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 collected.

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. 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;
```

`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`.

If you run `ANALYZE` on a table that does not contain data, statistics are not collected for the table. For example, if you perform a `TRUNCATE` operation on a table that has statistics, and then run `ANALYZE` on the table, the statistics do not change.

For a partitioned table, specifying which portion of the table to analyze, the root partition or subpartitions (leaf child partition tables) can be useful if the partitioned table has a large number of partitions that have been analyzed and only a few leaf child partitions have changed.
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).

- When you run `ANALYZE` on the root partitioned table, statistics are collected for all the leaf

child partitions. Leaf child partitions are 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 partition, statistics are collected only for that leaf child partition and the root partition. If data in the leaf partition has changed (for example, you made significant updates to the leaf child partition data or you exchanged the leaf child partition), then you can run ANALYZE on the leaf child partition to collect table statistics. By default, if all other leaf child partitions have statistics, the command updates the root partition statistics.

  For example, if you collected statistics on a partitioned table with a large number partitions and then updated data in only a few leaf child partitions, you can run `ANALYZE` only on those partitions to update statistics on the partitions and the statistics on the root partition.

- When you run `ANALYZE` on a child table that is not a leaf child partition, 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 partition for March of 2013, statistics are collected only for that leaf child partition.

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` is run on an external table partition, the partition is not analyzed.
- If `ANALYZE` or `ANALYZE ROOTPARTITION` is run on the root partition, external table partitions are not sampled and root table statistics do not include external table partition.
- 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 `on` (the default), the `ROOTPARTITION` keyword is not required to collect statistics on the root partition when you run `ANALYZE`. Root partition statistics are collected when you run `ANALYZE` on the root partition, or when you run `ANALYZE` on a child leaf partition of the partitioned table and the other child leaf partitions have statistics. If the parameter is `off`, you must run `ANALZYE ROOTPARTITION` to collect root partition statistics.

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 100, 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 `ANALYZE`.

When Greenplum Database performs an `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 `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 `VACUUM FULL` operation removes the empty pages and allows an `ANALYZE` operation to collect accurate statistics.

If there are no statistics for the table, the server configuration parameter `gp_enable_relsize_collection` controls whether the legacy query optimizer uses a default statistics file or estimates the size of a table using the `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 `mytable`:

```
ANALYZE mytable;
```

Compatibility

There is no `ANALYZE` statement in the SQL standard.

See Also

`ALTER TABLE`, `EXPLAIN`, `VACUUM`, `analyzedb` utility in the *Greenplum Database Utility Guide*.

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

BEGIN

Starts a transaction block.

Synopsis

```
BEGIN [WORK | TRANSACTION] [transaction_mode]
      [READ ONLY | READ WRITE]
```

where *transaction_mode* is one of:

```
ISOLATION LEVEL | {SERIALIZABLE | READ COMMITTED | READ UNCOMMITTED}
```

Description

`BEGIN` initiates a transaction block, that is, all statements after a `BEGIN` command will be executed in a single transaction until an explicit `COMMIT` or `ROLLBACK` is given. By default (without `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 `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`.

Incidentally, 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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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-optmized 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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`:

```
COMMENT ON TABLE mytable IS 'This is my table.';
```

Remove it again:

```
COMMENT ON TABLE mytable IS NULL;
```

Compatibility

There is no `COMMENT` statement in the SQL standard.

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

COMMIT

Commits the current transaction.

Synopsis

```
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

WORK
TRANSACTION
: 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:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

COPY

Copies data between a file and a table.

Synopsis

```
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 ]
```

Description

`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;
```

<SEG_DATA_DIR>

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.

<SEGID>

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.

BINARY

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.

OIDS

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. SeeCSV 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 ...`

`COPY` only deals with the specific table named; it does not copy data to or from child tables. Thus for example `COPY *table* TO` shows the same data as `SELECT * FROM ONLY table`. But `COPY (SELECT * FROM *table*) TO ...` can be used to dump all of the data in an inheritance hierarchy.

Similarly, to copy data from a partitioned table with a leaf child partition that is an external table, use an SQL query to select the data to copy. For example, if the table `my_sales` contains 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` keyword 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`. It is sufficient to have column privileges on the columns listed in the command.

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. Only database superusers are permitted to name files with `COPY`, because this 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`. It is also a good idea to avoid dumping data with *IntervalStyle* set to `sql_standard`, because negative interval values might be misinterpreted by a server that has a different setting for *IntervalStyle*.

Input data is interpreted according to `ENCODING` option or the current client encoding, and output data is encoded in `ENCODING` or the current client encoding, even if the data does not pass through the client but is read from or written to a file directly by the server.

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 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:

  ```
  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 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 = \

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 = \
```

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\n\0 — 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:

```
COPY country TO STDOUT WITH DELIMITER '|';
```

Copy data from a file into the `country` table:

```
COPY country FROM '/home/usr1/sql/country_data';
```

Copy into a file just the countries whose names start with 'A':

```
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:

```
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:

```
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:

```
contentid | dbid | file segment location
    0     |  1   | /home/usr1/data1/gpsegdir0
    0     |  3   | /home/usr1/data_mirror1/gpsegdir0
    1     |  4   | /home/usr1/data2/gpsegdir1
    1     |  2   | /home/usr1/data_mirror2/gpsegdir1
```

running the command:

```
COPY mytable TO '<SEG_DATA_DIR>/gpbackup<SEGID>.txt' ON SEGMENT;
```

would result in the following files:

```
/home/usr1/data1/gpsegdir0/gpbackup0.txt
/home/usr1/data2/gpsegdir1/gpbackup1.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 uses a `SELECT` statement to copy data to files on each segment:

```
COPY (SELECT * FROM testtbl) TO '/tmp/mytst<SEGID>' 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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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]
        [, SORTOP = sort_operator] )
```

Description

`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:

```
sfunc( internal-state, next-data-values ) ---> next-internal-state
prefunc( internal-state, internal-state ) ---> next-internal-state
ffunc( internal-state ) ---> 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:

```
SELECT agg(col) FROM tab;
```

must be equivalent to:

```
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.

```
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.

```
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.

```
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.

```
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.

```
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.

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

CREATE CAST

Defines a new cast.

Synopsis

```
CREATE CAST (sourcetype AS targettype)
       WITH FUNCTION funcname (argtypes)
       [AS ASSIGNMENT | AS IMPLICIT]
```

```
CREATE CAST (sourcetype AS targettype) WITHOUT FUNCTION
       [AS ASSIGNMENT | AS IMPLICIT]
```

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

sourcetype
: The name of the source data type of the cast.

targettype
: The name of the target data type of the cast.

funcname(argtypes)
: 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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

connlimit
: The maximum number of concurrent connections posible. 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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

CREATE DOMAIN

Defines a new domain.

Synopsis

```
CREATE DOMAIN name [AS] data_type [DEFAULT expression]
       [CONSTRAINT constraint_name
       | NOT NULL | NULL
```

```
    | CHECK (expression) [...]]
```

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

CREATE EXTERNAL TABLE

Defines a new external table.

Synopsis

```
CREATE [READABLE] EXTERNAL [TEMPORARY | TEMP] TABLE table_name
    ( column_name data_type [, ...] | LIKE other_table )
     LOCATION ('file://seghost[:port]/path/file' [, ...])
       | ('gpfdist://filehost[:port]/file_pattern[#transform=trans_name]'
           [, ...]
```

```
       | ('gpfdists://filehost[:port]/file_pattern[#transform=trans_name]'
           [, ...])
       | ('gphdfs://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 [PERSISTENTLY]] SEGMENT REJECT LIMIT count
      [ROWS | PERCENT] ]

CREATE [READABLE] EXTERNAL WEB [TEMPORARY | TEMP] 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] )]
          | 'CUSTOM' (Formatter=<formatter specifications>)
     [ ENCODING 'encoding' ]
     [ [LOG ERRORS [PERSISTENTLY]] SEGMENT REJECT LIMIT count
       [ROWS | PERCENT] ]

CREATE WRITABLE EXTERNAL [TEMPORARY | TEMP] TABLE table_name
    ( column_name data_type [, ...] | LIKE other_table )
     LOCATION('gpfdist://outputhost[:port]/filename[#transform=trans_name]'
          [, ...])
      | ('gpfdists://outputhost[:port]/file_pattern[#transform=trans_name]'
          [, ...])
      | ('gphdfs://hdfs_host[:port]/path')
      FORMAT 'TEXT'
              [( [DELIMITER [AS] 'delimiter']
```

```
                [NULL [AS] 'null string']
                [ESCAPE [AS] 'escape' | 'OFF'] )]
          | 'CSV'
                [([QUOTE [AS] 'quote']
                [DELIMITER [AS] 'delimiter']
                [NULL [AS] 'null string']
                [FORCE QUOTE column [, ...]] | * ]
                [ESCAPE [AS] 'escape'] )]
           | 'AVRO'
           | 'PARQUET'

           | 'CUSTOM' (Formatter=<formatter specifications>)
    [ ENCODING 'write_encoding' ]
    [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]

CREATE WRITABLE EXTERNAL [TEMPORARY | TEMP] TABLE table_name
    ( column_name data_type [, ...] | LIKE other_table )
     LOCATION('s3://S3_endpoint[:port]/bucket_name/[S3_prefix] [region=S3-region] [con
fig=config_file]')
      [ON MASTER]
      FORMAT 'TEXT'
                [( [DELIMITER [AS] 'delimiter']
                [NULL [AS] 'null string']
                [ESCAPE [AS] 'escape' | 'OFF'] )]
          | 'CSV'
                [([QUOTE [AS] 'quote']
                [DELIMITER [AS] 'delimiter']
                [NULL [AS] 'null string']
                [FORCE QUOTE column [, ...]] | * ]
                [ESCAPE [AS] 'escape'] )]

CREATE WRITABLE EXTERNAL WEB [TEMPORARY | TEMP] TABLE table_name
    ( column_name data_type [, ...] | LIKE other_table )
    EXECUTE 'command' [ON ALL]
    FORMAT 'TEXT'
                [( [DELIMITER [AS] 'delimiter']
                [NULL [AS] 'null string']
                [ESCAPE [AS] 'escape' | 'OFF'] )]
          | 'CSV'
                [([QUOTE [AS] 'quote']
                [DELIMITER [AS] 'delimiter']
                [NULL [AS] 'null string']
                [FORCE QUOTE column [, ...]] | * ]
                [ESCAPE [AS] 'escape'] )]
           | 'CUSTOM' (Formatter=<formatter specifications>)
    [ ENCODING 'write_encoding' ]
    [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]
```

Description

`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.

See Working with External Data for detailed information about working with external tables.

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.

TEMPORARY | TEMP

If specified, creates a temporary readable or writable external table definition in Greenplum Database. Temporary external tables exist in a special schema; you cannot specify a schema name when you create the table. Temporary external tables are automatically dropped at the end of a session.

An existing permanent table with the same name is not visible to the current session while the temporary table exists, unless you reference the permanent table with its schema-qualified name.

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 (deprecated) to read or write a file to a Hadoop file system (HDFS), refer to Specify gphdfs Protocol in an External Table Definition (Deprecated) 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 PXF Creating an External Table 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 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 (deprecated). 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` (deprecated), `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 ON ALL.

FORMAT 'TEXT | CSV | AVRO | PARQUET' *(options)*

When the FORMAT clause identfies 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 (deprecated). Refer to Accessing HDFS Data with gphdfs (Deprecated) 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 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 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.

```
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. If `*` is specified then non-`NULL` values will be quoted in all columns. `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 [PERSISTENTLY]

This is an optional clause that can precede a `SEGMENT REJECT LIMIT` clause to log information about rows with formatting errors. The error log data is stored internally. If error log data exists for a specified external table, new data is appended to existing error log data. The error log data is not replicated to mirror segments.

The data is deleted when the external table is dropped unless you specify the keyword `PERSISTENTLY`. If the keyword is specified, the log data persists after the external table is dropped.

The error log data is accessed with the Greenplum Database built-in SQL function `gp_read_error_log()`, or with the SQL function `gp_read_persistent_error_log()` if the `PERSISTENTLY` keyword is specified.

If you use the `PERSISTENTLY` keyword, you must install the functions that manage the persistent error log information.

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:

```
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:

```
CREATE EXTERNAL TABLE ext_expenses (name text, date date,
amount float4, category text, description text)
LOCATION (
'file://seghost1/dbfast/external/expenses1.csv',
'file://seghost1/dbfast/external/expenses2.csv',
'file://seghost2/dbfast/external/expenses3.csv',
'file://seghost2/dbfast/external/expenses4.csv',
'file://seghost3/dbfast/external/expenses5.csv',
'file://seghost3/dbfast/external/expenses6.csv'
)
FORMAT 'CSV' ( HEADER );
```

Create a readable external web table that executes a script once per segment host:

```
CREATE EXTERNAL WEB TABLE log_output (linenum int, message
text)  EXECUTE '/var/load_scripts/get_log_data.sh' ON HOST
 FORMAT 'TEXT' (DELIMITER '|');
```

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`.

```
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`:

```
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:

```
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. For information about the error log format, see Viewing Bad Rows in the Error Log.

You can view and manage the captured error log data. The functions to manage log data depend on whether the data is persistent (the `PERSISTENTLY` keyword is used with the `LOG ERRORS` clause).

- Functions that manage non-persistent error log data from external tables that were defined without the `PERSISTENTLY` keyword.
    - The built-in SQL function `gp_read_error_log('table_name')` displays error log information for an external table. This example displays the error log data from the external table `ext_expenses`.

      ```
      SELECT * from gp_read_error_log('ext_expenses');
      ```

      The function returns no data if you created the external table with the `LOG ERRORS PERSISTENTLY` clause, or if the external table does not exist.
    - The built-in SQL function `gp_truncate_error_log('table_name')` deletes the error log data for *table_name*. This example deletes the error log data captured from the external table `ext_expenses`:

      ```
      SELECT gp_truncate_error_log('ext_expenses');
      ```

      Dropping the table also deletes the table's log data. The function does not truncate log data if the external table is defined with the `LOG ERRORS PERSISTENTLY` clause.

      The function returns `FALSE` if the table does not exist.
- Functions that manage persistent error log data from external tables that were defined with the `PERSISTENTLY` keyword.
  Note: The functions that manage persistent error log data from external tables are defined in the file `$GPHOME/share/postgresql/contrib/gpexterrorhandle.sql`. The functions must be installed in the databases that use persistent error log data from an external table.

This `psql` command installs the functions into the database `testdb`.

```
psql -d test -U gpadmin -f $GPHOME/share/postgresql/contrib/gpexterrorhandle.sq
l
```

- The SQL function `gp_read_persistent_error_log('table_name')` displays persistent log data for an external table.

  The function returns no data if you created the external table without the `PERSISTENTLY` keyword. The function returns persistent log data for an external table even after the table has been dropped.

- The SQL function `gp_truncate_persistent_error_log('table_name')` truncates persistent log data for a table.

  For persistent log data, you must manually delete the data. Dropping the external table does not delete persistent log data.

- These items apply to both non-persistent and persistent error log data and the related functions.
  - The `gp_read_*` functions require `SELECT` privilege on the table.
  - The `gp_truncate_*` functions require owner privilege on the table.
  - You can use 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. Non-persistent and persistent error log data must be deleted with their respective `gp_truncate_*` functions.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 DEFINER
  | 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):

```
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:

```
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:

```
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:

```
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:

```
$$Dianne's horse$$
$SomeTag$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()`:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

CREATE INDEX

Defines a new index.

Synopsis

```
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`.

Currently, only the B-tree and GiST index methods support multicolumn indexes. Up to 32 fields can be specified by default. Only B-tree currently supports unique indexes.

GPORCA supports only B-tree, bitmap, and GiST indexes. GPORCA ignores indexes created with unsupported indexing methods.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Note: Most procedural languages for Greenplum Database have been made into "extensions," and should therefore be installed with CREATE EXTENSION, not `CREATE LANGUAGE`. Using `CREATE LANGUAGE` directly should be restricted to extension installation scripts. If you have a "bare" language in your database, perhaps as a result of an upgrade, you can convert it to an extension using `CREATE EXTENSION langname FROM unpackaged`.

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 name must be unique among the languages in the database. Built-in support is included for `plpgsql`, `plperl`, and `plpythonu`. 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 is to use `CREATE EXTENSION` instead of `CREATE LANGUAGE`. For example:

```
CREATE EXTENSION plperl;
```

For a language not known in the `pg_pltemplate` catalog:

```
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 EXTENSION`, `CREATE FUNCTION`, `DROP EXTENSION`, `DROP LANGUAGE`, `GRANT DO`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

CREATE OPERATOR

Defines a new operator.

Synopsis

```
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:

```
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 <=

`scalargtsel` 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

```
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:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 1. B-tree and Bitmap Strategies

Operation	Strategy Number
less than	1
less than or equal	2
equal	3
greater than or equal	4
greater than	5

Table 2. GiST Two-Dimensional Strategies (R-Tree)

Operation	Strategy Number
strictly left of	1
does not extend to right of	2
overlaps	3
does not extend to left of	4
strictly right of	5
same	6
contains	7
contained by	8
does not extend above	9
strictly below	10
strictly above	11
does not extend below	12

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 `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 *support function numbers* as follows:

Table 3. B-tree and Bitmap Support Functions

Function	Support Number

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.	1

Table 4. GiST Support Functions

Function	Support Number
consistent - determine whether key satisfies the query qualifier.	1
union - compute union of a set of keys.	2
compress - compute a compressed representation of a key or value to be indexed.	3
decompress - compute a decompressed representation of a compressed key.	4
penalty - compute penalty for inserting new key into subtree with given subtree's key.	5
picksplit - determine which entries of a page are to be moved to the new page and compute the union keys for resulting pages.	6
equal - compare two keys and return true if they are equal.	7

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):

```
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

`ALTER OPERATOR CLASS`, `DROP OPERATOR CLASS`, `CREATE FUNCTION`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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` command 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 a `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
: If not specified, only superusers and the protocol owner can create external tables using the protocol. If specified, superusers and the protocol owner can `GRANT` permissions on the protocol to other database roles.

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 handles external tables of type `file`, `gpfdist`, and `gpfdists` internally. The custom protocol `gphdfs` (deprecated) is registered in Greenplum Database by default. See Configuring and Using S3 External Tables for information about enabling the `S3` protocol. See the

Greenplum Platform Extension Framework (PXF) documentation for information about configuring the PXF extension and using the `pxf` protocol.

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`, GRANT

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

CREATE RESOURCE GROUP

Defines a new resource group.

Synopsis

```
CREATE RESOURCE GROUP name WITH (group_attribute=value [, ... ])
```

where *group_attribute* is:

```
CPU_RATE_LIMIT=integer | CPUSET=tuple
MEMORY_LIMIT=integer
[ CONCURRENCY=integer ]
[ MEMORY_SHARED_QUOTA=integer ]
[ MEMORY_SPILL_RATIO=integer ]
[ MEMORY_AUDITOR= {vmtracker | cgroup} ]
```

Description

Creates a new resource group for Greenplum Database resource management. You can create resource groups to manage resources for roles or to manage the resources of a Greenplum Database external component such as PL/Container.

A resource group that you create to manage a user role identifies concurrent transaction, memory, and CPU limits for the role when resource groups are enabled. You may assign such resource groups to one or more roles.

A resource group that you create to manage the resources of a Greenplum Database external component such as PL/Container identifies the memory and CPU limits for the component when resource groups are enabled. These resource groups use cgroups for both CPU and memory management. Assignment of resource groups to external components is component-specific. For example, you assign a PL/Container resource group when you configure a PL/Container runtime. You cannot assign a resource group that you create for external components to a role, nor can you assign a resource group that you create for roles to an external component.

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 and the external components they may be using, such as PL/Containers.

After creating a resource group for a role, assign the group to one or more roles using the `ALTER ROLE` or `CREATE ROLE` commands.

After you create a resource group to manage the CPU and memory resources of an external component, configure the external component to use the resource group. For example, configure the PL/Container runtime `resource_group_id`.

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 for resource groups defined for roles is 20.

You must set `CONCURRENCY` to zero (0) for resource groups that you create for external components.

Note: You cannot set the `CONCURRENCY` value for the `admin_group` to zero (0).

CPU_RATE_LIMIT *integer*

CPUSET *tuple*
: Required. You must specify only one of `CPU_RATE_LIMIT` or `CPUSET` when you create a resource group.

`CPU_RATE_LIMIT` is 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.

`CPUSET` identifies the CPU cores to reserve for this resource group. The CPU cores that you specify in *tuple* must be available in the system and cannot overlap with any CPU cores that you specify for other resource groups.

tuple is a comma-separated list of single core numbers or core number intervals. You must enclose *tuple* in single quotes, for example, '1,3-4'.

Note: You can configure `CPUSET` for a resource group only after you have enabled resource group-based resource management for your Greenplum Database cluster.

MEMORY_LIMIT *integer*
: Required. The total percentage of Greenplum Database memory resources to reserve for this resource group. The minimum memory percentage you can specify for a resource group is 0. The maximum is 100.

When you specify a `MEMORY_LIMIT` of 0, Greenplum Database reserves no memory for the resource group, but uses global shared memory to fulfill all memory requests in the group. If `MEMORY_LIMIT` is 0, `MEMORY_SPILL_RATIO` must also be 0.

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 this threshold is reached, a transaction spills to disk. You can specify an integer percentage value from 0 to 100 inclusive. The default `MEMORY_SPILL_RATIO` value is 20. When `MEMORY_SPILL_RATIO` is 0, Greenplum Database uses the `statement_mem` server configuration parameter value to control initial query operator memory.

MEMORY_AUDITOR {vmtracker | cgroup}

The memory auditor for the resource group. Greenplum Database employs virtual memory tracking for role resources and cgroup memory tracking for resources used by external components. The default `MEMORY_AUDITOR` is `vmtracker`. When you create a resource group with `vmtracker` memory auditing, Greenplum Database tracks that resource group's memory internally.

When you create a resource group specifying the `cgroup` `MEMORY_AUDITOR`, Greenplum Database defers the accounting of memory used by that resource group to cgroups. `CONCURRENCY` must be zero (0) for a resource group that you create for external components such as PL/Container. You cannot assign a resource group that you create for external components to a Greenplum Database role.

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 20, 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);
```

Create a resource group to manage PL/Container resources specifying a memory limit of 10, and a CPU limit of 10:

```
CREATE RESOURCE GROUP plc_run1 WITH (MEMORY_LIMIT=10, CPU_RATE_LIMIT=10,
  CONCURRENCY=0, MEMORY_AUDITOR=cgroup);
```

Create a resource group with a memory limit percentage of 11 to which you assign CPU cores 1 to 3:

```
CREATE RESOURCE GROUP rgroup3 WITH (CPUSET='1-3', MEMORY_LIMIT=11);
```

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 queueing.

Note: GPORCA and the legacy Greenplum Database query optimizer utilize different query costing models and may compute different costs for the same query. The Greenplum Database resource queue resource management scheme neither differentiates nor aligns costs between GPORCA and the legacy optimizer; it uses the literal cost value returned from the optimizer to throttle queries.

When resource queue-based resource management is active, use the `MEMORY_LIMIT` and `ACTIVE_STATEMENTS` limits for resource queues rather than configuring cost-based limits. Even when using GPORCA, Greenplum Database may fall back to using the legacy query optimizer for

certain queries, so using cost-based limits can lead to unexpected results.

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:

```
=> 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:

```
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):

```
CREATE RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=20,
  MEMORY_LIMIT='2000MB');
```

Create a resource queue with a query cost limit of 3000.0:

```
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. Allow small queries with a cost under 500 to run immediately:

```
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:

```
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:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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
      [ ( 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 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. Valid types are `gpfdist`, `gpfdists`, `http`, and `https`. `NOCREATEEXTTABLE` (the default type) 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.
Use the `GRANT...ON PROTOCOL` command to allow users to create and use external tables with a custom protocol type, including the `gphdfs` (deprecated), `s3`, and `pxf` protocols included with Greenplum Database.

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 *connlimit*
: 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 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.
You cannot assign a resource group that you create for an external component to a role.

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-superuser-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 that can create readable and writable external tables of type 'gpfdist':

```
CREATE ROLE jan WITH CREATEEXTTABLE(type='readable', protocol='gpfdist')
   CREATEEXTTABLE(type='writable', protocol='gpfdist');
```

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 `NOINHERIT` attribute, while roles are given the `INHERIT` attribute.

See Also

`SET ROLE`, `ALTER ROLE`, `DROP ROLE`, `GRANT`, `REVOKE`, `CREATE RESOURCE QUEUE` `CREATE RESOURCE GROUP`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

`CREATE RULE` defines a new rule applying to a specified table or view. `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.

`ON SELECT` rules must be unconditional `INSTEAD` rules and must have actions that consist of a single `SELECT` command. Thus, an `ON SELECT` rule effectively turns the table into a view, whose visible contents are the rows returned by the rule's `SELECT` command rather than whatever had been stored in the table (if anything). It is considered better style to write a `CREATE VIEW` command than to create a real table and define an `ON SELECT` rule for it.

You can create the illusion of an updatable view by defining `ON INSERT`, `ON UPDATE`, and `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 `INSTEAD` rule for each action you wish to allow on the view. If the rule is conditional, or is not `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 `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-`INSTEAD`; in the cases where they are applied, they add to the default `INSTEAD NOTHING` action. (This method does not currently work to support `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 `SELECT`, `INSERT`, `UPDATE`, or `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 `ONUPDATE` 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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 [ ... ]
[ 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
           [(...)]
```

```
        ) ]
   )
```

where *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 ]
```

where *storage_directive* for a column is:

```
   COMPRESSTYPE={ZLIB | QUICKLZ | RLE_TYPE | NONE}
    [COMPRESSLEVEL={0-9} ]
    [BLOCKSIZE={8192-2097152} ]
```

where *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.

COMPRESSTYPE — Set to `ZLIB` (the default), `RLE_TYPE`, or `QUICKLZ`[1] 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: [1]QuickLZ 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 `TIMESTAMP`, delta compression is also applied if the `COMPRESSTYPE` 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:
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 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`:

```
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):

```
CREATE TABLE films (
code        char(5) CONSTRAINT firstkey PRIMARY KEY,
title       varchar(40) NOT NULL,
did         integer NOT NULL,
date_prod   date,
kind        varchar(10),
len         interval hour to minute
);

CREATE TABLE distributors (
did    integer PRIMARY KEY DEFAULT nextval('serial'),
name   varchar(40) NOT NULL CHECK (name <> '')
);
```

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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[1] 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: [1]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. `DISTIBUTED 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`:

```
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:

```
PREPARE recentfilms(date) AS SELECT * FROM films WHERE
date_prod > $1;
CREATE TEMP TABLE films_recent WITH (OIDS) ON COMMIT DROP AS
EXECUTE recentfilms('2007-01-01');
```

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

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

tablespacename
: 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*

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 (
    INPUT = input_function,
    OUTPUT = output_function
    [, RECEIVE = receive_function]
    [, SEND = send_function]
    [, TYPMOD_IN = type_modifier_input_function ]
    [, TYPMOD_OUT = type_modifier_output_function ]
    [, INTERNALLENGTH = {internallength | VARIABLE}]
    [, PASSEDBYVALUE]
    [, ALIGNMENT = alignment]
    [, STORAGE = storage]
    [, DEFAULT = default]
    [, ELEMENT = element]
    [, DELIMITER = delimiter] )

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. If this function is not supplied, the type cannot participate in binary input. The binary representation should be chosen to be cheap to convert to internal form, while being reasonably portable. (For example, the standard integer data types use network byte order as the external binary representation, while the internal representation is in the machine's native byte order.) The receive function should perform adequate checking to ensure that the value is valid. The receive function may 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` key word. (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` key word. 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 `NAMEDATALEN - 2`, rather than the `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` *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:

```
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 `mood` and uses it in a table definition.

```
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 `box` and then uses the type in a table definition:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 only difference between `CREATE ROLE` and `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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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:

```
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:

```
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:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 not 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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 `TRUNCATE` command frees the disk space occupied by the deleted rows without requiring a `VACUUM` of the table. This improves scan performance of subsequent queries, and benefits ELT workloads that frequently insert and delete from temporary tables.

Execution of `UPDATE` and `DELETE` commands directly on a specific partition (child table) of a partitioned table is not supported. Instead, these commands must be executed on the root partitioned table, the table created with the `CREATE TABLE` command.

For a partitioned table, all the child tables are locked during the `DELETE` operation.

Examples

Delete all films but musicals:

```
DELETE FROM films WHERE kind <> 'Musical';
```

Clear the table films:

```
DELETE FROM films;
```

Delete using a join:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

DISCARD

Discards the session state.

Synopsis

```
DISCARD { ALL | PLANS | TEMPORARY | TEMP }
```

Description

`DISCARD` releases internal resources associated with a database session. This command is useful for partially or fully resetting the session's state. There are several subcommands to release different types of resources. `DISCARD ALL` is not supported by Greenplum Database.

Parameters

PLANS
: Releases all cached query plans, forcing re-planning to occur the next time the associated prepared statement is used.

SEQUENCES
: Discards all cached sequence-related state, including any preallocated sequence values that have not yet been returned by `nextval()`. (See `CREATE SEQUENCE` for a description of preallocated sequence values.)

TEMPORARY/TEMP
: Drops all temporary tables created in the current session.

ALL
: Releases all temporary resources associated with the current session and resets the session to its initial state.

  Note: Greenplum Database does not support `DISCARD ALL` and returns a notice message if you attempt to run the command.

As an alternative, you can the run following commands to release temporary session resources:

```
SET SESSION AUTHORIZATION DEFAULT;
RESET ALL;
DEALLOCATE ALL;
CLOSE ALL;
SELECT pg_advisory_unlock_all();
DISCARD PLANS;
DISCARD SEQUENCES;
DISCARD TEMP;
```

Compatibility

`DISCARD` is a Greenplum Database extension.

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

DO

Executes an anonymous code block as a transient anonymous function.

Synopsis

```
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`:

```
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.

```
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

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most

up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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:

```
DROP EXTERNAL TABLE IF EXISTS staging;
```

Compatibility

There is no `DROP EXTERNAL TABLE` statement in the SQL standard.

See Also

`CREATE EXTERNAL TABLE`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

DROP FILESPACE

Removes a filespace.

Synopsis

```
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`:

```
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*

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

DROP FUNCTION

Removes a function.

Synopsis

```
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:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 `~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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

DROP OPERATOR CLASS

Removes an operator class.

Synopsis

```
DROP OPERATOR CLASS [IF EXISTS] name USING index_method [CASCADE | RESTRICT]
```

Description

`DROP OPERATOR` 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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 prococol, 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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

DROP RESOURCE GROUP

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. When you drop a resource group, the memory and CPU resources reserved by the group are returned to Greenplum Database.

To drop a role resource group, the group cannot be assigned to any roles, nor can it have any statements pending or running in the group. If you drop a resource group that you created for an external component, the behavior is determined by the external component. For example, dropping a resource group that you assigned to a PL/Container runtime kills running containers 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 a 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:

```
ALTER ROLE bob RESOURCE GROUP NONE;
```

Remove the resource group named `adhoc`:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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:

```
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:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

See Also

`REASSIGN OWNED`, `DROP OWNED`, `CREATE ROLE`, `ALTER ROLE`, `SET ROLE`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

See Also

`CREATE TABLE`, `ALTER TABLE`, `TRUNCATE`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

name
: The name (optionally schema-qualified) of the data type to remove.

CASCADE
: Automatically drop objects that depend on the type (such as table columns, functions, operators).

RESTRICT
: Refuse to drop the type if any objects depend on it. This is the default.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

DROP VIEW

Removes a view.

Synopsis

```
DROP VIEW [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

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`;

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

END

Commits the current transaction.

Synopsis

```
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:

```
END;
```

Compatibility

`END` is a Greenplum Database extension that provides functionality equivalent to `COMMIT`, which is specified in the SQL standard.

See Also

`BEGIN`, `ROLLBACK`, `COMMIT`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

EXPLAIN

Shows the query plan of a statement.

Synopsis

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

FETCH

Retrieves rows from a query using a cursor.

Synopsis

```
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;
 code  |          title          | did | date_prod  |   kind   |  len
-------+-------------------------+-----+------------+----------+-------
 BL101 | The Third Man           | 101 | 1949-12-23 | Drama    | 01:44
 BL102 | The African Queen       | 101 | 1951-08-11 | Romantic | 01:43
 JL201 | Une Femme est une Femme | 102 | 1961-03-12 | Romantic | 01:25
 P_301 | Vertigo                 | 103 | 1958-11-14 | Action   | 02:08
 P_302 | Becket                  | 103 | 1964-02-03 | Drama    | 02:28
```

-- 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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN DATA WRAPPER fdwname [, ...]
    TO { [ GROUP ] rolename | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN SERVER servername [, ...]
    TO { [ GROUP ] 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, foreign table, sequence, database, foreign-data wrapper, foreign server, 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 able to allow 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

  ```
  GRANT SELECT ON PROTOCOL protocolname TO username
  ```

- To allow a user to create a writable external table with a trusted protocol

```
GRANT INSERT ON PROTOCOL protocolname TO username
```

- To allow a user to create both readable and writable external tables with a trusted protocol

```
GRANT ALL ON PROTOCOL protocolname TO username
```

You can also use this command to grant users permissions to create and use `hdfs`, `s3`, and `pxf` external tables. However, external tables of type `http`, `https`, `gpfdist`, and `gpfdists`, are implemented internally in Greenplum Database instead of as custom protocols. For these types, use the `CREATE ROLE` or `ALTER ROLE` command to set the `CREATEEXTTABLE` or `NOCREATEEXTTABLE` attribute for each user. See CREATE ROLE for syntax and examples.

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.

For foreign-data wrappers, this privilege enables the grantee to create new servers using that foreign-data wrapper.

For servers, this privilege enables the grantee to create, alter, and drop their own user's user mappings associated with that server. Also, it enables the grantee to query the options of the server and associated user mappings.

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.

The `GRANT` command cannot be used to set privileges for the protocols `file`, `gpfdist`, or `gpfdists`. These protocols are implemented internally in Greenplum Database. Instead, use the CREATE ROLE or ALTER ROLE command to set the `CREATEEXTTABLE` attribute for the role.

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`, CREATE ROLE, ALTER ROLE

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 a partitioned table, all the child tables are locked during the `INSERT` operation.

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`:

```
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:

```
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:

```
INSERT INTO films VALUES ('UA502', 'Bananas', 105, DEFAULT,
```

```
'Comedy', '82 minutes');
```

To insert a row consisting entirely of default values:

```
INSERT INTO films DEFAULT VALUES;
```

To insert multiple rows using the multirow `VALUES` syntax:

```
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`:

```
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

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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:

```
LOAD '/usr/local/greenplum-db/lib/myfuncs.so';
```

Compatibility

`LOAD` is a Greenplum Database extension.

See Also

`CREATE FUNCTION`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 `SELECT` command acquires 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 by `VACUUM` (without `FULL`) on heap tables and `ANALYZE`.
- 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. 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, SHAREROW 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:

```
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:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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;
 code  | title  | did | date_prod  |  kind  |  len
-------+--------+-----+------------+--------+-------
 P_303 | 48 Hrs | 103 | 1982-10-22 | Action | 01:37
(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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 possible). 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:

```
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:

```
PREPARE usrrptplan (int) AS SELECT * FROM users u, logs l
WHERE u.usrid=$1 AND u.usrid=l.usrid AND l.date = $2;
EXECUTE usrrptplan(1, current_date);
```

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

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

REINDEX

Rebuilds indexes.

Synopsis

```
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 is, it 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.

`REINDEX` does not update the `reltuples` and `relpages` statistics for the index. To update those statistics, run `ANALYZE` on the table after reindexing.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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;
    INSERT INTO table1 VALUES (4);
    RELEASE SAVEPOINT my_savepoint;
COMMIT;
```

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

RESET

Restores the value of a system configuration parameter to the default value.

Synopsis

```
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:

```
RESET statement_mem;
```

Compatibility

`RESET` is a Greenplum Database extension.

See Also

`SET`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

REVOKE

Removes access privileges.

Synopsis

```
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

`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. Similarly, revoking `SELECT` from a user might not prevent that user from using `SELECT` if `PUBLIC` or another membership role still has `SELECT` rights.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 `FETCH` command inside a savepoint that is later rolled back, the cursor position remains at the position that `FETCH` left it pointing to (that is, `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 `ROLLBACK TO SAVEPOINT`, the cursor can no longer be used.

Examples

To undo the effects of the commands executed after `my_savepoint` was established:

```
ROLLBACK TO SAVEPOINT my_savepoint;
```

Cursor positions are not affected by a savepoint rollback:

```
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;
```

Compatibility

The SQL standard specifies that the key word `SAVEPOINT` is mandatory, but Greenplum Database (and Oracle) allow it to be omitted. SQL allows only `WORK`, not `TRANSACTION`, as a noise word after `ROLLBACK`. Also, SQL has an optional clause `AND [NO] CHAIN` which is not currently supported by Greenplum Database. Otherwise, this command conforms to the SQL standard.

See Also

BEGIN, COMMIT, SAVEPOINT, RELEASE SAVEPOINT, ROLLBACK

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

SAVEPOINT

Defines a new savepoint within the current transaction.

Synopsis

```
SAVEPOINT savepoint_name
```

Description

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

savepoint_name
: 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:

```
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:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

SELECT

Retrieves rows from a table or view.

Synopsis

```
[ 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] [...]]
```

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: [1]The `RECURSIVE` keyword is a Beta feature.

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.
Note: The `RECURSIVE` keyword is a Beta feature.

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`.

```
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`.

```
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, `JOIN`s 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:

```
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:

```
WINDOW window_name AS (window_specification)
```

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
```

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, A UNION B INTERSECT C will be read as 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 EXCEPT clause has this general form:

```
select_statement EXCEPT [ALL] select_statement
```

where *select_statement* is any SELECT statement without an ORDER BY, LIMIT, FOR UPDATE, or FOR SHARE clause.

The EXCEPT operator computes the set of rows that are in the result of the left SELECT statement but not in the result of the right one.

The result of EXCEPT 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 max(*m*-*n*,0) times in the result set.

Multiple EXCEPT operators in the same SELECT statement are evaluated left to right, unless parentheses dictate otherwise. EXCEPT binds at the same level as UNION.

Currently, FOR UPDATE and FOR SHARE may not be specified either for an EXCEPT result or for any input of an EXCEPT.

The ORDER BY Clause

The optional ORDER BY clause has this general form:

```
ORDER BY expression [ASC | DESC | USING operator] [NULLS { FIRST | LAST}] [, ...]
```

where *expression* can be the name or ordinal number of an output column (SELECT list item), or it can be an arbitrary expression formed from input-column values.

The 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 AS clause.

It is also possible to use arbitrary expressions in the ORDER BY clause, including columns that do not appear in the SELECT result list. Thus the following statement is valid:

```
SELECT name FROM distributors ORDER BY code;
```

A limitation of this feature is that an ORDER BY clause applying to the result of a UNION, INTERSECT, or EXCEPT clause may only specify an output column name or number, not an expression.

If an ORDER BY expression is a simple name that matches both a result column name and an input column name, ORDER BY will interpret it as the result column name. This is the opposite of the choice that 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 `ASC` (ascending) or `DESC` (descending) after any expression in the `ORDER BY` clause. If not specified, `ASC` is assumed by default. Alternatively, a specific ordering operator name may be specified in the `USING` clause. `ASC` is usually equivalent to `USING <` and `DESC` is usually equivalent to `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 `NULLS LAST` is specified, null values sort after all non-null values; if `NULLS FIRST` is specified, null values sort before all non-null values. If neither is specified, the default behavior is `NULLS LAST` when `ASC` is specified or implied, and `NULLS FIRST` when `DESC` is specified (thus, the default is to act as though nulls are larger than non-nulls). When `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:

```
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:

```
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:

```
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:

```
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:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

SELECT INTO

Defines a new table from the results of a query.

Synopsis

```
[ 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]
    [...]]
```

Description

Note: [1]The `RECURSIVE` keyword is a Beta feature.

`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`.
Note: The `RECURSIVE` keyword is a Beta feature.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

configuration_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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

SET ROLE

Sets the current role identifier of the current session.

Synopsis

```
SET [SESSION | LOCAL] ROLE rolename

SET [SESSION | LOCAL] ROLE NONE

RESET ROLE
```

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
: 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 `SET ROLE` to a non-superuser role, she loses her superuser privileges.

`SET ROLE` has effects comparable to `SET SESSION AUTHORIZATION`, but the privilege checks involved are quite different. Also, `SET SESSION AUTHORIZATION` determines which roles are allowable for later `SET ROLE` commands, whereas changing roles with `SET ROLE` does not change the set of roles allowed to a later `SET ROLE`.

Examples

```
SELECT SESSION_USER, CURRENT_USER;
 session_user | current_user
--------------+--------------
 peter        | peter

SET ROLE 'paul';

SELECT SESSION_USER, CURRENT_USER;
 session_user | current_user
--------------+--------------
 peter        | paul
```

Compatibility

Greenplum Database allows identifier syntax (*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 `SESSION` and `LOCAL` modifiers are a Greenplum Database extension, as is the `RESET` syntax.

See Also

`SET SESSION AUTHORIZATION`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 *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 `setuid` functions and similar mechanisms; it can also be changed by `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
: 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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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]
```

where *transaction_mode* is one of:

```
   ISOLATION LEVEL {SERIALIZABLE | READ COMMITTED | READ UNCOMMITTED}
```

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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
: The name of a system configuration parameter.

ALL
: Shows the current value of all configuration parameters.

Examples

Show the current setting of the parameter `search_path`:

```
SHOW search_path;
```

Show the current setting of all parameters:

```
SHOW ALL;
```

Compatibility

`SHOW` is a Greenplum Database extension.

See Also

`SET`, `RESET`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

For a partitioned table, all the child tables are locked during the `UPDATE` operation.

Examples

Change the word `Drama` to `Dramatic` in the column `kind` of the table `films`:

```
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`:

```
UPDATE weather SET temp_lo = temp_lo+1, temp_hi =
temp_lo+15, prcp = DEFAULT
WHERE city = 'San Francisco' AND date = '2016-07-03';
```

Use the alternative column-list syntax to do the same update:

```
UPDATE weather SET (temp_lo, temp_hi, prcp) = (temp_lo+1,
temp_lo+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):

```
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:

```
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.

```
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:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

VACUUM

Garbage-collects and optionally analyzes a database.

Synopsis

```
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.

Note: Starting in Greenplum 5.x, `VACUUM` is supported with persistent table system catalogs, and is required to manage free space.

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: PostgreSQL has a separate optional server process called the *autovacuum daemon*, whose purpose is to automate the execution of `VACUUM` and `ANALYZE` commands. Greenplum Database enables the autovacuum daemon to perform `VACUUM` operations only on the Greenplum Database template database `template0`. Autovacuum is enabled for `template0` because connections are not allowed to `template0`. The autovacuum daemon performs `VACUUM` operations on `template0` to manage transaction IDs (XIDs) and help avoid transaction ID wraparound issues in `template0`.

Manual `VACUUM` operations must be performed in user-defined databases to manage transaction IDs (XIDs) in those databases.

`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:

```
VACUUM;
```

Vacuum a specific table only:

```
VACUUM mytable;
```

Vacuum all tables in the current database and collect statistics for the query optimizer:

```
VACUUM ANALYZE;
```

Compatibility

There is no VACUUM statement in the SQL standard.

See Also

ANALYZE

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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
SELECT 3, 'three';
```

More usually, VALUES is used within a larger SQL command. The most common use is in INSERT:

```
INSERT INTO films (code, title, did, date_prod, kind)
    VALUES ('T_601', 'Yojimbo', 106, '1961-06-16', 'Drama');
```

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:

```
INSERT INTO films VALUES
    ('UA502', 'Bananas', 105, DEFAULT, 'Comedy', '82
minutes'),
    ('T_601', 'Yojimbo', 106, DEFAULT, 'Drama', DEFAULT);
```

VALUES can also be used where a sub-SELECT might be written, for example in a FROM clause:

```
SELECT f.* FROM films f, (VALUES('MGM', 'Horror'), ('UA',
'Sci-Fi')) AS t (studio, kind) WHERE f.studio = t.studio AND
f.kind = t.kind;
UPDATE employees SET salary = salary * v.increase FROM
(VALUES(1, 200000, 1.2), (2, 400000, 1.4)) AS v (depno,
target, increase) WHERE employees.depno = v.depno AND
employees.sales >= v.target;
```

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:

```
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`

Parent topic: SQL Command Reference

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

SQL 2008 Optional Feature Compliance

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 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
B011	Embedded Ada	NO	
B012	Embedded C	NO	Due to issues with PostgreSQL `ecpg`
B013	Embedded COBOL	NO	
B014	Embedded Fortran	NO	
B015	Embedded MUMPS	NO	
B016	Embedded Pascal	NO	
B017	Embedded PL/I	NO	
B021	Direct SQL	YES	
B031	Basic dynamic SQL	NO	
B032	Extended dynamic SQL	NO	
B033	Untyped SQL-invoked function arguments	NO	
B034	Dynamic specification of cursor attributes	NO	
B035	Non-extended descriptor names	NO	
B041	Extensions to embedded SQL exception declarations	NO	
B051	Enhanced execution rights	NO	
B111	Module language Ada	NO	
B112	Module language C	NO	
B113	Module language COBOL	NO	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
B114	Module language Fortran	NO	
B115	Module language MUMPS	NO	
B116	Module language Pascal	NO	
B117	Module language PL/I	NO	
B121	Routine language Ada	NO	
B122	Routine language C	NO	
B123	Routine language COBOL	NO	
B124	Routine language Fortran	NO	
B125	Routine language MUMPS	NO	
B126	Routine language Pascal	NO	
B127	Routine language PL/I	NO	
B128	Routine language SQL	NO	
E011	Numeric data types	YES	
E011-01	`INTEGER` and `SMALLINT` data types	YES	
E011-02	`DOUBLE PRECISION` and `FLOAT` data types	YES	
E011-03	`DECIMAL` and `NUMERIC` data types	YES	
E011-04	Arithmetic operators	YES	
E011-05	Numeric comparison	YES	
E011-06	Implicit casting among the numeric data types	YES	
E021	Character data types	YES	
E021-01	`CHARACTER` data type	YES	
E021-02	`CHARACTER VARYING` data type	YES	
E021-03	Character literals	YES	
E021-04	`CHARACTER_LENGTH` function	YES	Trims trailing spaces from `CHARACTER` values before counting
E021-05	`OCTET_LENGTH` function	YES	
E021-06	`SUBSTRING` function	YES	
E021-07	Character concatenation	YES	
E021-08	`UPPER` and `LOWER` functions	YES	
E021-09	`TRIM` function	YES	
E021-10	Implicit casting among the character string types	YES	
E021-11	`POSITION` function	YES	
E021-12	Character comparison	YES	
E031	Identifiers	YES	
E031-01	Delimited identifiers	YES	
E031-02	Lower case identifiers	YES	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
E031-03	Trailing underscore	YES	
E051	Basic query specification	YES	
E051-01	`SELECT DISTINCT`	YES	
E051-02	`GROUP BY` clause	YES	
E051-03	`GROUP BY` can contain columns not in `SELECT` list	YES	
E051-04	`SELECT` list items can be renamed	YES	
E051-05	`HAVING` clause	YES	
E051-06	Qualified * in `SELECT` list	YES	
E051-07	Correlation names in the `FROM` clause	YES	
E051-08	Rename columns in the `FROM` clause	YES	
E061	Basic predicates and search conditions	YES	
E061-01	Comparison predicate	YES	
E061-02	`BETWEEN` predicate	YES	
E061-03	`IN` predicate with list of values	YES	
E061-04	`LIKE` predicate	YES	
E061-05	`LIKE` predicate `ESCAPE` clause	YES	
E061-06	`NULL` predicate	YES	
E061-07	Quantified comparison predicate	YES	
E061-08	`EXISTS` predicate	YES	Not all uses work in Greenplum
E061-09	Subqueries in comparison predicate	YES	
E061-11	Subqueries in IN predicate	YES	
E061-12	Subqueries in quantified comparison predicate	YES	
E061-13	Correlated subqueries	YES	
E061-14	Search condition	YES	
E071	Basic query expressions	YES	
E071-01	`UNION DISTINCT` table operator	YES	
E071-02	`UNION ALL` table operator	YES	
E071-03	`EXCEPT DISTINCT` table operator	YES	
E071-05	Columns combined via table operators need not have exactly the same data type	YES	
E071-06	Table operators in subqueries	YES	
E081	Basic Privileges	NO	Partial sub-feature support
E081-01	`SELECT` privilege	YES	
E081-02	`DELETE` privilege	YES	
E081-03	`INSERT` privilege at the table level	YES	
E081-04	`UPDATE` privilege at the table level	YES	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
E081-05	`UPDATE` privilege at the column level	NO	
E081-06	`REFERENCES` privilege at the table level	NO	
E081-07	`REFERENCES` privilege at the column level	NO	
E081-08	`WITH GRANT OPTION`	YES	
E081-09	`USAGE` privilege	YES	
E081-10	`EXECUTE` privilege	YES	
E091	Set Functions	YES	
E091-01	`AVG`	YES	
E091-02	`COUNT`	YES	
E091-03	`MAX`	YES	
E091-04	`MIN`	YES	
E091-05	`SUM`	YES	
E091-06	`ALL` quantifier	YES	
E091-07	`DISTINCT` quantifier	YES	
E101	Basic data manipulation	YES	
E101-01	`INSERT` statement	YES	
E101-03	Searched `UPDATE` statement	YES	
E101-04	Searched `DELETE` statement	YES	
E111	Single row `SELECT` statement	YES	
E121	Basic cursor support	YES	
E121-01	`DECLARE CURSOR`	YES	
E121-02	`ORDER BY` columns need not be in select list	YES	
E121-03	Value expressions in `ORDER BY` clause	YES	
E121-04	`OPEN` statement	YES	
E121-06	Positioned `UPDATE` statement	NO	
E121-07	Positioned `DELETE` statement	NO	
E121-08	`CLOSE` statement	YES	
E121-10	`FETCH` statement implicit `NEXT`	YES	
E121-17	`WITH HOLD` cursors	YES	
E131	Null value support	YES	
E141	Basic integrity constraints	YES	
E141-01	`NOT NULL` constraints	YES	
E141-02	`UNIQUE` constraints of `NOT NULL` columns	YES	Must be the same as or a superset of the Greenplum distribution key
E141-03	`PRIMARY KEY` constraints	YES	Must be the same as or a superset of the Greenplum distribution key

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
E141-04	Basic `FOREIGN KEY` constraint with the `NO ACTION` default for both referential delete action and referential update action	NO	
E141-06	`CHECK` constraints	YES	
E141-07	Column defaults	YES	
E141-08	`NOT NULL` inferred on `PRIMARY KEY`	YES	
E141-10	Names in a foreign key can be specified in any order	YES	Foreign keys can be declared but are not enforced in Greenplum
E151	Transaction support	YES	
E151-01	`COMMIT` statement	YES	
E151-02	`ROLLBACK` statement	YES	
E152	Basic SET TRANSACTION statement	YES	
E152-01	`ISOLATION LEVEL SERIALIZABLE` clause	YES	
E152-02	`READ ONLY` and `READ WRITE` clauses	YES	
E153	Updatable queries with subqueries	NO	
E161	SQL comments using leading double minus	YES	
E171	SQLSTATE support	YES	
E182	Module language	NO	
F021	Basic information schema	YES	
F021-01	`COLUMNS` view	YES	
F021-02	`TABLES` view	YES	
F021-03	`VIEWS` view	YES	
F021-04	`TABLE_CONSTRAINTS` view	YES	
F021-05	`REFERENTIAL_CONSTRAINTS` view	YES	
F021-06	`CHECK_CONSTRAINTS` view	YES	
F031	Basic schema manipulation	YES	
F031-01	`CREATE TABLE` statement to create persistent base tables	YES	
F031-02	`CREATE VIEW` statement	YES	
F031-03	`GRANT` statement	YES	
F031-04	`ALTER TABLE` statement: `ADD COLUMN` clause	YES	
F031-13	`DROP TABLE` statement: `RESTRICT` clause	YES	
F031-16	`DROP VIEW` statement: `RESTRICT` clause	YES	
F031-19	`REVOKE` statement: `RESTRICT` clause	YES	
F032	`CASCADE` drop behavior	YES	
F033	`ALTER TABLE` statement: `DROP COLUMN` clause	YES	
F034	Extended REVOKE statement	YES	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
F034-01	`REVOKE` statement performed by other than the owner of a schema object	YES	
F034-02	`REVOKE` statement: `GRANT OPTION FOR` clause	YES	
F034-03	`REVOKE` statement to revoke a privilege that the grantee has `WITH GRANT OPTION`	YES	
F041	Basic joined table	YES	
F041-01	Inner join (but not necessarily the `INNER` keyword)	YES	
F041-02	`INNER` keyword	YES	
F041-03	`LEFT OUTER JOIN`	YES	
F041-04	`RIGHT OUTER JOIN`	YES	
F041-05	Outer joins can be nested	YES	
F041-07	The inner table in a left or right outer join can also be used in an inner join	YES	
F041-08	All comparison operators are supported (rather than just `=`)	YES	
F051	Basic date and time	YES	
F051-01	`DATE` data type (including support of `DATE` literal)	YES	
F051-02	`TIME` data type (including support of `TIME` literal) with fractional seconds precision of at least 0	YES	
F051-03	`TIMESTAMP` data type (including support of `TIMESTAMP` literal) with fractional seconds precision of at least 0 and 6	YES	
F051-04	Comparison predicate on `DATE`, `TIME`, and `TIMESTAMP` data types	YES	
F051-05	Explicit `CAST` between datetime types and character string types	YES	
F051-06	`CURRENT_DATE`	YES	
F051-07	`LOCALTIME`	YES	
F051-08	`LOCALTIMESTAMP`	YES	
F052	Intervals and datetime arithmetic	YES	
F053	`OVERLAPS` predicate	YES	
F081	`UNION` and `EXCEPT` in views	YES	
F111	Isolation levels other than SERIALIZABLE	YES	
F111-01	`READ UNCOMMITTED` isolation level	NO	Can be declared but is treated as a synonym for `READ COMMITTED`
F111-02	`READ COMMITTED` isolation level	YES	
F111-03	`REPEATABLE READ` isolation level	NO	Use `SERIALIZABLE`
F121	Basic diagnostics management	NO	
F122	Enhanced diagnostics management	NO	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
F123	All diagnostics	NO	
F131-	Grouped operations	YES	
F131-01	`WHERE`, `GROUP BY`, and `HAVING` clauses supported in queries with grouped views	YES	
F131-02	Multiple tables supported in queries with grouped views	YES	
F131-03	Set functions supported in queries with grouped views	YES	
F131-04	Subqueries with `GROUP BY` and `HAVING` clauses and grouped views	YES	
F131-05	Single row `SELECT` with `GROUP BY` and `HAVING` clauses and grouped views	YES	
F171	Multiple schemas per user	YES	
F181	Multiple module support	NO	
F191	Referential delete actions	NO	
F200	`TRUNCATE TABLE` statement	YES	
F201	`CAST` function	YES	
F202	`TRUNCATE TABLE`: identity column restart option	NO	
F221	Explicit defaults	YES	
F222	`INSERT` statement: `DEFAULT VALUES` clause	YES	
F231	Privilege tables	YES	
F231-01	`TABLE_PRIVILEGES` view	YES	
F231-02	`COLUMN_PRIVILEGES` view	YES	
F231-03	`USAGE_PRIVILEGES` view	YES	
F251	Domain support		
F261	CASE expression	YES	
F261-01	Simple `CASE`	YES	
F261-02	Searched `CASE`	YES	
F261-03	`NULLIF`	YES	
F261-04	`COALESCE`	YES	
F262	Extended `CASE` expression	NO	
F263	Comma-separated predicates in simple `CASE` expression	NO	
F271	Compound character literals	YES	
F281	`LIKE` enhancements	YES	
F291	`UNIQUE` predicate	NO	
F301	`CORRESPONDING` in query expressions	NO	
F302	INTERSECT table operator	YES	
F302-01	`INTERSECT DISTINCT` table operator	YES	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
F302-02	`INTERSECT ALL` table operator	YES	
F304	`EXCEPT ALL` table operator		
F311	Schema definition statement	YES	Partial sub-feature support
F311-01	`CREATE SCHEMA`	YES	
F311-02	`CREATE TABLE` for persistent base tables	YES	
F311-03	`CREATE VIEW`	YES	
F311-04	`CREATE VIEW: WITH CHECK OPTION`	NO	
F311-05	`GRANT` statement	YES	
F312	`MERGE` statement	NO	
F313	Enhanced `MERGE` statement	NO	
F321	User authorization	YES	
F341	Usage Tables	NO	
F361	Subprogram support	YES	
F381	Extended schema manipulation	YES	
F381-01	`ALTER TABLE` statement: `ALTER COLUMN` clause		Some limitations on altering distribution key columns
F381-02	`ALTER TABLE` statement: `ADD CONSTRAINT` clause		
F381-03	`ALTER TABLE` statement: `DROP CONSTRAINT` clause		
F382	Alter column data type	YES	Some limitations on altering distribution key columns
F391	Long identifiers	YES	
F392	Unicode escapes in identifiers	NO	
F393	Unicode escapes in literals	NO	
F394	Optional normal form specification	NO	
F401	Extended joined table	YES	
F401-01	`NATURAL JOIN`	YES	
F401-02	`FULL OUTER JOIN`	YES	
F401-04	`CROSS JOIN`	YES	
F402	Named column joins for LOBs, arrays, and multisets	NO	
F403	Partitioned joined tables	NO	
F411	Time zone specification	YES	Differences regarding literal interpretation
F421	National character	YES	
F431	Read-only scrollable cursors	YES	Forward scrolling only
01	`FETCH` with explicit `NEXT`	YES	
02	`FETCH FIRST`	NO	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
03	`FETCH LAST`	YES	
04	`FETCH PRIOR`	NO	
05	`FETCH ABSOLUTE`	NO	
06	`FETCH RELATIVE`	NO	
F441	Extended set function support	YES	
F442	Mixed column references in set functions	YES	
F451	Character set definition	NO	
F461	Named character sets	NO	
F471	Scalar subquery values	YES	
F481	Expanded `NULL` predicate	YES	
F491	Constraint management	YES	
F501	Features and conformance views	YES	
F501-01	`SQL_FEATURES` view	YES	
F501-02	`SQL_SIZING` view	YES	
F501-03	`SQL_LANGUAGES` view	YES	
F502	Enhanced documentation tables	YES	
F502-01	`SQL_SIZING_PROFILES` view	YES	
F502-02	`SQL_IMPLEMENTATION_INFO` view	YES	
F502-03	`SQL_PACKAGES` view	YES	
F521	Assertions	NO	
F531	Temporary tables	YES	Non-standard form
F555	Enhanced seconds precision	YES	
F561	Full value expressions	YES	
F571	Truth value tests	YES	
F591	Derived tables	YES	
F611	Indicator data types	YES	
F641	Row and table constructors	NO	
F651	Catalog name qualifiers	YES	
F661	Simple tables	NO	
F671	Subqueries in `CHECK`	NO	Intentionally ommitted
F672	Retrospective check constraints	YES	
F690	Collation support	NO	
F692	Enhanced collation support	NO	
F693	SQL-session and client module collations	NO	
F695	Translation support	NO	
F696	Additional translation documentation	NO	
F701	Referential update actions	NO	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
F711	`ALTER` domain	YES	
F721	Deferrable constraints	NO	
F731	`INSERT` column privileges	NO	
F741	Referential `MATCH` types	NO	No partial match
F751	View `CHECK` enhancements	NO	
F761	Session management	YES	
F762	`CURRENT_CATALOG`	NO	
F763	`CURRENT_SCHEMA`	NO	
F771	Connection management	YES	
F781	Self-referencing operations	YES	
F791	Insensitive cursors	YES	
F801	Full set function	YES	
F812	Basic flagging	NO	
F813	Extended flagging	NO	
F831	Full cursor update	NO	
F841	`LIKE_REGEX` predicate	NO	Non-standard syntax for regex
F842	`OCCURENCES_REGEX` function	NO	
F843	`POSITION_REGEX` function	NO	
F844	`SUBSTRING_REGEX` function	NO	
F845	`TRANSLATE_REGEX` function	NO	
F846	Octet support in regular expression operators	NO	
F847	Nonconstant regular expressions	NO	
F850	Top-level `ORDER BY` clause in *query expression*	YES	
F851	Top-level `ORDER BY` clause in subqueries	NO	
F852	Top-level `ORDER BY` clause in views	NO	
F855	Nested `ORDER BY` clause in *query expression*	NO	
F856	Nested `FETCH FIRST` clause in *query expression*	NO	
F857	Top-level `FETCH FIRST` clause in *query expression*	NO	
F858	`FETCH FIRST` clause in subqueries	NO	
F859	Top-level `FETCH FIRST` clause in views	NO	
F860	`FETCH FIRST ROW`*count* in `FETCH FIRST` clause	NO	
F861	Top-level `RESULT OFFSET` clause in *query expression*	NO	
F862	`RESULT OFFSET` clause in subqueries	NO	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
F863	Nested `RESULT OFFSET` clause in *query expression*	NO	
F864	Top-level `RESULT OFFSET` clause in views	NO	
F865	`OFFSET ROW`*count* in `RESULT OFFSET` clause	NO	
S011	Distinct data types	NO	
S023	Basic structured types	NO	
S024	Enhanced structured types	NO	
S025	Final structured types	NO	
S026	Self-referencing structured types	NO	
S027	Create method by specific method name	NO	
S028	Permutable UDT options list	NO	
S041	Basic reference types	NO	
S043	Enhanced reference types	NO	
S051	Create table of type	NO	
S071	SQL paths in function and type name resolution	YES	
S091	Basic array support	NO	Greenplum has arrays, but is not fully standards compliant
S091-01	Arrays of built-in data types	NO	Partially compliant
S091-02	Arrays of distinct types	NO	
S091-03	Array expressions	NO	
S092	Arrays of user-defined types	NO	
S094	Arrays of reference types	NO	
S095	Array constructors by query	NO	
S096	Optional array bounds	NO	
S097	Array element assignment	NO	
S098	`ARRAY_AGG`	Partially	Supported: Using array_agg without a window specification; for example SELECT array_agg(x) FROM … SELECT array_agg (x order by y) FROM … Not supported: Using array_agg as an aggregate derived window function; for example SELECT array_agg(x) over (ORDER BY y) FROM … SELECT array_agg(x order by y) over (PARTITION BY z) FROM … SELECT array_agg(x order by y) over (ORDER BY z) FROM …
S111	`ONLY` in query expressions	YES	
S151	Type predicate	NO	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
S161	Subtype treatment	NO	
S162	Subtype treatment for references	NO	
S201	SQL-invoked routines on arrays	NO	Functions can be passed Greenplum array types
S202	SQL-invoked routines on multisets	NO	
S211	User-defined cast functions	YES	
S231	Structured type locators	NO	
S232	Array locators	NO	
S233	Multiset locators	NO	
S241	Transform functions	NO	
S242	Alter transform statement	NO	
S251	User-defined orderings	NO	
S261	Specific type method	NO	
S271	Basic multiset support	NO	
S272	Multisets of user-defined types	NO	
S274	Multisets of reference types	NO	
S275	Advanced multiset support	NO	
S281	Nested collection types	NO	
S291	Unique constraint on entire row	NO	
S301	Enhanced `UNNEST`	NO	
S401	Distinct types based on array types	NO	
S402	Distinct types based on distinct types	NO	
S403	`MAX_CARDINALITY`	NO	
S404	`TRIM_ARRAY`	NO	
T011	Timestamp in Information Schema	NO	
T021	`BINARY` and `VARBINARY` data types	NO	
T022	Advanced support for `BINARY` and `VARBINARY` data types	NO	
T023	Compound binary literal	NO	
T024	Spaces in binary literals	NO	
T031	`BOOLEAN` data type	YES	
T041	Basic `LOB` data type support	NO	
T042	Extended `LOB` data type support	NO	
T043	Multiplier T	NO	
T044	Multiplier P	NO	
T051	Row types	NO	
T052	`MAX` and `MIN` for row types	NO	
T053	Explicit aliases for all-fields reference	NO	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
T061	UCS support	NO	
T071	`BIGINT` data type	YES	
T101	Enhanced nullability determiniation	NO	
T111	Updatable joins, unions, and columns	NO	
T121	`WITH` (excluding `RECURSIVE`) in query expression	NO	
T122	`WITH` (excluding `RECURSIVE`) in subquery	NO	
T131	Recursive query	NO	
T132	Recursive query in subquery	NO	
T141	`SIMILAR` predicate	YES	
T151	`DISTINCT` predicate	YES	
T152	`DISTINCT` predicate with negation	NO	
T171	`LIKE` clause in table definition	YES	
T172	`AS` subquery clause in table definition	YES	
T173	Extended `LIKE` clause in table definition	YES	
T174	Identity columns	NO	
T175	Generated columns	NO	
T176	Sequence generator support	NO	
T177	Sequence generator support: simple restart option	NO	
T178	Identity columns: simple restart option	NO	
T191	Referential action `RESTRICT`	NO	
T201	Comparable data types for referential constraints	NO	
T211	Basic trigger capability	NO	
T211-01	Triggers activated on `UPDATE`, `INSERT`, or `DELETE` of one base table	NO	
T211-02	`BEFORE` triggers	NO	
T211-03	`AFTER` triggers	NO	
T211-04	`FOR EACH ROW` triggers	NO	
T211-05	Ability to specify a search condition that must be true before the trigger is invoked	NO	
T211-06	Support for run-time rules for the interaction of triggers and constraints	NO	
T211-07	`TRIGGER` privilege	YES	
T211-08	Multiple triggers for the same event are executed in the order in which they were created in the catalog	NO	Intentionally omitted
T212	Enhanced trigger capability	NO	
T213	`INSTEAD OF` triggers	NO	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
T231	Sensitive cursors	YES	
T241	`START TRANSACTION` statement	YES	
T251	`SET TRANSACTION` statement: `LOCAL` option	NO	
T261	Chained transactions	NO	
T271	Savepoints	YES	
T272	Enhanced savepoint management	NO	
T281	`SELECT` privilege with column granularity	NO	
T285	Enhanced derived column names	NO	
T301	Functional dependencies	NO	
T312	OVERLAY function	YES	
T321	Basic SQL-invoked routines	NO	Partial support
T321-01	User-defined functions with no overloading	YES	
T321-02	User-defined stored procedures with no overloading	NO	
T321-03	Function invocation	YES	
T321-04	CALL statement	NO	
T321-05	RETURN statement	NO	
T321-06	ROUTINES view	YES	
T321-07	PARAMETERS view	YES	
T322	Overloading of SQL-invoked functions and procedures	YES	
T323	Explicit security for external routines	YES	
T324	Explicit security for SQL routines	NO	
T325	Qualified SQL parameter references	NO	
T326	Table functions	NO	
T331	Basic roles	NO	
T332	Extended roles	NO	
T351	Bracketed SQL comments (`/*...*/` comments)	YES	
T431	Extended grouping capabilities	NO	
T432	Nested and concatenated `GROUPING SETS`	NO	
T433	Multiargument `GROUPING` function	NO	
T434	`GROUP BY DISTINCT`	NO	
T441	`ABS` and `MOD` functions	YES	
T461	Symmetric `BETWEEN` predicate	YES	
T471	Result sets return value	NO	
T491	`LATERAL` derived table	NO	
T501	Enhanced `EXISTS` predicate	NO	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
T511	Transaction counts	NO	
T541	Updatable table references	NO	
T561	Holdable locators	NO	
T571	Array-returning external SQL-invoked functions	NO	
T572	Multiset-returning external SQL-invoked functions	NO	
T581	Regular expression substring function	YES	
T591	`UNIQUE` constraints of possibly null columns	YES	
T601	Local cursor references	NO	
T611	Elementary OLAP operations	YES	
T612	Advanced OLAP operations	NO	Partially supported
T613	Sampling	NO	
T614	`NTILE` function	YES	
T615	`LEAD` and `LAG` functions	YES	
T616	Null treatment option for `LEAD` and `LAG` functions	NO	
T617	`FIRST_VALUE` and `LAST_VALUE` function	YES	
T618	`NTH_VALUE`	NO	Function exists in Greenplum but not all options are supported
T621	Enhanced numeric functions	YES	
T631	N predicate with one list element	NO	
T641	Multiple column assignment	NO	Some syntax variants supported
T651	SQL-schema statements in SQL routines	NO	
T652	SQL-dynamic statements in SQL routines	NO	
T653	SQL-schema statements in external routines	NO	
T654	SQL-dynamic statements in external routines	NO	
T655	Cyclically dependent routines	NO	
M001	Datalinks	NO	
M002	Datalinks via SQL/CLI	NO	
M003	Datalinks via Embedded SQL	NO	
M004	Foreign data support	NO	
M005	Foreign schema support	NO	
M006	GetSQLString routine	NO	
M007	TransmitRequest	NO	
M009	GetOpts and GetStatistics routines	NO	
M010	Foreign data wrapper support	NO	
M011	Datalinks via Ada	NO	
M012	Datalinks via C	NO	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
M013	Datalinks via COBOL	NO	
M014	Datalinks via Fortran	NO	
M015	Datalinks via M	NO	
M016	Datalinks via Pascal	NO	
M017	Datalinks via PL/I	NO	
M018	Foreign data wrapper interface routines in Ada	NO	
M019	Foreign data wrapper interface routines in C	NO	
M020	Foreign data wrapper interface routines in COBOL	NO	
M021	Foreign data wrapper interface routines in Fortran	NO	
M022	Foreign data wrapper interface routines in MUMPS	NO	
M023	Foreign data wrapper interface routines in Pascal	NO	
M024	Foreign data wrapper interface routines in PL/I	NO	
M030	SQL-server foreign data support	NO	
M031	Foreign data wrapper general routines	NO	
X010	XML type	YES	
X011	Arrays of XML type	YES	
X012	Multisets of XML type	NO	
X013	Distinct types of XML type	NO	
X014	Attributes of XML type	NO	
X015	Fields of XML type	NO	
X016	Persistent XML values	YES	
X020	XMLConcat	YES	xmlconcat2() supported
X025	XMLCast	NO	
X030	XMLDocument	NO	
X031	XMLElement	YES	
X032	XMLForest	YES	
X034	XMLAgg	YES	
X035	XMLAgg: ORDER BY option	YES	
X036	XMLComment	YES	
X037	XMLPI	YES	
X038	XMLText	NO	
X040	Basic table mapping	NO	
X041	Basic table mapping: nulls absent	NO	
X042	Basic table mapping: null as nil	NO	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
X043	Basic table mapping: table as forest	NO	
X044	Basic table mapping: table as element	NO	
X045	Basic table mapping: with target namespace	NO	
X046	Basic table mapping: data mapping	NO	
X047	Basic table mapping: metadata mapping	NO	
X048	Basic table mapping: base64 encoding of binary strings	NO	
X049	Basic table mapping: hex encoding of binary strings	NO	
X051	Advanced table mapping: nulls absent	NO	
X052	Advanced table mapping: null as nil	NO	
X053	Advanced table mapping: table as forest	NO	
X054	Advanced table mapping: table as element	NO	
X055	Advanced table mapping: target namespace	NO	
X056	Advanced table mapping: data mapping	NO	
X057	Advanced table mapping: metadata mapping	NO	
X058	Advanced table mapping: base64 encoding of binary strings	NO	
X059	Advanced table mapping: hex encoding of binary strings	NO	
X060	XMLParse: Character string input and CONTENT option	YES	
X061	XMLParse: Character string input and DOCUMENT option	YES	
X065	XMLParse: BLOB input and CONTENT option	NO	
X066	XMLParse: BLOB input and DOCUMENT option	NO	
X068	XMLSerialize: BOM	NO	
X069	XMLSerialize: INDENT	NO	
X070	XMLSerialize: Character string serialization and CONTENT option	YES	
X071	XMLSerialize: Character string serialization and DOCUMENT option	YES	
X072	XMLSerialize: Character string serialization	YES	
X073	XMLSerialize: BLOB serialization and CONTENT option	NO	
X074	XMLSerialize: BLOB serialization and DOCUMENT option	NO	
X075	XMLSerialize: BLOB serialization	NO	
X076	XMLSerialize: VERSION	NO	
X077	XMLSerialize: explicit ENCODING option	NO	
X078	XMLSerialize: explicit XML declaration	NO	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
X080	Namespaces in XML publishing	NO	
X081	Query-level XML namespace declarations	NO	
X082	XML namespace declarations in DML	NO	
X083	XML namespace declarations in DDL	NO	
X084	XML namespace declarations in compound statements	NO	
X085	Predefined namespace prefixes	NO	
X086	XML namespace declarations in XMLTable	NO	
X090	XML document predicate	NO	xml_is_well_formed_document() supported
X091	XML content predicate	NO	xml_is_well_formed_content() supported
X096	XMLExists	NO	xmlexists() supported
X100	Host language support for XML: CONTENT option	NO	
X101	Host language support for XML: DOCUMENT option	NO	
X110	Host language support for XML: VARCHAR mapping	NO	
X111	Host language support for XML: CLOB mapping	NO	
X112	Host language support for XML: BLOB mapping	NO	
X113	Host language support for XML: STRIP WHITESPACE option	YES	
X114	Host language support for XML: PRESERVE WHITESPACE option	YES	
X120	XML parameters in SQL routines	YES	
X121	XML parameters in external routines	YES	
X131	Query-level XMLBINARY clause	NO	
X132	XMLBINARY clause in DML	NO	
X133	XMLBINARY clause in DDL	NO	
X134	XMLBINARY clause in compound statements	NO	
X135	XMLBINARY clause in subqueries	NO	
X141	IS VALID predicate: data-driven case	NO	
X142	IS VALID predicate: ACCORDING TO clause	NO	
X143	IS VALID predicate: ELEMENT clause	NO	
X144	IS VALID predicate: schema location	NO	
X145	IS VALID predicate outside check constraints	NO	
X151	IS VALID predicate with DOCUMENT option	NO	
X152	IS VALID predicate with CONTENT option	NO	
X153	IS VALID predicate with SEQUENCE option	NO	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
X155	IS VALID predicate: NAMESPACE without ELEMENT clause	NO	
X157	IS VALID predicate: NO NAMESPACE with ELEMENT clause	NO	
X160	Basic Information Schema for registered XML Schemas	NO	
X161	Advanced Information Schema for registered XML Schemas	NO	
X170	XML null handling options	NO	
X171	NIL ON NO CONTENT option	NO	
X181	XML(DOCUMENT (UNTYPED)) type	NO	
X182	XML(DOCUMENT (ANY)) type	NO	
X190	XML(SEQUENCE) type	NO	
X191	XML(DOCUMENT (XMLSCHEMA)) type	NO	
X192	XML(CONTENT (XMLSCHEMA)) type	NO	
X200	XMLQuery	NO	
X201	XMLQuery: RETURNING CONTENT	NO	
X202	XMLQuery: RETURNING SEQUENCE	NO	
X203	XMLQuery: passing a context item	NO	
X204	XMLQuery: initializing an XQuery variable	NO	
X205	XMLQuery: EMPTY ON EMPTY option	NO	
X206	XMLQuery: NULL ON EMPTY option	NO	
X211	XML 1.1 support	NO	
X221	XML passing mechanism BY VALUE	NO	
X222	XML passing mechanism BY REF	NO	
X231	XML(CONTENT (UNTYPED)) type	NO	
X232	XML(CONTENT (ANY)) type	NO	
X241	RETURNING CONTENT in XML publishing	NO	
X242	RETURNING SEQUENCE in XML publishing	NO	
X251	Persistent XML values of XML(DOCUMENT (UNTYPED)) type	NO	
X252	Persistent XML values of XML(DOCUMENT (ANY)) type	NO	
X253	Persistent XML values of XML(CONTENT (UNTYPED)) type	NO	
X254	Persistent XML values of XML(CONTENT (ANY)) type	NO	
X255	Persistent XML values of XML(SEQUENCE) type	NO	
X256	Persistent XML values of XML(DOCUMENT (XMLSCHEMA)) type	NO	

Table 1. SQL 2008 Optional Feature Compliance Details

ID	Feature	Supported	Comments
X257	Persistent XML values of XML(CONTENT (XMLSCHEMA) type	NO	
X260	XML type: ELEMENT clause	NO	
X261	XML type: NAMESPACE without ELEMENT clause	NO	
X263	XML type: NO NAMESPACE with ELEMENT clause	NO	
X264	XML type: schema location	NO	
X271	XMLValidate: data-driven case	NO	
X272	XMLValidate: ACCORDING TO clause	NO	
X273	XMLValidate: ELEMENT clause	NO	
X274	XMLValidate: schema location	NO	
X281	XMLValidate: with DOCUMENT option	NO	
X282	XMLValidate with CONTENT option	NO	
X283	XMLValidate with SEQUENCE option	NO	
X284	XMLValidate NAMESPACE without ELEMENT clause	NO	
X286	XMLValidate: NO NAMESPACE with ELEMENT clause	NO	
X300	XMLTable	NO	
X301	XMLTable: derived column list option	NO	
X302	XMLTable: ordinality column option	NO	
X303	XMLTable: column default option	NO	
X304	XMLTable: passing a context item	NO	
X305	XMLTable: initializing an XQuery variable	NO	
X400	Name and identifier mapping	NO	

Parent topic: Greenplum Database Reference Guide

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

Parent topic: Greenplum Database Reference Guide

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...`)

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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**

Parent topic: Greenplum Database Reference Guide

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 `gpbackup` and `gprestore` utilities.

Parent topic: Greenplum Database Reference Guide

Checking for Tables that Need Routine Maintenance

The following views can help identify tables that need routine table maintenance (`VACUUM` and/or `ANALYZE`).

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.

Parent topic: The gp_toolkit Administrative Schema

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 1. gp_bloat_diag view

Column	Description
bdirelid	Table object id.
bdinspname	Schema name.
bdirelname	Table name.
bdirelpages	Actual number of pages on disk.
bdiexppages	Expected number of pages given the table data.
bdidiag	Bloat diagnostic message.

gp_stats_missing

This view shows tables that do not have statistics and therefore may require an `ANALYZE` be run on the table.

Table 2. gp_stats_missing view

Column	Description
smischema	Schema name.
smitable	Table name.
smisize	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.
smicols	Number of columns in the table.
smirecs	Number of rows in the 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

Parent topic: The gp_toolkit Administrative Schema

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 3. gp_locks_on_relation view

Column	Description
lorlocktype	Type of the lockable object: `relation`, `extend`, `page`, `tuple`, `transactionid`, `object`, `userlock`, `resource queue`, or `advisory`
lordatabase	Object ID of the database in which the object exists, zero if the object is a shared object.
lorrelname	The name of the relation.
lorrelation	The object ID of the relation.
lortransaction	The transaction ID that is affected by the lock.
lorpid	Process ID of the server process holding or awaiting this lock. NULL if the lock is held by a prepared transaction.
lormode	Name of the lock mode held or desired by this process.
lorgranted	Displays whether the lock is granted (true) or not granted (false).
lorcurrentquery	The current query in the session.

gp_locks_on_resqueue

Note: The `gp_locks_on_resqueue` view is valid only when resource queue-based resource management is active.

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 4. gp_locks_on_resqueue view

Column	Description
lorusename	Name of the user executing the session.
lorrsqname	The resource queue name.
lorlocktype	Type of the lockable object: `resource queue`
lorobjid	The ID of the locked transaction.
lortransaction	The ID of the transaction that is affected by the lock.
lorpid	The process ID of the transaction that is affected by the lock.
lormode	The name of the lock mode held or desired by this process.
lorgranted	Displays whether the lock is granted (true) or not granted (false).
lorwaiting	Displays whether or not the session is waiting.

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.

Parent topic: The gp_toolkit Administrative Schema

__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 5. __gp_aovisimap_compaction_info output table

Column	Description
content	Greenplum Database segment ID.
datafile	ID of the data file on the segment.
compaction_possible	The value is either `t` or `f`. The value `t` indicates that the data in data file be compacted when a VACUUM operation is performed. The server configuration parameter `gp_appendonly_compaction_threshold` affects this value.
hidden_tupcount	In the data file, the number of hidden (deleted or updated) rows.
total_tupcount	In the data file, the total number of rows.
percent_hidden	In the data file, the ratio (as a percentage) of hidden (deleted or updated) rows to total rows.

__gp_aoseg_name('table_name')

This function returns metadata information contained in the append-optimized table's on-disk segment file.

Table 6. __gp_aoseg_name output table

Column	Description
segno	The file segment number.
eof	The effective end of file for this file segment.
tupcount	The total number of tuples in the segment, including invisible tuples.
varblockcount	The total number of varblocks in the file segment.
eof_uncompressed	The end of file if the file segment were uncompressed.
modcount	The number of data modification operations.
state	The state of the file segment. Indicates if the segment is active or ready to be dropped after compaction.

__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 usefulfor 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 7. __gp_aoseg_history output table

Column	Description
gp_tid	The id of the tuple.
gp_xmin	The id of the earliest transaction.
gp_xmin_status	Status of the gp_xmin transaction.
gp_xmin_commit_	The commit distribution id of the gp_xmin transaction.
gp_xmax	The id of the latest transaction.
gp_xmax_status	The status of the latest transaction.
gp_xmax_commit_	The commit distribution id of the gp_xmax transaction.
gp_command_id	The id of the query command.
gp_infomask	A bitmap containing state information.
gp_update_tid	The ID of the newer tuple if the row is updated.
gp_visibility	The tuple visibility status.
segno	The number of the segment in the segment file.
tupcount	The number of tuples, including hidden tuples.
eof	The effective end of file for the segment.
eof_uncompressed	The end of file for the segment if data were uncompressed.
modcount	A count of data modifications.
state	The status of the segment.

__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 8. __gp_aocsseg(oid) output table

Column	Description
gp_tid	The table id.
segno	The segment number.
column_num	The column number.
physical_segno	The number of the segment in the segment file.
tupcount	The number of rows in the segment, excluding hidden tuples.
eof	The effective end of file for the segment.
eof_uncompressed	The end of file for the segment if the data were uncompressed.
modcount	A count of data modification operations for the segment.
state	The status of the segment.

__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 9. __gp_aocsseg_history output table

Column	Description
gp_tid	The oid of the tuple.
gp_xmin	The earliest transaction.
gp_xmin_status	The status of the gp_xmin transaction.
gp_xmin_	Text representation of gp_xmin.
gp_xmax	The latest transaction.
gp_xmax_status	The status of the gp_xmax transaction.
gp_xmax_	Text representation of gp_max.
gp_command_id	ID of the command operating on the tuple.
gp_infomask	A bitmap containing state information.
gp_update_tid	The ID of the newer tuple if the row is updated.
gp_visibility	The tuple visibility status.
segno	The segment number in the segment file.
column_num	The column number.
physical_segno	The segment containing data for the column.
tupcount	The total number of tuples in the segment.
eof	The effective end of file for the segment.
eof_uncompressed	The end of file for the segment if the data were uncompressed.
modcount	A count of the data modification operations.
state	The state of the segment.

__gp_aovisimap(oid)

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.

Column	Description
tid	The tuple id.
segno	The number of the segment file.
row_num	The row number of a row that has been deleted or updated.

__gp_aovisimap_hidden_info(oid)

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.

Column	Description
segno	The number of the segment file.
hidden_tupcount	The number of hidden tuples in the segment file.
total_tupcount	The total number of tuples in the segment file.

__gp_aovisimap_entry(oid)

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 10. __gp_aovisimap_entry output table

Column	Description
segno	Segment number of the visibility map entry.
first_row_num	The first row number of the entry.
hidden_tupcount	The number of hidden tuples in the entry.
bitmap	A text representation of the visibility bitmap.

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

Parent topic: The gp_toolkit Administrative Schema

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 11. gp_log_command_timings view

Column	Description
logsession	The session identifier (prefixed with "con").
logcmdcount	The command number within a session (prefixed with "cmd").
logdatabase	The name of the database.
loguser	The name of the database user.
logpid	The process id (prefixed with "p").
logtimemin	The time of the first log message for this command.
logtimemax	The time of the last log message for this command.
logduration	Statement duration from start to end time.

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 12. gp_log_database view

Column	Description
logtime	The timestamp of the log message.
loguser	The name of the database user.
logdatabase	The name of the database.
logpid	The associated process id (prefixed with "p").
logthread	The associated thread count (prefixed with "th").
loghost	The segment or master host name.
logport	The segment or master port.
logsessiontime	Time session connection was opened.
logtransaction	Global transaction id.
logsession	The session identifier (prefixed with "con").
logcmdcount	The command number within a session (prefixed with "cmd").
logsegment	The segment content identifier (prefixed with "seg" for primary or "mir" for mirror. The master always has a content id of -1).
logslice	The slice id (portion of the query plan being executed).
logdistxact	Distributed transaction id.
loglocalxact	Local transaction id.
logsubxact	Subtransaction id.
logseverity	LOG, ERROR, FATAL, PANIC, DEBUG1 or DEBUG2.
logstate	SQL state code associated with the log message.
logmessage	Log or error message text.
logdetail	Detail message text associated with an error message.
loghint	Hint message text associated with an error message.
logquery	The internally-generated query text.
logquerypos	The cursor index into the internally-generated query text.
logcontext	The context in which this message gets generated.
logdebug	Query string with full detail for debugging.
logcursorpos	The cursor index into the query string.
logfunction	The function in which this message is generated.
logfile	The log file in which this message is generated.
logline	The line in the log file in which this message is generated.
logstack	Full text of the stack trace associated with this message.

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 13. gp_log_master_concise view

Column	Description
logtime	The timestamp of the log message.
logdatabase	The name of the database.
logsession	The session identifier (prefixed with "con").
logcmdcount	The command number within a session (prefixed with "cmd").
logmessage	Log or error message text.

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 14. gp_log_system view

Column	Description
logtime	The timestamp of the log message.
loguser	The name of the database user.
logdatabase	The name of the database.
logpid	The associated process id (prefixed with "p").
logthread	The associated thread count (prefixed with "th").
loghost	The segment or master host name.
logport	The segment or master port.
logsessiontime	Time session connection was opened.
logtransaction	Global transaction id.
logsession	The session identifier (prefixed with "con").
logcmdcount	The command number within a session (prefixed with "cmd").
logsegment	The segment content identifier (prefixed with "seg" for primary or "mir" for mirror. The master always has a content id of -1).
logslice	The slice id (portion of the query plan being executed).
logdistxact	Distributed transaction id.
loglocalxact	Local transaction id.
logsubxact	Subtransaction id.
logseverity	LOG, ERROR, FATAL, PANIC, DEBUG1 or DEBUG2.
logstate	SQL state code associated with the log message.
logmessage	Log or error message text.
logdetail	Detail message text associated with an error message.
loghint	Hint message text associated with an error message.
logquery	The internally-generated query text.
logquerypos	The cursor index into the internally-generated query text.
logcontext	The context in which this message gets generated.
logdebug	Query string with full detail for debugging.

Table 14. gp_log_system view

Column	Description
logcursorpos	The cursor index into the query string.
logfunction	The function in which this message is generated.
logfile	The log file in which this message is generated.
logline	The line in the log file in which this message is generated.
logstack	Full text of the stack trace associated with this message.

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

Parent topic: The gp_toolkit Administrative Schema

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 15. gp_param_setting('parameter_name') function

Column	Description
paramsegment	The segment content id (only active segments are shown). The master content id is always -1.
paramname	The name of the parameter.
paramvalue	The value of the parameter.

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 16. gp_param_settings_seg_value_diffs view

Column	Description
psdname	The name of the parameter.
psdvalue	The value of the parameter.
psdcount	The number of segments that have this value.

Checking for Failed Segments

The gp_pgdatabase_invalid view can be used to check for down segments.

Parent topic: The gp_toolkit Administrative Schema

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 17. gp_pgdatabase_invalid view

Column	Description
pgdbidbid	The segment dbid. Every segment has a unique dbid.
pgdbiisprimary	Is the segment currently acting as the primary (active) segment? (t or f)
pgdbicontent	The content id of this segment. A primary and mirror will have the same content id.
pgdbivalid	Is this segment up and valid? (t or f)
pgdbidefinedprimary	Was this segment assigned the role of primary at system initialization time? (t or f)

Checking Resource Group Activity and Status

Note: The resource group activity and status views described in this section are valid only when resource group-based resource management is active.

Resource groups manage transactions to avoid exhausting system CPU and memory resources. Every database user is assigned a resource group. Greenplum Database evaluates every transaction submitted by a user against the limits configured for the user's resource group before running the transaction.

You can use the `gp_resgroup_config` view to check the configuration of each resource group. You can use the `gp_resgroup_status` view to display the current transaction status and resource usage of each resource group.

- gp_resgroup_config
- gp_resgroup_status

Parent topic: The gp_toolkit Administrative Schema

gp_resgroup_config

The `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 the current limit when the limit has been altered, but the new value could not be immediately applied.

This view is accessible to all users.

Table 18. gp_resgroup_config

Column	Description
groupid	The ID of the resource group.
groupname	The name of the resource group.
concurrency	The concurrency (`CONCURRENCY`) value specified for the resource group.
proposed_concurrency	The pending concurrency value for the resource group.
cpu_rate_limit	The CPU limit (`CPU_RATE_LIMIT`) value specified for the resource group, or -1.
memory_limit	The memory limit (`MEMORY_LIMIT`) value specified for the resource group.
proposed_memory_limit	The pending memory limit value for the resource group.

Table 18. gp_resgroup_config

Column	Description
memory_shared_quota	The shared memory quota (MEMORY_SHARED_QUOTA) value specified for the resource group.
proposed_memory_shared_quota	The pending shared memory quota value for the resource group.
memory_spill_ratio	The memory spill ratio (MEMORY_SPILL_RATIO) value specified for the resource group.
proposed_memory_spill_ratio	The pending memory spill ratio value for the resource group.
memory_auditor	The memory auditor for the resource group.
cpuset	The CPU cores reserved for the resource group, or -1.

gp_resgroup_status

The 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.

Note: Resource groups use the Linux control groups (cgroups) configured on the host systems. The cgroups are used to manage host system resources. When resource groups use cgroups that are as part of a nested set of cgroups, resource group limits are relative to the parent cgroup allotment. For information about nested cgroups and Greenplum Database resource group limits, see Understanding Role and Component Resource Groups.

This view is accessible to all users.

Table 19. gp_resgroup_status view

Column	Description
rsgname	The name of the resource group.
groupid	The ID of the resource group.
num_running	The number of transactions currently executing in the resource group.
num_queueing	The number of currently queued transactions for the resource group.
num_queued	The total number of queued transactions for the resource group since the Greenplum Database cluster was last started, excluding the num_queueing.
num_executed	The total number of executed transactions in the resource group since the Greenplum Database cluster was last started, excluding the num_running.
total_queue_duration	The total time any transaction was queued since the Greenplum Database cluster was last started.
cpu_usage	The real-time CPU usage of the resource group on each Greenplum Database segment's host.
memory_usage	The real-time memory usage of the resource group on each Greenplum Database segment's host.

The cpu_usage field is a JSON-formatted, key:value string that identifies, for each resource group, the per-segment CPU usage percentage. The key is segment id, the value is the percentage of CPU usage by the resource group on the segment host. The total CPU usage of all segments running on a segment host should not exceed the gp_resource_group_cpu_limit. Example cpu_usage column output:

```
{"-1":0.01, "0":0.31, "1":0.31}
```

In this example, segment `0` and segment `1` are running on the same host; their CPU usage is the same.

The `memory_usage` field is also a JSON-formatted, key:value string. The string contents differ depending upon the type of resource group. For each resource group that you assign to a role (default memory auditor `vmtracker`), this string identifies the used, available, granted, and proposed fixed and shared memory quota allocations on each segment. The key is segment id. The values are memory values displayed in MB units. The following example shows `memory_usage` column output for a single segment for a resource group that you assign to a role:

```
"0":{"used":0, "available":76, "quota_used":-1, "quota_available":60, "quota_granted":
60, "quota_proposed":60, "shared_used":0, "shared_available":16, "shared_granted":16,
"shared_proposed":16}
```

For each resource group that you assign to an external component, the `memory_usage` JSON-formatted string identifies the memory used and the memory limit on each segment. The following example shows `memory_usage` column output for an external component resource group for a single segment:

```
"1":{"used":11, "limit_granted":15}
```

Checking Resource Queue Activity and Status

Note: The resource queue activity and status views described in this section are valid only when resource queue-based resource management is active.

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

Parent topic: The gp_toolkit Administrative Schema

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 20. gp_resq_activity view

Column	Description
resqprocpid	Process ID assigned to this statement (on the master).
resqrole	User name.
resqoid	Resource queue object id.
resqname	Resource queue name.
resqstart	Time statement was issued to the system.
resqstatus	Status of statement: running, waiting or cancelled.

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 21. gp_resq_activity_by_queue Column

Column	Description
resqoid	Resource queue object id.
resqname	Resource queue name.
resqlast	Time of the last statement issued to the queue.
resqstatus	Status of last statement: running, waiting or cancelled.
resqtotal	Total statements in this queue.

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 22. gp_resq_priority_statement view

Column	Description
rqpdatname	The database name that the session is connected to.
rqpusename	The user who issued the statement.
rqpsession	The session ID.
rqpcommand	The number of the statement within this session (the command id and session id uniquely identify a statement).
rqppriority	The resource queue priority for this statement (MAX, HIGH, MEDIUM, LOW).
rqpweight	An integer value associated with the priority of this statement.
rqpquery	The query text of the statement.

gp_resq_role

This view shows the resource queues associated with a role. This view is accessible to all users.

Table 23. gp_resq_role view

Column	Description
rrrolname	Role (user) name.
rrrsqname	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*).

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 24. gp_resqueue_status view

Column	Description
`queueid`	The ID of the resource queue.
`rsqname`	The name of the resource queue.
`rsqcountlimit`	The active query threshold of the resource queue. A value of -1 means no limit.

Table 24. gp_resqueue_status view

Column	Description
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

Parent topic: The gp_toolkit Administrative Schema

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 25. gp_workfile_entries

Column	Type	References	Description
command_cnt	integer		Command ID of the query.
content	smallint		The content identifier for a segment instance.
current_query	text		Current query that the process is running.
datname	name		Greenplum database name.
directory	text		Path to the work file.
optype	text		The query operator type that created the work file.
procpid	integer		Process ID of the server process.
sess_id	integer		Session ID.
size	bigint		The size of the work file in bytes.

Table 25. gp_workfile_entries

Column	Type	References	Description
numfiles	bigint		The number of files created.
slice	smallint		The query plan slice. The portion of the query plan that is being executed.
state	text		The state of the query that created the work file.
usename	name		Role name.
workmem	integer		The amount of memory allocated to the operator in KB.

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 26. gp_workfile_usage_per_query

Column	Type	References	Description
command_cnt	integer		Command ID of the query.
content	smallint		The content identifier for a segment instance.
current_query	text		Current query that the process is running.
datname	name		Greenplum database name.
procpid	integer		Process ID of the server process.
sess_id	integer		Session ID.
size	bigint		The size of the work file in bytes.
numfiles	bigint		The number of files created.
state	text		The state of the query that created the work file.
usename	name		Role name.

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 27. gp_workfile_usage_per_segment

Column	Type	References	Description
content	smallint		The content identifier for a segment instance.
size	bigint		The total size of the work files on a segment.
numfiles	bigint		The number of files created.

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).

Parent topic: The gp_toolkit Administrative Schema

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 28. gp_roles_assigned view

Column	Description
raroleid	The role object ID. If this role has members (users), it is considered a *group* role.
rarolename	The role (user or group) name.
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:

```
SELECT relname as name, sotdsize as size, sotdtoastsize as
toast, sotdadditionalsize as other
FROM gp_size_of_table_disk as sotd, pg_class
WHERE sotd.sotdoid=pg_class.oid ORDER BY relname;
```

Parent topic: The gp_toolkit Administrative Schema

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 29. gp_size_of_all_table_indexes view

Column	Description
soatioid	The object ID of the table
soatisize	The total size of all table indexes in bytes
soatischemaname	The schema name
soatitablename	The table name

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 30. gp_size_of_database view

Column	Description
sodddatname	The name of the database
sodddatsize	The size of the database in bytes

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 31. gp_size_of_index view

Column	Description
soioid	The object ID of the index
soitableoid	The object ID of the table to which the index belongs
soisize	The size of the index in bytes
soiindexschemaname	The name of the index schema
soiindexname	The name of the index
soitableschemaname	The name of the table schema
soitablename	The name of the 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 32. gp_size_of_partition_and_indexes_disk view

Column	Description
sopaidparentoid	The object ID of the parent table
sopaidpartitionoid	The object ID of the partition table
sopaidpartitiontablesize	The partition table size in bytes
sopaidpartitionindexessize	The total size of all indexes on this partition
Sopaidparentschemaname	The name of the parent schema
Sopaidparenttablename	The name of the parent table
Sopaidpartitionschemaname	The name of the partition schema
sopaidpartitiontablename	The name of the partition 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 33. gp_size_of_schema_disk view

Column	Description
sosdnsp	The name of the schema
sosdschematablesize	The total size of tables in the schema in bytes
sosdschemaidxsize	The total size of indexes in the schema in bytes

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 34. gp_size_of_table_and_indexes_disk view

Column	Description
sotaidoid	The object ID of the parent table
sotaidtablesize	The disk size of the table
sotaididxsize	The total size of all indexes on the table
sotaidschemaname	The name of the schema
sotaidtablename	The name of the 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 35. gp_size_of_table_and_indexes_licensing view

Column	Description
sotailoid	The object ID of the table
sotailtablesizedisk	The total disk size of the table
sotailtablesizeuncompressed	If the table is a compressed append-optimized table, shows the uncompressed table size in bytes.
sotailindexessize	The total size of all indexes in the table
sotailschemaname	The schema name
sotailtablename	The table name

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 36. gp_size_of_table_disk view

Column	Description
sotdoid	The object ID of the table
sotdsize	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.
sotdtoastsize	The size of the TOAST table (oversized attribute storage), if there is one.

Table 36. gp_size_of_table_disk view

Column	Description
sotdadditionalsize	Reflects the segment and block directory table sizes for append-optimized (AO) tables.
sotdschemaname	The schema name
sotdtablename	The table name

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 37. gp_size_of_table_uncompressed view

Column	Description
sotuoid	The object ID of the table
sotusize	The uncomressed size of the table in bytes if it is a compressed AO table. Otherwise, the table size on disk.
sotuschemaname	The schema name
sotutablename	The table name

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 38. gp_disk_free external table

Column	Description
dfsegment	The content id of the segment (only active segments are shown)
dfhostname	The hostname of the segment host
dfdevice	The device name
dfspace	Free disk space in the segment file system in kilobytes

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

Parent topic: The gp_toolkit Administrative Schema

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 39. gp_skew_coefficients view

Column	Description
skcoid	The object id of the table.
skcnamespace	The namespace where the table is defined.

Table 39. gp_skew_coefficients view

Column	Description
skcrelname	The table name.
skccoeff	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.

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 40. gp_skew_idle_fractions view

Column	Description
sifoid	The object id of the table.
sifnamespace	The namespace where the table is defined.
sifrelname	The table name.
siffraction	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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 for query history.

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.

Note: `gpperfmon` does not support SQL `ALTER` commands. `ALTER` queries are not recorded in the `gpperfmon` query history tables.

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 `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_gperfmon_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 syslogger 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 `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 `gpsmon` processes continue to accumulate data until they receive a dump command from `gpmmon`.
4. The `gpsmon` processes respond to a dump command by sending their accumulated status data and log alerts to a listening `gpmmon` event handler thread.
5. The `gpmmon` event handler saves the metrics to `.txt` files in the `$MASTER_DATA_DIRECTORY/gpperfmon/data` directory on the master host.

At each `quantum` interval (15 seconds by default), `gpmmon` performs the following steps:

1. Sends a dump command to the `gpsmon` processes.
2. Gathers and converts the `.txt` files saved in `the $MASTER_DATA_DIRECTORY/gpperfmon/data` directory into `.dat` external data files for the `_now` and `_tail` external tables in the `gpperfmon` database.

   For example, disk space metrics are added to the `diskspace_now.dat` and `_diskspace_tail.dat` delimited text files. These text files are accessed via the `diskspace_now` and `_diskspace_tail` tables in the `gpperfmon` database.

At each `harvest_interval` (120 seconds by default), `gpmmon` performs the following steps for each

_tail file:

1. Renames the _tail file to a _stage file.
2. Creates a new _tail file.
3. Appends data from the _stage file into the _tail file.
4. Runs a SQL command to insert the data from the _tail external table into the corresponding _history table.

   For example, the contents of the _database_tail external table is inserted into the database_history regular (heap) table.
5. Deletes the _tail file after its contents have been loaded into the database table.
6. Gathers all of the gpdb-alert-*.csv files in the $MASTER_DATA_DIRECTORY/gpperfmon/logs directory (except the most recent, which the syslogger has open and is writing to) into a single file, alert_log_stage.
7. Loads the alert_log_stage file into the log_alert_history table in the gpperfmon database.
8. Truncates the alert_log_stage file.

The following topics describe the contents of the tables in the gpperfmon database.

- **database_***
- **diskspace_***
- **interface_stats_***
- **log_alert_***
- **queries_***
- **segment_***
- **socket_stats_***
- **system_***
- **dynamic_memory_info**
- **memory_info**

Parent topic: Greenplum Database Reference Guide

database_*

The database_* tables store query workload information for a Greenplum Database instance. There are three database tables, all having the same columns:

- database_now is an external table whose data files are stored in $MASTER_DATA_DIRECTORY/gpperfmon/data. Current query workload data is stored in database_now during the period between data collection from the data collection agents and automatic commitment to the database_history table.
- database_tail is an external table whose data files are stored in $MASTER_DATA_DIRECTORY/gpperfmon/data. This is a transitional table for query workload data that has been cleared from database_now but has not yet been committed to database_history. It typically only contains a few minutes worth of data.
- 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.

Column	Type	Description
ctime	timestamp	Time this row was created.
queries_total	int	The total number of queries in Greenplum Database at data collection time.
queries_running	int	The number of active queries running at data collection time.
queries_queued	int	The number of queries waiting in a resource group or resource queue, depending upon which resource management scheme is active, at data collection time.

Parent topic: The gpperfmon Database

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.

Column	Type	Description
ctime	timestamp(0) without time zone	Time of diskspace measurement.
hostname	varchar(64)	The hostname associated with the diskspace measurement.
Filesystem	text	Name of the filesystem for the diskspace measurement.
total_bytes	bigint	Total bytes in the file system.
bytes_used	bigint	Total bytes used in the file system.
bytes_available	bigint	Total bytes available in file system.

Parent topic: The gpperfmon Database

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.

Column	Type	Description
`interface_name`	string	Name of the interface. For example: eth0, eth1, lo.
`bytes_received`	bigint	Amount of data received in bytes.
`packets_received`	bigint	Number of packets received.
`receive_errors`	bigint	Number of errors encountered while data was being received.
`receive_drops`	bigint	Number of times packets were dropped while data was being received.
`receive_fifo_errors`	bigint	Number of times FIFO (first in first out) errors were encountered while data was being received.
`receive_frame_errors`	bigint	Number of frame errors while data was being received.
`receive_compressed_packets`	int	Number of packets received in compressed format.
`receive_multicast_packets`	int	Number of multicast packets received.
`bytes_transmitted`	bigint	Amount of data transmitted in bytes.
`packets_transmitted`	bigint	Amount of data transmitted in bytes.
`packets_transmitted`	bigint	Number of packets transmitted.
`transmit_errors`	bigint	Number of errors encountered during data transmission.
`transmit_drops`	bigint	Number of times packets were dropped during data transmission.
`transmit_fifo_errors`	bigint	Number of times fifo errors were encountered during data transmission.
`transmit_collision_errors`	bigint	Number of times collision errors were encountered during data transmission.
`transmit_carrier_errors`	bigint	Number of times carrier errors were encountered during data transmission.
`transmit_compressed_packets`	int	Number of packets transmitted in compressed format.

Parent topic: The gpperfmon Database

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.

Column	Type	Description
`logtime`	timestamp with time zone	Timestamp for this log
`loguser`	text	User of the query
`logdatabase`	text	The accessed database
`logpid`	text	Process id
`logthread`	text	Thread number
`loghost`	text	Host name or ip address
`logport`	text	Port number
`logsessiontime`	timestamp with time zone	Session timestamp
`logtransaction`	integer	Transaction id
`logsession`	text	Session id
`logcmdcount`	text	Command count
`logsegment`	text	Segment number
`logslice`	text	Slice number
`logdistxact`	text	Distributed transaction
`loglocalxact`	text	Local transaction
`logsubxact`	text	Subtransaction
`logseverity`	text	Log severity
`logstate`	text	State
`logmessage`	text	Log message
`logdetail`	text	Detailed message
`loghint`	text	Hint info
`logquery`	text	Executed query
`logquerypos`	text	Query position
`logcontext`	text	Context info
`logdebug`	text	Debug
`logcursorpos`	text	Cursor position
`logfunction`	text	Function info
`logfile`	text	Source code file
`logline`	text	Source code line
`logstack`	text	Stack trace

Parent topic: The gpperfmon Database

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	Type	Description
`ctime`	timestamp	Time this row was created.
`tmid`	int	A time identifier for a particular query. All records associated with the query will have the same `tmid`.
`ssid`	int	The session id as shown by `gp_session_id`. All records associated with the query will have the same `ssid`.
`ccnt`	int	The command number within this session as shown by `gp_command_count`. All records associated with the query will have the same `ccnt`.
`username`	varchar(64)	Greenplum role name that issued this query.
`db`	varchar(64)	Name of the database queried.
`cost`	int	Not implemented in this release.
`tsubmit`	timestamp	Time the query was submitted.
`tstart`	timestamp	Time the query was started.
`tfinish`	timestamp	Time the query finished.
`status`	varchar(64)	Status of the query -- `start`, `done`, or `abort`.
`rows_out`	bigint	Rows out for the query.
`cpu_elapsed`	bigint	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.
`cpu_currpct`	float	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.

Column	Type	Description
skew_cpu	float	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.
skew_rows	float	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.
query_hash	bigint	Not implemented in this release.
query_text	text	The SQL text of this query.
query_plan	text	Text of the query plan. Not implemented in this release.
application_name	varchar(64)	The name of the application.
rsqname	varchar(64)	If the resource queue-based resource management scheme is active, this column specifies the name of the resource queue.
rqppriority	varchar(64)	If the resource queue-based resource management scheme is active, this column specifies the priority of the query -- max, high, med, low, or min.

Parent topic: The gpperfmon Database

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 settings configured by the currently active resource management scheme (resource group-based or resource queue-based). See the *Greenplum Database Administrator Guide* for more information about resource management schemes.

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).

Column	Type	Description

ctime	timestamp(0) (without time zone)	The time the row was created.
dbid	int	The segment ID (dbid from gp_segment_configuration).
hostname	charvar(64)	The segment hostname.
dynamic_memory_used	bigint	The amount of dynamic memory (in bytes) allocated to query processes running on this segment.
dynamic_memory_available	bigint	The amount of additional dynamic memory (in bytes) that the segment can request before reaching the limit set by the currently active resource management scheme (resource group-based or resource queue-based).

See also the views memory_info and dynamic_memory_info for aggregated memory allocation and utilization by host.

Parent topic: The gpperfmon Database

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Column	Type	Description
total_sockets_used	int	Total sockets used in the system.
tcp_sockets_inuse	int	Number of TCP sockets in use.
tcp_sockets_orphan	int	Number of TCP sockets orphaned.
tcp_sockets_timewait	int	Number of TCP sockets in Time-Wait.
tcp_sockets_alloc	int	Number of TCP sockets allocated.
tcp_sockets_memusage_inbytes	int	Amount of memory consumed by TCP sockets.
udp_sockets_inuse	int	Number of UDP sockets in use.
udp_sockets_memusage_inbytes	int	Amount of memory consumed by UDP sockets.
raw_sockets_inuse	int	Number of RAW sockets in use.
frag_sockets_inuse	int	Number of FRAG sockets in use.
frag_sockets_memusage_inbytes	int	Amount of memory consumed by FRAG sockets.

Parent topic: The gpperfmon Database

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Column	Type	Description
`ctime`	timestamp	Time this row was created.
`hostname`	varchar(64)	Segment or master hostname associated with these system metrics.
`mem_total`	bigint	Total system memory in Bytes for this host.
`mem_used`	bigint	Used system memory in Bytes for this host.
`mem_actual_used`	bigint	Used actual memory in Bytes for this host (not including the memory reserved for cache and buffers).
`mem_actual_free`	bigint	Free actual memory in Bytes for this host (not including the memory reserved for cache and buffers).
`swap_total`	bigint	Total swap space in Bytes for this host.
`swap_used`	bigint	Used swap space in Bytes for this host.
`swap_page_in`	bigint	Number of swap pages in.
`swap_page_out`	bigint	Number of swap pages out.
`cpu_user`	float	CPU usage by the Greenplum system user.
`cpu_sys`	float	CPU usage for this host.
`cpu_idle`	float	Idle CPU capacity at metric collection time.
`load0`	float	CPU load average for the prior one-minute period.
`load1`	float	CPU load average for the prior five-minute period.
`load2`	float	CPU load average for the prior fifteen-minute period.
`quantum`	int	Interval between metric collection for this metric entry.
`disk_ro_rate`	bigint	Disk read operations per second.
`disk_wo_rate`	bigint	Disk write operations per second.
`disk_rb_rate`	bigint	Bytes per second for disk read operations.
`disk_wb_rate`	bigint	Bytes per second for disk write operations.
`net_rp_rate`	bigint	Packets per second on the system network for read operations.
`net_wp_rate`	bigint	Packets per second on the system network for write operations.

Column	Type	Description
net_rb_rate	bigint	Bytes per second on the system network for read operations.
net_wb_rate	bigint	Bytes per second on the system network for write operations.

Parent topic: The gpperfmon Database

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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, determined by the currently active resource management scheme (resource group-based or resource queue-based), is evaluated on a per-segment basis.

Column	Type	Description
ctime	timestamp(0) without time zone	Time this row was created in the segment_history table.
hostname	varchar(64)	Segment or master hostname associated with these system memory metrics.
dynamic_memory_used_mb	numeric	The amount of dynamic memory in MB allocated to query processes running on this segment.
dynamic_memory_available_mb	numeric	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.

Parent topic: The gpperfmon Database

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Column	Type	Description
ctime	timestamp(0) without time zone	Time this row was created in the segment_history table.
hostname	varchar(64)	Segment or master hostname associated with these system memory metrics.
mem_total_mb	numeric	Total system memory in MB for this segment host.
mem_used_mb	numeric	Total system memory used in MB for this segment host.
mem_actual_used_mb	numeric	Actual system memory used in MB for this segment host.
mem_actual_free_mb	numeric	Actual system memory free in MB for this segment host.
swap_total_mb	numeric	Total swap space in MB for this segment host.

Column	Type	Description
`swap_used_mb`	numeric	Total swap space used in MB for this segment host.
`dynamic_memory_used_mb`	numeric	The amount of dynamic memory in MB allocated to query processes running on this segment.
`dynamic_memory_available_mb`	numeric	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.

Parent topic: The gpperfmon Database

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 `citext` module adds a case-insensitive text data type. See citext Data Type.

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 1. Greenplum Database Built-in Data Types

Name	Alias	Size	Range	Description
bigint	int8	8 bytes	-9223372036854775808 to 9223372036854775807	large range integer
bigserial	serial8	8 bytes	1 to 9223372036854775807	large autoincrementing integer
bit [(n)]		*n* bits	bit string constant	fixed-length bit string
bit varying [(n)][1]	varbit	actual number of bits	bit string constant	variable-length bit string
boolean	bool	1 byte	true/false, t/f, yes/no, y/n, 1/0	logical boolean (true/false)
box		32 bytes	((x1,y1),(x2,y2))	rectangular box in the plane - not allowed in distribution key columns.
bytea[1]		1 byte + *binary string*	sequence of octets	variable-length binary string
character [(n)][1]	char [(n)]	1 byte + *n*	strings up to *n* characters in length	fixed-length, blank padded

Table 1. Greenplum Database Built-in Data Types

Name	Alias	Size	Range	Description
character varying [(n)][1]	varchar [(n)]	1 byte + *string size*	strings up to *n* characters in length	variable-length with limit
cidr		12 or 24 bytes		IPv4 and IPv6 networks
circle		24 bytes	<(x,y),r> (center and radius)	circle in the plane - not allowed in distribution key columns.
date		4 bytes	4713 BC - 294,277 AD	calendar date (year, month, day)
decimal [(p, s)][1]	numeric [(p, s)]	variable	no limit	user-specified precision, exact
double precision	float8 float	8 bytes	15 decimal digits precision	variable-precision, inexact
inet		12 or 24 bytes		IPv4 and IPv6 hosts and networks
integer	int, int4	4 bytes	-2147483648 to +2147483647	usual choice for integer
interval [(p)]		12 bytes	-178000000 years - 178000000 years	time span
json		1 byte + json size	json of any length	variable unlimited length
lseg		32 bytes	((x1,y1),(x2,y2))	line segment in the plane - not allowed in distribution key columns.
macaddr		6 bytes		MAC addresses
money		8 bytes	-92233720368547758.08 to +92233720368547758.07	currency amount
path[1]		16+16n bytes	[(x1,y1),...]	geometric path in the plane - not allowed in distribution key columns.
point		16 bytes	(x,y)	geometric point in the plane - not allowed in distribution key columns.
polygon		40+16n bytes	((x1,y1),...)	closed geometric path in the plane - not allowed in distribution key columns.
real	float4	4 bytes	6 decimal digits precision	variable-precision, inexact
serial	serial4	4 bytes	1 to 2147483647	autoincrementing integer
smallint	int2	2 bytes	-32768 to +32767	small range integer
text[1]		1 byte + *string size*	strings of any length	variable unlimited length
time [(p)] [without time zone]		8 bytes	00:00:00[.000000] - 24:00:00[.000000]	time of day only
time [(p)] with time zone	timetz	12 bytes	00:00:00+1359 - 24:00:00-1359	time of day only, with time zone
timestamp [(p)] [without time zone]		8 bytes	4713 BC - 294,277 AD	both date and time

Table 1. Greenplum Database Built-in Data Types

Name	Alias	Size	Range	Description
timestamp [(p)] with time zone	timestamptz	8 bytes	4713 BC - 294,277 AD	both date and time, with time zone
uuid		16 bytes		Universally Unique Identifiers according to RFC 4122, ISO/IEC 9834-8:2005
xml[1]		1 byte + *xml size*	xml of any length	variable unlimited length
txid_snapshot				user-level transaction ID snapshot

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*.

The pseudo-type *anytable* is a Greenplum Database type that specifies a table expression—an expression that computes a table. Greenplum Database allows this type only as an argument to a user-defined function. See Table Value Expressions for more about the *anytable* pseudo-type.

For more information about pseudo-types, see the Postgres documentation about Pseudo-Types.

Polymorphic Types

Four pseudo-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 subscripting 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 anyenum` 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.

Table Value Expressions

The *anytable* pseudo-type declares a function argument that is a table value expression. The notation for a table value expression is a `SELECT` statement enclosed in a `TABLE()` function. You can specify a distribution policy for the table by adding `SCATTER RANDOMLY`, or a `SCATTER BY` clause with a column list to specify the distribution key.

The `SELECT` statement is executed when the function is called and the result rows are distributed to segments so that each segment executes the function with a subset of the result table.

For example, this table expression selects three columns from a table named `customer` and sets the distribution key to the first column:

```
TABLE(SELECT cust_key, name, address FROM customer SCATTER BY 1)
```

The `SELECT` statement may include joins on multiple base tables, `WHERE` clauses, aggregates, and any other valid query syntax.

The *anytable* type is only permitted in functions implemented in the C or C++ languages. The body of the function can access the table using the Greenplum Database Server Programming Interface (SPI) or the Greenplum Partner Connector (GPPC) API.

The *anytable* type is used in some user-defined functions in the Pivotal GPText API. The following GPText example uses the `TABLE` function with the `SCATTER BY` clause in the GPText function `gptext.index()` to populate the index `mydb.mytest.articles` with data from the messages table:

```
SELECT * FROM gptext.index(TABLE(SELECT * FROM mytest.messages
          SCATTER BY distrib_id), 'mydb.mytest.messages');
```

For information about the function `gptext.index()`, see the Pivotal GPText documentation.

Parent topic: Greenplum Database Reference Guide

[1] 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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 1. Greenplum Database Character Sets [1]

Name	Description	Language	Server?	Bytes/Char	Aliases
BIG5	Big Five	Traditional Chinese	No	1-2	WIN950, Windows950
EUC_CN	Extended UNIX Code-CN	Simplified Chinese	Yes	1-3	
EUC_JP	Extended UNIX Code-JP	Japanese	Yes	1-3	
EUC_KR	Extended UNIX Code-KR	Korean	Yes	1-3	
EUC_TW	Extended UNIX Code-TW	Traditional Chinese, Taiwanese	Yes	1-3	
GB18030	National Standard	Chinese	No	1-2	
GBK	Extended National Standard	Simplified Chinese	No	1-2	WIN936, Windows936
ISO_8859_5	ISO 8859-5, ECMA 113	Latin/Cyrillic	Yes	1	
ISO_8859_6	ISO 8859-6, ECMA 114	Latin/Arabic	Yes	1	
ISO_8859_7	ISO 8859-7, ECMA 118	Latin/Greek	Yes	1	
ISO_8859_8	ISO 8859-8, ECMA 121	Latin/Hebrew	Yes	1	
JOHAB	JOHA	Korean (Hangul)	Yes	1-3	
KOI8	KOI8-R(U)	Cyrillic	Yes	1	KOI8R
LATIN1	ISO 8859-1, ECMA 94	Western European	Yes	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	Central European	Yes	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	South European	Yes	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	North European	Yes	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	Turkish	Yes	1	ISO88599
LATIN6	ISO 8859-10, ECMA 144	Nordic	Yes	1	ISO885910
LATIN7	ISO 8859-13	Baltic	Yes	1	ISO885913
LATIN8	ISO 8859-14	Celtic	Yes	1	ISO885914
LATIN9	ISO 8859-15	LATIN1 with Euro and accents	Yes	1	ISO885915
LATIN10	ISO 8859-16, ASRO SR 14111	Romanian	Yes	1	ISO885916

Table 1. Greenplum Database Character Sets [1]

Name	Description	Language	Server?	Bytes/Char	Aliases
MULE_INTERNAL	Mule internal code	Multilingual Emacs	Yes	1-4	
SJIS	Shift JIS	Japanese	No	1-2	Mskanji, ShiftJIS, WIN932, Windows932
SQL_ASCII	unspecified [2]	any	No	1	
UHC	Unified Hangul Code	Korean	No	1-2	WIN949, Windows949
UTF8	Unicode, 8-bit	all	Yes	1-4	Unicode
WIN866	Windows CP866	Cyrillic	Yes	1	ALT
WIN874	Windows CP874	Thai	Yes	1	
WIN1250	Windows CP1250	Central European	Yes	1	
WIN1251	Windows CP1251	Cyrillic	Yes	1	WIN
WIN1252	Windows CP1252	Western European	Yes	1	
WIN1253	Windows CP1253	Greek	Yes	1	
WIN1254	Windows CP1254	Turkish	Yes	1	
WIN1255	Windows CP1255	Hebrew	Yes	1	
WIN1256	Windows CP1256	Arabic	Yes	1	
WIN1257	Windows CP1257	Baltic	Yes	1	
WIN1258	Windows CP1258	Vietnamese	Yes	1	ABC, TCVN, TCVN5712, VSCII

Parent topic: Greenplum Database Reference Guide

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 2. Client/Server Character Set Conversions

Server Character Set	Available Client Character Sets
BIG5	not supported as a server encoding
EUC_CN	EUC_CN, MULE_INTERNAL, UTF8
EUC_JP	EUC_JP, MULE_INTERNAL, SJIS, UTF8
EUC_KR	EUC_KR, MULE_INTERNAL, UTF8
EUC_TW	EUC_TW, BIG5, MULE_INTERNAL, UTF8
GB18030	not supported as a server encoding
GBK	not supported as a server encoding
ISO_8859_5	ISO_8859_5, KOI8, MULE_INTERNAL, UTF8, WIN866, WIN1251
ISO_8859_6	ISO_8859_6, UTF8
ISO_8859_7	ISO_8859_7, UTF8
ISO_8859_8	ISO_8859_8, UTF8
JOHAB	JOHAB, UTF8
KOI8	KOI8, ISO_8859_5, MULE_INTERNAL, UTF8, WIN866, WIN1251
LATIN1	LATIN1, MULE_INTERNAL, UTF8
LATIN2	LATIN2, MULE_INTERNAL, UTF8, WIN1250
LATIN3	LATIN3, MULE_INTERNAL, UTF8
LATIN4	LATIN4, MULE_INTERNAL, UTF8
LATIN5	LATIN5, UTF8
LATIN6	LATIN6, UTF8
LATIN7	LATIN7, UTF8
LATIN8	LATIN8, UTF8
LATIN9	LATIN9, UTF8
LATIN10	LATIN10, UTF8
MULE_INTERNAL	MULE_INTERNAL, BIG5, EUC_CN, EUC_JP, EUC_KR, EUC_TW, ISO_8859_5, KOI8, LATIN1 to LATIN4, SJIS, WIN866, WIN1250, WIN1251
SJIS	not supported as a server encoding
SQL_ASCII	not supported as a server encoding
UHC	not supported as a server encoding
UTF8	all supported encodings
WIN866	WIN866
ISO_8859_5	KOI8, MULE_INTERNAL, UTF8, WIN1251
WIN874	WIN874, UTF8
WIN1250	WIN1250, LATIN2, MULE_INTERNAL, UTF8
WIN1251	WIN1251, ISO_8859_5, KOI8, MULE_INTERNAL, UTF8, WIN866
WIN1252	WIN1252, UTF8

Table 2. Client/Server Character Set Conversions

Server Character Set	Available Client Character Sets
WIN1253	WIN1253, UTF8
WIN1254	WIN1254, UTF8
WIN1255	WIN1255, UTF8
WIN1256	WIN1256, UTF8
WIN1257	WIN1257, UTF8
WIN1258	WIN1258, UTF8

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 `SETclient_encoding TO`.

  To set the client encoding, use the following SQL command:

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

  To query the current client encoding:

  ```
  => SHOW client_encoding;
  ```

  To return to the default encoding:

  ```
  => 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.

[1] Not all APIs support all the listed character sets. For example, the JDBC driver does not support MULE_INTERNAL, LATIN6, LATIN8, and LATIN10.

[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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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 1. Settable Classifications

Set Classification	Description
master or local	A *master* parameter only needs to be set in the `postgresql.conf` 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 `postgresql.conf` 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 requires a system restart for changes to take effect.

Table 1. Settable Classifications

Set Classification	Description
session or system	*Session* parameters can be changed on the fly within a database session, and can have a hierarchy of settings: at the system level (`postgresql.conf`), at the database level (`ALTER DATABASE...SET`), at the role level (`ALTER ROLE...SET`), or at the session level (`SET`). 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 *system* parameter can only be changed via the `postgresql.conf` file(s).
restart or reload	When changing parameter values in the postgresql.conf file(s), some require a *restart* of Greenplum Database for the change to take effect. Other parameter values can be refreshed by just reloading the server configuration file (using `gpstop -u`), and do not require stopping the system.
superuser	These session parameters can only be set by a database superuser. Regular database users cannot set this parameter.
read only	These parameters are not settable by database users or superusers. The current value of the parameter can be shown but not altered.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Greenplum Database Reference Guide

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 1. Functions in Greenplum Database

Function Type	Greenplum Support	Description	Comments
IMMUTABLE	Yes	Relies only on information directly in its argument list. Given the same argument values, always returns the same result.	
STABLE	Yes, in most cases	Within a single table scan, returns the same result for same argument values, but results change across SQL statements.	Results depend on database lookups or parameter values. `current_timestamp` family of functions is `STABLE`; values do not change within an execution.

Table 1. Functions in Greenplum Database

Function Type	Greenplum Support	Description	Comments
VOLATILE	Restricted	Function values can change within a single table scan. For example: `random()`,`timeofday()`.	Any function with side effects is volatile, even if its result is predictable. For example: `setval()`.

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:

```
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 2. Built-in functions and operators

Operator/Function Category	VOLATILE Functions	STABLE Functions	Restrictions
Logical Operators			
Comparison Operators			
Mathematical Functions and Operators	random setseed		
String Functions and Operators	*All built-in conversion functions*	convert pg_client_encoding	
Binary String Functions and Operators			
Bit String Functions and Operators			
Pattern Matching			
Data Type Formatting Functions		to_char to_timestamp	

Table 2. Built-in functions and operators

Operator/Function Category	VOLATILE Functions	STABLE Functions	Restrictions
Date/Time Functions and Operators	timeofday	age current_date current_time current_timestamp localtime localtimestamp now	
Enum Support Functions			
Geometric Functions and Operators			
Network Address Functions and Operators			
Sequence Manipulation Functions	nextval() setval()		
Conditional Expressions			
Array Functions and Operators		*All array functions*	
Aggregate Functions			
Subquery Expressions			
Row and Array Comparisons			
Set Returning Functions	generate_series		
System Information Functions		*All session information functions* *All access privilege inquiry functions* *All schema visibility inquiry functions* *All system catalog information functions* *All comment information functions* *All transaction ids and snapshots*	
System Administration Functions	set_config pg_cancel_backend pg_reload_conf pg_rotate_logfile pg_start_backup pg_stop_backup pg_size_pretty pg_ls_dir pg_read_file pg_stat_file	current_setting *All database object size functions*	**Note:** The function `pg_column_size` displays bytes required to store the value, possibly with TOAST compression.

Table 2. Built-in functions and operators

Operator/Function Category	VOLATILE Functions	STABLE Functions	Restrictions
XML Functions and function-like expressions		cursor_to_xml(cursor refcursor, count int, nulls boolean, tableforest boolean, targetns text) cursor_to_xmlschema(cursor refcursor, nulls boolean, tableforest boolean, targetns text) database_to_xml(nulls boolean, tableforest boolean, targetns text) database_to_xmlschema(nulls boolean, tableforest boolean, targetns text) database_to_xml_and_xmlschema(nulls boolean, tableforest boolean, targetns text) query_to_xml(query text, nulls boolean, tableforest boolean, targetns text) query_to_xmlschema(query text, nulls boolean, tableforest boolean, targetns text) query_to_xml_and_xmlschema(query text, nulls boolean, tableforest boolean, targetns text) schema_to_xml(schema name, nulls boolean, tableforest boolean, targetns text) schema_to_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text) schema_to_xml_and_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text) table_to_xml(tbl regclass, nulls boolean, tableforest boolean, targetns text) table_to_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text) 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 | CONTENT } value) xpath(text, xml) xpath(text, xml, text[]) xpath_exists(text, xml) xpath_exists(text, xml, text[]) xmlpi(name target [, content]) xmlroot(xml, version text | no value [,	

Table 2. Built-in functions and operators

Operator/Function Category	VOLATILE Functions	STABLE Functions	Restrictions
		standalone yes|no|no value]) xmlserialize ({ DOCUMENT | CONTENT } value AS type) xml(text) text(xml) xmlcomment(xml) xmlconcat2(xml, xml)	

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 3. *json* Operators

Operator	Right Operand Type	Description	Example	Example Result
`->`	`int`	Get JSON array element (indexed from zero).	`'[{"a":"foo"}, {"b":"bar"}, {"c":"baz"}]'::json->2`	`{"c":"baz"}`
`->`	`text`	Get JSON object field by key.	`'{"a": {"b":"foo"}}'::json->'a'`	`{"b":"foo"}`
`->>`	`int`	Get JSON array element as text.	`'[1,2,3]'::json->>2`	`3`
`->>`	`text`	Get JSON object field as text.	`'{"a":1,"b":2}'::json->>'b'`	`2`
`#>`	`text[]`	Get JSON object at specified path.	`'{"a": {"b":{"c": "foo"}}}'::json#>'{a,b}'`	`{"c": "foo"}`
`#>>`	`text[]`	Get JSON object at specified path as text.	`'{"a":[1,2,3],"b": [4,5,6]}'::json#>>'{a,2}'`	`3`

JSON Creation Functions

This table describes the functions that create `json` values.

Table 4. JSON Creation Functions

Function	Description	Example	Example Result

`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 5. JSON Processing Functions

Function	Return Type	Description	Example	Example Result

`json_each(json)`	`setof key text, value json`
	`setof key text, value jsonb`
		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`

Table 5. JSON Processing Functions

Function	Return Type	Description	Example	Example Result

`json_object_keys(json)`	`setof text`	Returns set of keys in the outermost JSON object.	`json_object_keys('{"f1":"abc","f2":{"f3":"a", "f4":"b"}}')`	`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`

`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 6. Window functions

Function	Return Type	Full Syntax	Description
`cume_dist()`	`double precision`	`CUME_DIST() OVER ( [PARTITION BY` expr `] ORDER BY` expr `)`	Calculates the cumulative distribution of a value in a group of values. Rows with equal values always evaluate to the same cumulative distribution value.

Table 6. Window functions

Function	Return Type	Full Syntax	Description
`dense_rank()`	`bigint`	`DENSE_RANK () OVER ( [PARTITION BY` expr `] ORDER BY` expr `)`	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.
`first_value(`*expr*`)`	same as input expr type	`FIRST_VALUE(` expr `) OVER ( [PARTITION BY` expr `] ORDER BY` expr `[ROWS|RANGE` frame_expr `] )`	Returns the first value in an ordered set of values.
`lag(`*expr* `[,`*offset*`] [,`*default*`])`	same as input *expr* type	`LAG(` *expr* `[,` *offset* `] [,` *default* `]) OVER ( [PARTITION BY` *expr* `] ORDER BY` *expr* `)`	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.
`last_value(`*expr*`)`	same as input *expr* type	`LAST_VALUE(`*expr*`) OVER ( [PARTITION BY` *expr*`] ORDER BY` *expr* `[ROWS|RANGE` *frame_expr*`] )`	Returns the last value in an ordered set of values.
`lead(`*expr* `[,`*offset*`] [,`*default*`])`	same as input *expr* type	`LEAD(`*expr* `[,`*offset*`] [,`*exprdefault*`]) OVER ( [PARTITION BY` *expr*`] ORDER BY` *expr* `)`	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.
`ntile(`*expr*`)`	`bigint`	`NTILE(`*expr*`) OVER ( [PARTITION BY` *expr*`] ORDER BY` *expr* `)`	Divides an ordered data set into a number of buckets (as defined by *expr*) and assigns a bucket number to each row.
`percent_rank()`	`double precision`	`PERCENT_RANK () OVER ( [PARTITION BY` *expr*`] ORDER BY` *expr* `)`	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).

Table 6. Window functions

Function	Return Type	Full Syntax	Description
`rank()`	`bigint`	`RANK () OVER ( [PARTITION BY expr] ORDER BY expr )`	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.
`row_number()`	`bigint`	`ROW_NUMBER () OVER ( [PARTITION BY expr] ORDER BY expr )`	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).

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 7. Advanced Aggregate Functions

Function	Return Type	Full Syntax	Description

`MEDIAN (expr)`	`timestamp, timestamptz, interval, float`	`MEDIAN (expression)`
		Example:
		`SELECT department_id, MEDIAN( salary)`
		`FROM employees`
		`GROUP BY department_id;`
			Can take a two-dimensional array as input. Treats such arrays as matrices.

`PERCENTILE_CONT (expr) WITHIN GROUP (ORDER BY expr [DESC/ASC])`	`timestamp, timestamptz, interval, float`	`PERCENTILE_CONT(percentage) WITHIN GROUP (ORDER BY expression)`
		Example:
		`SELECT department_id, PERCENTILE_CONT (0.5) WITHIN GROUP (ORDER BY salary DESC) "Median_cont";`
		`FROM employees GROUP BY depar tment_id;`
			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.

Table 7. Advanced Aggregate Functions

Function	Return Type	Full Syntax	Description

`PERCENTILE_DISC (expr) WITHIN GROUP (ORDER BY expr [DESC/ASC])`	`timestamp, timestamptz, interval, float`	`PERCENTILE_DISC(percentage) WITHIN GROUP (ORDER BY expression)`
		Example:
		`SELECT department_id, PERCENTILE_DISC (0.5) WITHIN GROUP (ORDER BY salary DESC) "Median_desc"; FROM employees GROUP BY depar tment_id;`
			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.

`sum(array[])`	`smallint[]int[], bigint[], float[]`	`sum(array[[1,2],[3,4]])`
		Example:
		`CREATE TABLE mymatrix (myvalu e int[]); INSERT INTO mymatrix VALUES ( array[[1,2],[3,4]]); INSERT INTO mymatrix VALUES ( array[[0,1],[1,0]]); SELECT sum(myvalue) FROM myma trix; sum --------------- {{1,3},{4,4}}`
			Performs matrix summation. Can take as input a two-dimensional array that is treated as a matrix.

`pivot_sum (label[], label, expr)`	`int[], bigint[], float[]`	`pivot_sum( array['A1','A2'], attr, value)`	A pivot aggregation using sum to resolve duplicate entries.
`unnest (array[])`	set of `anyelement`	`unnest( array['one', 'row', 'per', 'item'])`	Transforms a one dimensional array into rows. Returns a set of `anyelement`, a polymorphic pseudotype in PostgreSQL.

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`.

Parent topic: Greenplum Database Reference Guide

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:

```
%YAML 1.1
---
# 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.

```
# 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 indentifiers 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.

```
# 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 1.1
---
VERSION: 1.0.0.2
DATABASE: dbname
USER: db_username
HOST: master_hostname
PORT: master_port
```

```
DEFINE:
  - INPUT:
     NAME: input_name
     FILE:
      - hostname:/path/to/file
     GPFDIST:
       - hostname:port/file_pattern
     TABLE: table_name
     QUERY: SELECT_statement
     EXEC: command_string
     COLUMNS:
       - field_name data_type
     FORMAT: TEXT | CSV
     DELIMITER: delimiter_character
     ESCAPE: escape_character
     NULL: null_string
     QUOTE: csv_quote_character
     ERROR_LIMIT: integer
     ENCODING: database_encoding
```

```
  - OUTPUT:
     NAME: output_name
     FILE: file_path_on_client
     TABLE: table_name
     KEYS:
       - column_name
     MODE: REPLACE | APPEND
```

```
  - MAP:
     NAME: function_name
     FUNCTION: function_definition
     LANGUAGE: perl | python | c
     LIBRARY: /path/filename.so
     PARAMETERS:
       - nametype
     RETURNS:
       - nametype
     OPTIMIZE: STRICT IMMUTABLE
     MODE: SINGLE | MULTI
```

```
  - TRANSITION | CONSOLIDATE | FINALIZE:
```

```
    NAME: function_name
    FUNCTION: function_definition
    LANGUAGE: perl | python | c
    LIBRARY: /path/filename.so
    PARAMETERS:
      - nametype
    RETURNS:
      - nametype
    OPTIMIZE: STRICT IMMUTABLE
    MODE: SINGLE | MULTI
```

```
  - REDUCE:
    NAME: reduce_job_name
    TRANSITION: transition_function_name
    CONSOLIDATE: consolidate_function_name
    FINALIZE: finalize_function_name
    INITIALIZE: value
    KEYS:
      - key_name
```

```
  - TASK:
    NAME: task_name
    SOURCE: input_name
    MAP: map_function_name
    REDUCE: reduce_function_name
EXECUTE
```

```
  - RUN:
    SOURCE: input_or_task_name
    TARGET: output_name
    MAP: map_function_name
    REDUCE: reduce_function_name...
```

VERSION

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 `EXEC` input and the server configuration parameter Server Configuration Parameters is set to `on`.

COLUMNS

Optional. Columns are specified as: `column_name [data_type]`. If not specified, the default is `value text`. The 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 `TEXT`.

DELIMITER

Optional for FILE, GPFDIST and 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 FILE, GPFDIST and EXEC inputs. 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 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

Optional for FILE, GPFDIST and EXEC inputs. Specifies the string that represents a null value. The default is `\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 FILE, GPFDIST and 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 `'SQL_ASCII'`), an integer encoding number, or `DEFAULT` to use the default client encoding. See 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 `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 `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 KEYS.

KEYS

Optional for TABLE output. Specifies the column(s) to use as the Greenplum Database distribution key. If the EXECUTE task contains a REDUCE definition, then the `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 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 `APPEND` adds output data to an existing table (provided the table schema matches the output format) without removing any existing data. Declaring `REPLACE` will drop the table if it exists and then recreate it. Both `APPEND` and `REPLACE` will create a new table if one does not exist.

MAP

Required. Each `MAP` function takes data structured in (`key`, `value`) pairs, processes

each pair, and generates zero or more output (`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 REDUCE task, which is comprised of TRANSITION | CONSOLIDATE | FINALIZE functions. There is one predefined `MAP` function named `IDENTITY` that returns (`key`, `value`) pairs unchanged. Although (`key`, `value`) are the default parameters, you can specify other prototypes as needed.

TRANSITION | CONSOLIDATE | FINALIZE

`TRANSITION`, `CONSOLIDATE` and `FINALIZE` are all component pieces of REDUCE. A `TRANSITION` function is required. `CONSOLIDATE` and `FINALIZE` functions are optional. By default, all take `state` as the first of their input PARAMETERS, but other prototypes can be defined as well.

A `TRANSITION` function iterates through each value of a given key and accumulates values in a `state` variable. When the transition function is called on the first value of a key, the `state` is set to the value specified by INITALIZE of a 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 `state`.

If a `CONSOLIDATE` function is specified, `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 `state` value for a given key is redistributed, resulting in lower interconnect traffic and greater parallelism. `CONSOLIDATE` is handled like a `TRANSITION`, except that instead of `(state + value) => state`, it is `(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

```
# This example MapReduce job processes documents and looks for keywords in them.
# It takes two database tables as input:
#   - documents (doc_id integer, url text, data text)
# - keywords (keyword_id integer, keyword text)#
# The documents data is searched for occurences of keywords and returns results of
# url, data and keyword (a keyword can be multiple words, such as "high performance #
computing")
%YAML 1.1
---
VERSION:1.0.0.1
```

```
# Connect to Greenplum Database using this database and role
DATABASE:webdata
USER:jsmith

# Begin definition section
DEFINE:

 # Declare the input, which selects all columns and rows from the
 # 'documents' and 'keywords' tables.
- INPUT:
NAME:doc
TABLE:documents
- INPUT:
NAME:kw
TABLE:keywords
# Define the map functions to extract terms from documents and keyword
# This example simply splits on white space, but it would be possible
# to make use of a python library like nltk (the natural language toolkit)
# to perform more complex tokenization and word stemming.
 - MAP:
NAME:doc_map
 LANGUAGE:python
 FUNCTION:|
        i = 0            # the index of a word within the document
terms = {}# a hash of terms and their indexes within the document

# Lower-case and split the text string on space
for term in data.lower().split():
i = i + 1# increment i (the index)

        # Check for the term in the terms list:
        # if stem word already exists, append the i value to the array entry
        # corresponding to the term. This counts multiple occurrences of the word.
        # If stem word does not exist, add it to the dictionary with position i.
        # For example:
 # data: "a computer is a machine that manipulates data"
 # "a" [1, 4]
 # "computer" [2]
 # "machine" [3]
 # …
          if term in terms:
            terms[term] += ','+str(i)
          else:
            terms[term] = str(i)

# Return multiple lines for each document. Each line consists of
# the doc_id, a term and the positions in the data where the term appeared.
        # For example:
        #   (doc_id => 100, term => "a", [1,4]
        #   (doc_id => 100, term => "computer", [2]
        #    …
for term in terms:
yield([doc_id, term, terms[term]])
      OPTIMIZE:STRICT IMMUTABLE
      PARAMETERS:
- doc_id integer
        - data text
RETURNS:
- doc_id integer
        - term text
        - positions text

  # The map function for keywords is almost identical to the one for documents
 # but it also counts of the number of terms in the keyword.
 - MAP:
NAME:kw_map
LANGUAGE:python
FUNCTION:|
        i = 0
        terms = {}
```

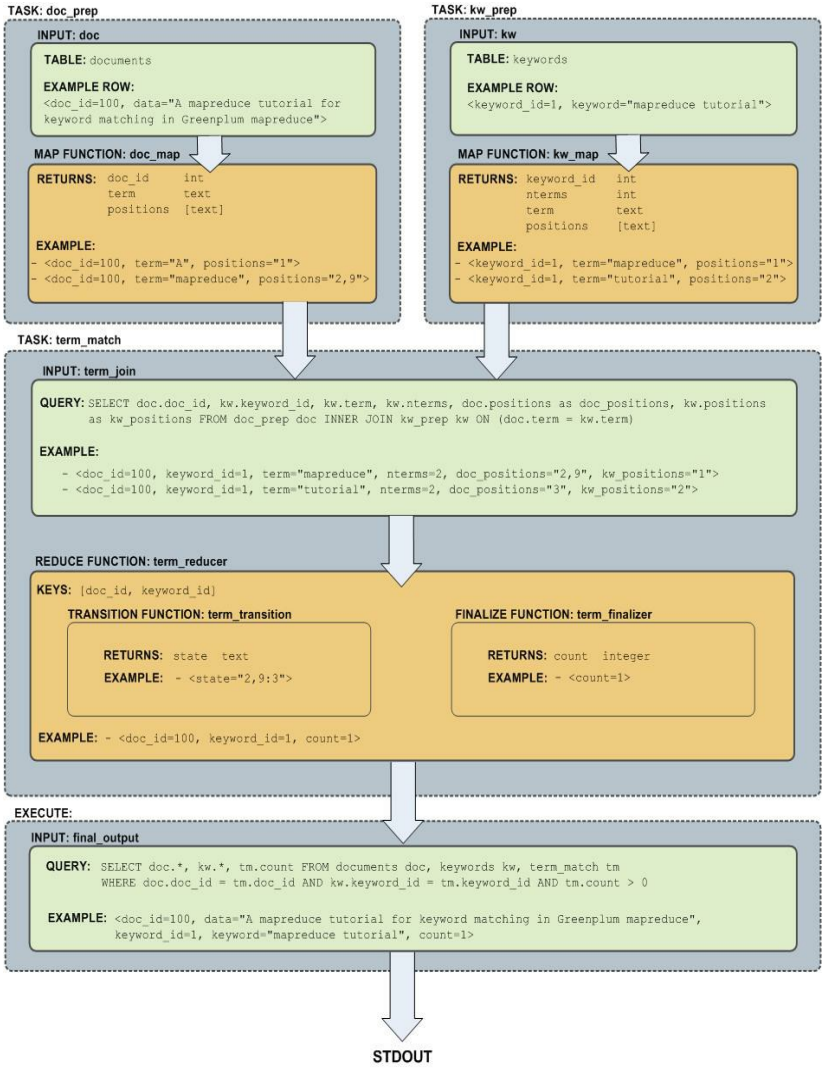
```
        for term in keyword.lower().split():
          i = i + 1
          if term in terms:
            terms[term] += ','+str(i)
          else:
            terms[term] = str(i)

 # output 4 values including i (the total count for term in terms):
yield([keyword_id, i, term, terms[term]])
      OPTIMIZE:STRICT IMMUTABLE
PARAMETERS:
- keyword_id integer
        - keyword text
RETURNS:
- keyword_id integer
        - nterms integer
        - term text
        - positions text

# A TASK is an object that defines an entire INPUT/MAP/REDUCE stage
# within a Greenplum MapReduce pipeline. It is like EXECUTION, but it is
# executed only when called as input to other processing stages.
# Identify a task called 'doc_prep' which takes in the 'doc' INPUT defined earlier
# and runs the 'doc_map' MAP function which returns doc_id, term, [term_position]
- TASK:
NAME:doc_prep
SOURCE:doc
MAP:doc_map

# Identify a task called 'kw_prep' which takes in the 'kw' INPUT defined earlier
# and runs the kw_map MAP function which returns kw_id, term, [term_position]
- TASK:
NAME:kw_prep
SOURCE:kw
MAP:kw_map

# One advantage of Greenplum MapReduce is that MapReduce tasks can be
# used as input to SQL operations and SQL can be used to process a MapReduce task.
# This INPUT defines a SQL query that joins the output of the 'doc_prep'
# TASK to that of the 'kw_prep' TASK. Matching terms are output to the 'candidate'
# list (any keyword that shares at least one term with the document).
- INPUT:
NAME: term_join
QUERY: |
        SELECT doc.doc_id, kw.keyword_id, kw.term, kw.nterms,
               doc.positions as doc_positions,
               kw.positions as kw_positions
          FROM doc_prep doc INNER JOIN kw_prep kw ON (doc.term = kw.term)

# In Greenplum MapReduce, a REDUCE function is comprised of one or more functions.
# A REDUCE has an initial 'state' variable defined for each grouping key. that is
# A TRANSITION function adjusts the state for every value in a key grouping.
# If present, an optional CONSOLIDATE function combines multiple
# 'state' variables. This allows the TRANSITION function to be executed locally at
# the segment-level and only redistribute the accumulated 'state' over
# the network. If present, an optional FINALIZE function can be used to perform
# final computation on a state and emit one or more rows of output from the state.
#
# This REDUCE function is called 'term_reducer' with a TRANSITION function
# called 'term_transition' and a FINALIZE function called 'term_finalizer'
- REDUCE:
NAME:term_reducer
TRANSITION:term_transition
FINALIZE:term_finalizer

- TRANSITION:
NAME:term_transition
 LANGUAGE:python
 PARAMETERS:
- state text
        - term text
```

```
        - nterms integer
        - doc_positions text
        - kw_positions text
FUNCTION: |

 # 'state' has an initial value of '' and is a colon delimited set
        # of keyword positions. keyword positions are comma delimited sets of
        # integers. For example, '1,3,2:4:'
        # If there is an existing state, split it into the set of keyword positions
 # otherwise construct a set of 'nterms' keyword positions - all empty
if state:
          kw_split = state.split(':')
        else:
          kw_split = []
          for i in range(0,nterms):
            kw_split.append('')

        # 'kw_positions' is a comma delimited field of integers indicating what
 # position a single term occurs within a given keyword.
        # Splitting based on ',' converts the string into a python list.
 # add doc_positions for the current term
for kw_p in kw_positions.split(','):
          kw_split[int(kw_p)-1] = doc_positions

        # This section takes each element in the 'kw_split' array and strings
        # them together placing a ':' in between each element from the array.
 # For example: for the keyword "computer software computer hardware",
        # the 'kw_split' array matched up to the document data of
        # "in the business of computer software software engineers"
        # would look like: ['5', '6,7', '5', '']
 # and the outstate would look like: 5:6,7:5:
outstate = kw_split[0]
        for s in kw_split[1:]:
          outstate = outstate + ':' + s
        return outstate

   - FINALIZE:
NAME: term_finalizer
LANGUAGE: python
RETURNS:
        - count integer
      MODE:MULTI
      FUNCTION:|
        if not state:
          return 0
        kw_split = state.split(':')

        # This function does the following:
 # 1) Splits 'kw_split' on ':'
        #    for example, 1,5,7:2,8 creates '1,5,7' and '2,8'
 # 2) For each group of positions in 'kw_split', splits the set on ','
        #    to create ['1','5','7'] from Set 0: 1,5,7 and
        #    eventually ['2', '8'] from Set 1: 2,8
 # 3) Checks for empty strings
 # 4) Adjusts the split sets by subtracting the position of the set
        #      in the 'kw_split' array
 # ['1','5','7'] - 0 from each element = ['1','5','7']
 # ['2', '8'] - 1 from each element = ['1', '7']
        # 5) Resulting arrays after subtracting the offset in step 4 are
 #    intersected and their overlaping 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:



A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Greenplum Database Reference Guide

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:

```
[ 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:

```
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 $$
DECLARE
    subtotal ALIAS FOR $1;
BEGIN
    RETURN subtotal * 0.06;
END;
$$ LANGUAGE plpgsql;
```

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:

```
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:

```
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:

```
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.

```
CREATE TABLE table1 (
  f1 text,
  f2 numeric,
  f3 integer
) distributed by (f1);
```

This `INSERT` command adds data to the table.

```
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`.

```
CREATE OR REPLACE FUNCTION t1_calc( name text) RETURNS integer
AS $$
DECLARE
    t1_row   table1%ROWTYPE;
    calc_int table1.f3%TYPE;
BEGIN
    SELECT * INTO t1_row FROM table1 WHERE table1.f1 = $1 ;
    calc_int = (t1_row.f2 * t1_row.f3)::integer ;
    RETURN calc_int ;
END;
$$ LANGUAGE plpgsql VOLATILE;
```

Note: 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 t1_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:

```
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:

```
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:

```
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:

```
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.

```
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

The Postgres documentation about PL/pgSQL is at
https://www.postgresql.org/docs/8.3/static/plpgsql.html

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*

For information about porting Oracle functions, see
https://www.postgresql.org/docs/8.3/static/plpgsql-porting.html. For information about installing and using the Oracle compatibility functions with Greenplum Database, see "Oracle Compatibility Functions" in the *Greenplum Database Utility Guide*.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Greenplum PostGIS Extension

This chapter contains the following information:

Parent topic: Greenplum Database Reference Guide

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`.

```
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.

```
export GDAL_DATA=$GPHOME/share/gdal
export POSTGIS_ENABLE_OUTDB_RASTERS=0
export POSTGIS_GDAL_ENABLED_DRIVERS=DISABLE_ALL
```

Enabling GDAL Raster Drivers

PostGIS uses GDAL raster drivers when processing raster data with commands such as `ST_AsJPEG()`. As the default, PostGIS disables all raster drivers. You enable raster drivers by setting the value of the `POSTGIS_GDAL_ENABLED_DRIVERS` environment variable in the `greenplum_path.sh` file on all Greenplum Database hosts.

To see the list of supported GDAL raster drivers for a Greenplum Database system, run the `raster2pgsql` utility with the `-G` option on the Greenplum Database master.

```
raster2pgsql -G
```

The command lists the driver long format name. The *GDAL Raster Formats* table at http://www.gdal.org/formats_list.html lists the long format names and the corresponding codes that you specify as the value of the environment variable. For example, the code for the long name Portable Network Graphics is `PNG`. This example `export` line enables four GDAL raster drivers.

```
export POSTGIS_GDAL_ENABLED_DRIVERS="GTiff PNG JPEG GIF"
```

The `gpstop -r` command restarts the Greenplum Database system to use the updated settings in the `greenplum_path.sh` file.

After you have updated the `greenplum_path.sh` file on all hosts, and have restarted the Greenplum Database system, you can display the enabled raster drivers with the `ST_GDALDrivers()` function. This `SELECT` command lists the enabled raster drivers.

```
SELECT short_name, long_name FROM ST_GDALDrivers();
```

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`.

```
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 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 1,5 5 5,7 7 5)', '3D Line' );
INSERT INTO geom_test ( gid, geom, name )
  VALUES ( 3, 'MULTIPOINT(3 4,8 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:

- box2d
- box3d
- geometry
- geography

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 information about PostGIS Raster data Management, queries, and applications http://postgis.net/docs/manual-2.1/using_raster_dataman.html

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 work around is to use the `AddGeometryColumn` function for PostGIS geometry. For example, a table with PostGIS geometry cannot be created with the following SQL command:

  ```
  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.sql` 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 `pslq` 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.

```
export GDAL_DATA=$GPHOME/share/gdal
export POSTGIS_ENABLE_OUTDB_RASTERS=0
export POSTGIS_GDAL_ENABLED_DRIVERS=DISABLE_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-2.1.5+pivotal.2
```

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.

```
export GDAL_DATA=$GPHOME/share/gdal
export POSTGIS_ENABLE_OUTDB_RASTERS=0
export POSTGIS_GDAL_ENABLED_DRIVERS=DISABLE_ALL
```

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Greenplum Database Reference Guide

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:

   ```
   $ gppkg -i plr-2.3.1-gp5-rhel6-x86_64.gppkg
   ```

4. Source the file `$GPHOME/greenplum_path.sh`.
5. Restart the database.

```
$ gpstop -r
```

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.

```
export LD_LIBRARY_PATH=
   $GPHOME/ext/R-3.3.3/lib:$LD_LIBRARY_PATH
```

Enabling PL/R Language Support

For each database that requires its use, register the PL/R language with the SQL command `CREATE EXTENSION`. Because PL/R is an untrusted language, only superusers can register PL/R with a database. For example, run this command as the `gpadmin` user to register the language with the database named `testdb`:

```
$ psql -d testdb -c 'CREATE EXTENSION plr;'
```

PL/R is registered as an untrusted language.

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 EXTENSION`. Because PL/R is an untrusted language, only superusers can remove support for the PL/R language from a database. For example, run this command as the `gpadmin` user to remove support for PL/R from the database named `testdb`:

```
$ psql -d testdb -c 'DROP EXTENSION plr;'
```

The default command fails if any existing objects (such as functions) depend on the language. Specify the `CASCADE` option to also drop all dependent objects, including functions that you created with PL/R.

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
```

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()`.

```
CREATE OR REPLACE FUNCTION r_norm(n integer, mean float8,
  std_dev float8) RETURNS float8[ ] AS
$$
  x<-rnorm(n,mean,std_dev)
  return(x)
$$
LANGUAGE 'plr';
```

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`:

  ```
  CREATE TYPE t1 AS ...
  ```

- Use this `TYPE` when defining your PL/R function

  ```
  ... 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:

```
--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, lme4, 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.ta
   r.gz
   ```

3. Use the `gpscp` utility and the `hosts_all` file to copy the `tar.gz` files to the same directory on all nodes of the Greenplum Database cluster. The `hosts_all` file contains a list of all the Greenplum Database segment hosts. You might require root access to do this.

   ```
   gpscp -f hosts_all Matrix_0.9996875-1.tar.gz =:/home/gpadmin
   ```

   ```
   gpscp -f /hosts_all arm_1.5-03.tar.gz =:/home/gpadmin
   ```

4. 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 use to install the package). For example, this `gpssh` command list the contents of the R library directory.

   ```
   gpssh -s -f all_hosts "ls $R_HOME/library"
   ```

   The `gpssh` option `-s` sources the `greenplum_path.sh` file before running commands on the remote hosts.

6. Test if the R package can be loaded.

   This function performs a simple test to if an R package can be loaded:

   ```
   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:

   ```
   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.

```
> 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()`:

```
> install.packages("package_name")
> install.packages("mypkg", dependencies = TRUE, type="source")
```

Load a package from the R command line.

```
> library(" package_name ")
```

An R package can be removed with `remove.packages`

```
> 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 -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-3.3.3/doc`

R Functions and Arguments

- See http://www.joeconway.com/doc/plr-funcs.html

Passing Data Values in R

- See http://www.joeconway.com/doc/plr-data.html

Aggregate Functions in R

- See http://www.joeconway.com/doc/plr-aggregate-funcs.html

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Greenplum Database Reference Guide

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.

PL/Python translates Python's `None` into the SQL `null` value

Data Type Mapping

The Greenplum to Python data type mapping follows.

Greenplum Primitive Type	Python Data Type

boolean[1]	bool
bytea	bytes
smallint, bigint, oid	int
real, double	float
numeric	decimal
other primitive types	string
SQL null value	None

[1] When the UDF return type is `boolean`, the Greenplum Database evaluates the return value for truth according to Python rules. That is, `0` and empty string are `false`, but notably `'f'` is `true`.

Example:

```
CREATE OR REPLACE FUNCTION pybool_func(a int) RETURNS boolean AS $$
# container: plc_python3_shared
    if (a > 0):
        return True
    else:
        return False
$$ LANGUAGE plcontainer;

SELECT pybool_func(-1);

 pybool_func
-------------
 f
(1 row)
```

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], [4, 5, 6]], <type 'list'>)
CONTEXT:  PL/Python function "return_multidim_py_type"
 return_multidim_py_array
--------------------------
 {{1,2,3},{4,5,6}}
(1 row)
```

PL/Python 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:

```
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();
                               composite_type_as_list
-------------------------------------------------------------------------------------
 {{"(first,1)","(second,1)"},{"(first,2)","(second,2)"},{"(first,3)","(second,3)"}}
(1 row)
```

Refer to the PostgreSQL Arrays, Lists documentation for additional information on PL/Python handling of arrays and composite types.

Set-Returning Functions

A Python function can return a set of scalar or composite types from any sequence type (for example: tuple, list, set).

In the following example, you create a composite type and a Python function that returns a `SETOF` of the composite type:

```
CREATE TYPE greeting AS (
  how text,
  who text
);

CREATE FUNCTION greet (how text)
  RETURNS SETOF greeting
```

```
AS $$
  # return tuple containing lists as composite types
  # all other combinations work also
  return ( {"how": how, "who": "World"}, {"how": how, "who": "Greenplum"} )
$$ LANGUAGE plpythonu;

select greet('hello');
       greet
-------------------
 (hello,World)
 (hello,Greenplum)
(2 rows)
```

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.

PL/Python also supports the `plpy.subtransaction()` function to help manage `plpy.execute` calls in an explicit subtransaction. See Explicit Subtransactions in the PostgreSQL documentation for additional information about `plpy.subtransaction()`.

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.

```
rv = 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:

```
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:

```
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:

```
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.

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:

```
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`
- `plpy.info`
- `plpy.notice`
- `plpy.warning`
- `plpy.error`
- `plpy.fatal`
- `plpy.debug`

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.

```
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
```

When running shell commands on remote hosts with `gpssh`, you can specify the `-s` option. When the option is specified, `gpssh` sources the `greenplum_path.sh` file before running commands on the remote hosts. For example, this command should display the Python installed with Greenplum Database on each host.

```
gpssh -f gpdb_hosts which python
```

If it does not, you can add the `-s` to source `greenplum_path.sh` on the remote hosts before running the command.

```
gpssh -s -f gpdb_hosts which python
```

To display the list of currently installed Python modules, run this command.

```
python -c "help('modules')"
```

Run `gpssh` in interactive mode to display Python modules on remote hosts. This example starts `gpssh` in interactive mode and lists the Python modules on the Greenplum Database host `sdw1`.

```
$ gpssh -s -h sdw1
=> python -c "help('modules')"
. . .
=> exit
$
```

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.

These sections describe installing and testing Python modules:

- Installing Python pip
- Installing Python Packages with pip

- Building and Installing Python Modules Locally
- Testing Installed Python Modules

Installing Python pip

The Python utility `pip` installs Python packages that contain Python modules and other resource files from versioned archive files.

Run this command to install `pip`.

```
python -m ensurepip --default-pip
```

The command runs the `ensurepip` module to bootstrap (install and configure) the `pip` utility from the local Python installation.

You can run this command to ensure the `pip`, `setuptools` and `wheel` projects are current. Current Python projects ensure that you can install Python packages from source distributions or pre-built distributions (wheels).

```
python -m pip install --upgrade pip setuptools wheel
```

You can use `gpssh` to run the commands on the Greenplum Database hosts. This example runs `gpssh` in interactive mode to install `pip` on the hosts listed in the file `gpdb_hosts`.

```
$ gpssh -s -f gpdb_hosts
=> python -m ensurepip --default-pip
[centos6-mdw1] Ignoring indexes: https://pypi.python.org/simple
[centos6-mdw1] Collecting setuptools
[centos6-mdw1] Collecting pip
[centos6-mdw1] Installing collected packages: setuptools, pip
[centos6-mdw1] Successfully installed pip-8.1.1 setuptools-20.10.1
[centos6-sdw1] Ignoring indexes: https://pypi.python.org/simple
[centos6-sdw1] Collecting setuptools
[centos6-sdw1] Collecting pip
[centos6-sdw1] Installing collected packages: setuptools, pip
[centos6-sdw1] Successfully installed pip-8.1.1 setuptools-20.10.1
=> exit
$
```

The `=>` is the inactive prompt for `gpssh`. The utility displays the output from each host. The `exit` command exits from `gpssh` interactive mode.

This `gpssh` command runs a single command on all hosts listed in the file `gpdb_hosts`.

```
gpssh -s -f gpdb_hosts python -m pip install --upgrade pip setuptools wheel
```

The utility displays the output from each host.

For more information about installing Python packages, see https://packaging.python.org/tutorials/installing-packages/.

Installing Python Packages with pip

After installing `pip`, you can install Python packages. This command installs the `numpy` and `scipy` packages.

```
python -m pip install --user numpy scipy
```

The `--user` option attempts to avoid conflicts when installing Python packages.

You can use `gpssh` to run the command on the Greenplum Database hosts.

For information about these and other Python packages, see References.

Building and Installing Python Modules Locally

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.

You can use the Greenplum Database utilities `gpssh` and `gpscp` to run commands on Greenplum Database hosts and to copy files to the hosts.

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`.

  ```
  gpssh -s -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 Greenplum Database might not have been restarted after installing the Python Package on the host system.

Examples

This PL/Python UDF returns the maximum of two integers:

```
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.

```
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.

```
SELECT ( pymax(123, 43));
column1
---------
     123
(1 row)
```

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.

```
CREATE TABLE sales (id int, year int, qtr int, day int, region text)
  DISTRIBUTED BY (id) ;

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'),
 (6, 2014, 2,6, 'asia'),
 (7, 2002, 3,7, 'usa') ;
```

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.

```
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.

```
SELECT mypytest(2) ;
```

This command deletes the UDF from the database.

```
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.

```
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/.

For information about PL/Python see the PostgreSQL documentation at https://www.postgresql.org/docs/8.3/static/plpython.html.

For information about Python Package Index (PyPI), see https://pypi.python.org/pypi.

These are some Python modules that can be installed:

- SciPy library provides user-friendly and efficient numerical routines such as routines for numerical integration and optimization. The SciPy site includes other similar Python libraries http://www.scipy.org/index.html.
- Natural Language Toolkit (nltk) is a platform for building Python programs to work with human language data. http://www.nltk.org/. For information about installing the toolkit see http://www.nltk.org/install.html.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

PL/Container Language

PL/Container enables users to run Greenplum procedural language functions inside a Docker container, to avoid security risks associated with executing Python or R code on Greenplum segment hosts. This topic covers information about the architecture, installation, and setup of PL/Container:

- About the PL/Container Language Extension
- Install PL/Container
- Upgrade PL/Container
- Uninstall PL/Container
- Docker References

For detailed information about using PL/Container, refer to the sections:

- PL/Container Resource Management
- PL/Container Functions

For reference documentation, see:

- plcontainer utility reference page

- plcontainer configuration file reference page

The PL/Container language extension is available as an open source module. For information about the module, see the README file in the GitHub repository at https://github.com/greenplum-db/plcontainer

Parent topic: Greenplum Database Reference Guide

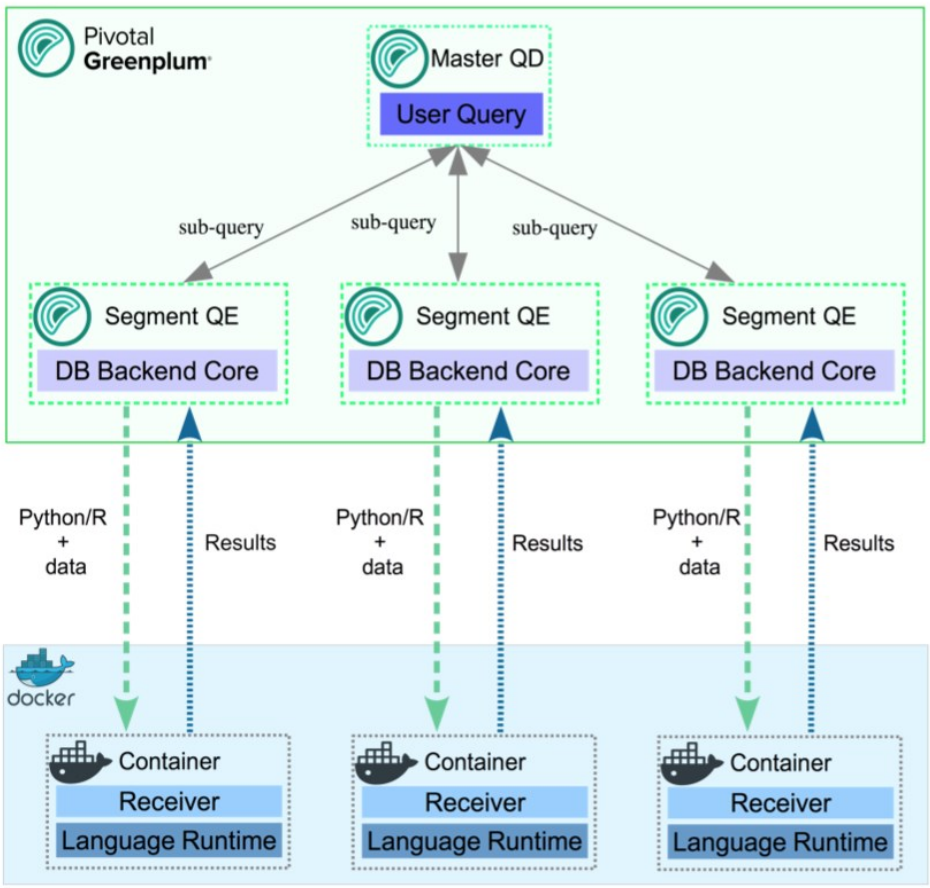
About the PL/Container Language Extension

The Greenplum Database PL/Container language extension allows users to create and run PL/Python or PL/R user-defined functions (UDFs) securely, inside a Docker container. Docker provides the ability to package and run an application in a loosely isolated environment called a container. For information about Docker, see the Docker web site.

Running UDFs inside the Docker container ensures that:

- The function execution process takes place in a separate environment and allows decoupling of the data processing. SQL operators such as "scan," "filter," and "project" are executed at the query executor (QE) side, and advanced data analysis is executed at the container side.
- User code cannot access the OS or the file system of the local host.
- User code cannot introduce any security risks.
- Functions cannot connect back to the Greenplum Database if the container is started with limited or no network access.

PL/Container Architecture



Example of the process flow:

Consider a query that selects table data using all available segments, and transforms the data using a PL/Container function. On the first call to a function in a segment container, the query executor on the master host starts the container on that segment host. It then contacts the running container to obtain the results. The container might respond with a Service Provider Interface (SPI) - a SQL query executed by the container to get some data back from the database - returning the result to the query executor.

A container running in standby mode waits on the socket and does not consume any CPU resources. PL/Container memory consumption depends on the amount of data cached in global dictionaries.

The container connection is closed by closing the Greenplum Database session that started the container, and the container shuts down.

Install PL/Container

Warning: PL/Container is compatible with Greenplum Database 5.2.0 and later. PL/Container has not been tested for compatibility with Greenplum Database 5.1.0 or 5.0.0.
This topic describes how to:

- Install Docker
- Install PL/Container
- Install the PL/Container Docker images
- Test the PL/Container installation

The following sections describe these tasks in detail.

Prerequisites

- PL/Container is supported on Pivotal Greenplum Database 5.2.x on Red Hat Enterprise Linux (RHEL) 7.x (or later) and CentOS 7.x (or later). PL/Container is not supported on RHEL/CentOS 6.x systems, because those platforms do not officially support Docker.
  Note: PL/Container 1.6.0 (introduced in Greenplum 5.26) and later versions support Docker images with Python 3 installed.
- The minimum Linux OS kernel version supported is 3.10. To verfiy your kernel release use:

  ```
  $ uname -r
  ```

Install Docker

To use PL/Container you need to install Docker on all Greenplum Database host systems. These are the minimum Docker versions that must be installed on Greenplum Database hosts (master, primary and all standby hosts):

- For PL/Container versions up to 1.5.0 - Docker 17.05
- For PL/Container 1.6.0 and later - Docker 19.03

These instructions show how to set up the Docker service on CentOS 7, but RHEL 7 is a similar process. These steps install the docker package and start the Docker service as a user with sudo privileges.

1. Ensure the user has sudo privileges or is root.
2. Install the dependencies required for Docker:

   ```
   sudo yum install -y yum-utils device-mapper-persistent-data lvm2
   ```

3. Add the Docker repo:

   ```
   sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/doc
   ker-ce.repo
   ```

4. Update yum cache:

```
sudo yum makecache fast
```

5. Install Docker:

```
sudo yum -y install docker-ce
```

6. Start Docker daemon:

```
sudo systemctl start docker
```

7. On each Greenplum Database host, the `gpadmin` user should be part of the docker group for the user to be able to manage Docker images and containers. Assign the Greenplum Database administrator `gpadmin` to the group `docker`:

```
sudo usermod -aG docker gpadmin
```

8. Exit the session and login again to update the privileges.
9. Configure Docker to start when the host system starts:

```
sudo systemctl enable docker.service
```

```
sudo systemctl start docker.service
```

10. Run a Docker command to test the Docker installation. This command lists the currently running Docker containers.

```
docker ps
```

11. After you install Docker on all Greenplum Database hosts, restart the Greenplum Database system to give Greenplum Database access to Docker.

```
gpstop -ra
```

For a list of observations while using Docker and PL/Container, see the Notes section. For a list of Docker reference documentation, see Docker References.

Docker Notes

- If a PL/Container Docker container exceeds the maximum allowed memory, it is terminated and an out of memory warning is displayed.
- PL/Container does not limit the Docker base device size, the size of the Docker container. In some cases, the Docker daemon controls the base device size. For example, if the Docker storage driver is devicemapper, the Docker daemon `--storage-opt` option flag `dm.basesize` controls the base device size. The default base device size for devicemapper is 10GB. The Docker command `docker info` displays Docker system information including the storage driver. The base device size is displayed in Docker 1.12 and later. For information about Docker storage drivers, see the Docker information Daemon storage-driver.

  When setting the Docker base device size, the size must be set on all Greenplum Database hosts.

- *Known issue*:

  Occasionally, when PL/Container is running in a high concurrency environment, the Docker daemon hangs with log entries that indicate a memory shortage. This can happen even when the system seems to have adequate free memory.

  The issue seems to be triggered by the aggressive virtual memory requirement of the Go language (golang) runtime that is used by PL/Container, and the Greenplum Database Linux

server kernel parameter setting for *overcommit_memory*. The parameter is set to 2 which does not allow memory overcommit.

A workaround that might help is to increase the amount of swap space and increase the Linux server kernel parameter overcommit_ratio. If the issue still occurs after the changes, there might be memory shortage. You should check free memory on the system and add more RAM if needed. You can also decrease the cluster load.

Install PL/Container

Note: The PL/Container version 1.1 and later extension is installed by `gppkg` as a Greenplum Database extension, while the PL/Container version 1.0 extension is installed as a Greenplum Database language. Refer to the documentation associated with your Greenplum Database version to install PL/Container.

Install the PL/Container language extension using the `gppkg` utility.

1. Download the "PL/Container for RHEL 7" package that applies to your Greenplum Database version, from the VMware Tanzu Network. PL/Container is listed under Greenplum Procedural Languages.
2. As `gpadmin`, copy the PL/Container language extension package to the master host.
3. Run the package installation command:

   ```
   $ gppkg -i plcontainer-1.6.0-rhel7-x86_64.gppkg
   ```

4. Source the file `$GPHOME/greenplum_path.sh`:

   ```
   $ source $GPHOME/greenplum_path.sh
   ```

5. Make sure Greenplum Database is up and running:

   ```
   $ gpstate -s
   ```

   If it is not, start it:

   ```
   $ gpstart -a
   ```

6. Restart Greenplum Database:

   ```
   $ gpstop -ra
   ```

7. Login into one of the available databases, for example:

   ```
   $ psql postgres
   ```

8. Register the PL/Container extension, which installs the `plcontainer` utility:

   ```
   postgres=# CREATE EXTENSION plcontainer;
   ```

   You need to register the utility separately on each database that might need the PL/Container functionality.

Install PL/Container Docker Images

Install the Docker images that PL/Container uses to create language-specific containers to run the UDFs.

- Download the `tar.gz` file that contains the Docker images from Pivotal Network.
  - `plcontainer-python3-images-<version>.tar.gz`
  - `plcontainer-python-images-<version>.tar.gz`

    - `plcontainer-r-images-<version>.tar.gz`

  Each image supports a different language or language version:

    - PL/Container for Python 3 - Docker image with Python 3.7 and the Python Data Science Module package installed.
      Note: PL/Container 1.6.0 and later supports Docker images with Python 3 installed.
    - PL/Container for Python 2 - Docker image with Python 2.7.12 and the Python Data Science Module package installed.
    - PL/Container for R - A Docker image with container with R-3.3.3 and the R Data Science Library package installed.

  The Data Science packages contain a set of Python modules or R functions and data sets related to data science. For information about the packages, see Python Data Science Module Package and R Data Science Library Package.

  If you require different images from the ones provided by Pivotal Greenplum, you can create custom Docker images, install the image, and add the image to the PL/Container configuration.

- Use the `plcontainer` utility command `image-add` to install the images on all Greenplum Database hosts where `-f` indicates the location of the downloaded files:

  ```
  # Install a Python 2 based Docker image
  $ plcontainer image-add -f /home/gpadmin/plcontainer-python-image-1.6.0.tar.gz

  # Install a Python 3 based Docker image
  $ plcontainer image-add -f /home/gpadmin/plcontainer-python3-image-1.6.0.tar.gz

  # Install an R based Docker image
  $ plcontainer image-add -f /home/gpadmin/plcontainer-r-image-1.6.0.tar.gz
  ```

  The utility displays progress information, similar to:

  ```
  20200127:21:54:43:004607 plcontainer:mdw:gpadmin-[INFO]:-Checking whether docke
  r is installed on all hosts...
  20200127:21:54:43:004607 plcontainer:mdw:gpadmin-[INFO]:-Distributing image fil
  e /home/gpadmin/plcontainer-python-images-1.6.0.tar to all hosts...
  20200127:21:54:55:004607 plcontainer:mdw:gpadmin-[INFO]:-Loading image on all h
  osts...
  20200127:21:55:37:004607 plcontainer:mdw:gpadmin-[INFO]:-Removing temporary ima
  ge files on all hosts...
  ```

  For more information on `image-add` options, visit the plcontainer reference page.

- To display the installed Docker images on the local host use:

  ```
  $ plcontainer image-list
  ```

REPOSITORY	TAG	IMAGE ID
pivotaldata/plcontainer_r_shared	devel	7427f920669d
pivotaldata/plcontainer_python_shared	devel	e36827eba53e
pivotaldata/plcontainer_python3_shared	devel	y32827ebe55b

- Add the image information to the PL/Container configuration file using `plcontainer runtime-add`, to allow PL/Container to associate containers with specified Docker images.

  Use the `-r` option to specify your own user defined runtime ID name, use the `-i` option to specify the Docker image, and the `-l` option to specify the Docker image language. When there are multiple versions of the same docker image, for example 1.0.0 or 1.2.0, specify the TAG version using ":" after the image name.

  ```
  # Add a Python 2 based runtime
  ```

```
$ plcontainer runtime-add -r plc_python_shared -i pivotaldata/plcontainer_pytho
n_shared:devel -l python

# Add a Python 3 based runtime that is supported with PL/Container 1.6.0
$ plcontainer runtime-add -r plc_python3_shared -i pivotaldata/plcontainer_pyth
on3_shared:devel -l python3

# Add an R based runtime
$ plcontainer runtime-add -r plc_r_shared -i pivotaldata/plcontainer_r_shared:d
evel -l r
```

The utility displays progress information as it updates the PL/Container configuration file on the Greenplum Database instances.

For details on other `runtime-add` options, see the plcontainer reference page.

- Optional: Use Greenplum Database resource groups to manage and limit the total CPU and memory resources of containers in PL/Container runtimes. In this example, the Python runtime will be used with a preconfigured resource group 16391:

  ```
  $ plcontainer runtime-add -r plc_python_shared -i pivotaldata/plcontainer_pytho
  n_shared:devel -l
   python -s resource_group_id=16391
  ```

  For more information about enabling, configuring, and using Greenplum Database resource groups with PL/Container, see PL/Container Resource Management .

You can now create a simple function to test your PL/Container installation.

Test the PL/Container Installation

List the names of the runtimes your created and added to the PL/Container XML file:

```
$ plcontainer runtime-show
```

The command shows a list of all installed runtimes:

```
PL/Container Runtime Configuration:
---------------------------------------------------------
  Runtime ID: plc_python_shared
  Linked Docker Image: pivotaldata/plcontainer_python_shared:devel
  Runtime Setting(s):
  Shared Directory:
  ---- Shared Directory From HOST '/usr/local/greenplum-db/./bin/plcontainer_clients'
to Container '/clientdir', access mode is 'ro'
---------------------------------------------------------
```

You can also view the PL/Container configuration information with the `plcontainer runtime-show -r <runtime_id>` command. You can view the PL/Container configuration XML file with the `plcontainer runtime-edit` command.

Use the `psql` utility and select an existing database:

```
$ psql postgres;
```

If the PL/Container extension is not registered with the selected database, first enable it using:

```
postgres=# CREATE EXTENSION plcontainer;
```

Create a simple function to test your installation; in the example, the function will use the runtime `plc_python_shared`:

```
postgres=# CREATE FUNCTION dummyPython() RETURNS text AS $$
# container: plc_python_shared
return 'hello from Python'
$$ LANGUAGE plcontainer;
```

Test the function using:

```
postgres=# SELECT dummyPython();
    dummypython
-------------------
 hello from Python
(1 row)
```

Similarly, to test the R runtime:

```
postgres=# CREATE FUNCTION dummyR() RETURNS text AS $$
# container: plc_r_shared
return ('hello from R')
$$ LANGUAGE plcontainer;
CREATE FUNCTION
postgres=# select dummyR();
    dummyr
--------------
 hello from R
(1 row)
```

For further details and examples about using PL/Container functions, see PL/Container Functions .

Upgrade PL/Container

Upgrading from PL/Container 1.1 or Later

Note: To upgrade PL/Container 1.0, you must uninstall the old version and install the new version. See Upgrading from PL/Container 1.0.

To upgrade from PL/Container 1.1 or higher, you save the current configuration, use `gppkg` to upgrade the PL/Container software, and then restore the configuration after upgrade. There is no need to update the Docker images when you upgrade PL/Container.

Note: Before you perform this upgrade procedure, ensure that you have migrated your existing PL/Container package from your previous Greenplum Database installation to your new Greenplum Database installation. Refer to the gppkg command for package installation and migration information.
Perform the following procedure to upgrade from PL/Container 1.1 or later:

1. Save the PL/Container configuration. For example, to save the configuration to a file named `plcontainer150-backup.xml` in the local directory:

   ```
   $ plcontainer runtime-backup -f plcontainer150-backup.xml
   ```

2. Use the Greenplum Database `gppkg` utility with the `-u` option to update the PL/Container language extension. For example, the following command updates the PL/Container language extension to version 1.6.0 on a Linux system:

   ```
   $ gppkg -u plcontainer-1.6.0-rhel7-x86_64.gppkg
   ```

3. Source the Greenplum Database environment file `$GPHOME/greenplum_path.sh`:

   ```
   $ source $GPHOME/greenplum_path.sh
   ```

4. Restore the PL/Container configuration. For example, this command restores the PL/Container configuration that you saved in a previous step:

   ```
   $ plcontainer runtime-restore -f plcontainer150-backup.xml
   ```

5. Restart Greenplum Database:

   ```
   $ gpstop -ra
   ```

6. You do not need to re-register the PL/Container extension in the databases in which you previously created the extension. However, you must register the PL/Container extension in each new database that will run PL/Container UDFs. For example, the following command registers PL/Container in a database named `mytest`:

   ```
   $ psql -d mytest -c 'CREATE EXTENSION plcontainer;'
   ```

   The command also creates PL/Container-specific functions and views.

Note: PL/Container 1.2 and later utilizes the new resource group capabilities introduced in Greenplum Database 5.8.0. If you downgrade to a Greenplum Database system that uses PL/Container 1.1. or earlier, you must use `plcontainer runtime-edit` to remove any `resource_group_id` settings from your PL/Container runtime configuration.

Upgrading from PL/Container 1.0

You cannot use the `gppkg -u` option to upgrade from PL/Container 1.0, because PL/Container 1.0 is installed as a Greenplum Database language rather than as an extension. To upgrade to from PL/Container 1.0, you must install uninstall version 1.0 and then install the new version. The Docker images and the PL/Container configuration do not change when upgrading PL/Container, only the PL/Container extension installation changes.

As part of the upgrade process, you must drop PL/Container from all databases that are configured with PL/Container.

Important: Dropping PL/Container from a database drops all PL/Container UDFs from the database, including user-created PL/Container UDFs. If the UDFs are required, ensure you can re-create the UDFs before dropping PL/Container. This `SELECT` command lists the names of and body of PL/Container UDFs in a database.

```
SELECT proname, prosrc FROM pg_proc WHERE prolang = (SELECT oid FROM pg_language WHERE
 lanname = 'plcontainer');
```

For information about the catalog tables, `pg_proc` and `pg_language`, see System Tables.

These steps upgrade from PL/Container 1.0 to PL/Container 1.1 or later. The steps save the PL/Container 1.0 configuration and restore the configuration for use with PL/Container 1.1 or later.

1. Save the PL/Container configuration. This example saves the configuration to `plcontainer10-backup.xml` in the local directory.

   ```
   $ plcontainer runtime-backup -f plcontainer10-backup.xml
   ```

2. Remove any `setting` elements that contain the `use_container_network` attribute from the configuration file. For example, this `setting` element must be removed from the configuration file.

   ```
   <setting use_container_network="yes"/>
   ```

3. Run the `plcontainer_uninstall.sql` script as the `gpadmin` user for each database that is configured with PL/Container. For example, this command drops the `plcontainer` language in the `mytest` database.

   ```
   $ psql -d mytest -f $GPHOME/share/postgresql/plcontainer/plcontainer_uninstall.
   sql
   ```

   The script drops the `plcontainer` language with the `CASCADE` clause that drops PL/Container-specific functions and views in the database.

4. Use the Greenplum Database `gppkg` utility with the `-r` option to uninstall the PL/Container language extension. This example uninstalls the PL/Container language extension on a Linux system.

```
$ gppkg -r plcontainer-1.0.0
```

5. Run the package installation command. This example installs the PL/Container 1.6.0 language extension on a Linux system.

   ```
   $ gppkg -i plcontainer-1.6.0-rhel7-x86_64.gppkg
   ```

6. Source the file `$GPHOME/greenplum_path.sh`.

   ```
   $ source $GPHOME/greenplum_path.sh
   ```

7. Update the PL/Container configuration. This command restores the saved configuration.

   ```
   $ plcontainer runtime-restore -f plcontainer10-backup.xml
   ```

8. Restart Greenplum Database.

   ```
   $ gpstop -ra
   ```

9. Register the new PL/Container extension as an extension for each database that uses PL/Container UDFs. This `psql` command runs a `CREATE EXTENSION` command to register PL/Container in the database `mytest`.

   ```
   $ psql -d mytest -c 'CREATE EXTENSION plcontainer;'
   ```

   The command registers PL/Container as an extension and creates PL/Container-specific functions and views.

After upgrading PL/Container for a database, re-install any user-created PL/Container UDFs that are required.

Uninstall PL/Container

To uninstall PL/Container, remove Docker containers and images, and then remove the PL/Container support from Greenplum Database.

When you remove support for PL/Container, the `plcontainer` user-defined functions that you created in the database will no longer work.

Uninstall Docker Containers and Images

On the Greenplum Database hosts, uninstall the Docker containers and images that are no longer required.

The `plcontainer image-list` command lists the Docker images that are installed on the local Greenplum Database host.

The `plcontainer image-delete` command deletes a specified Docker image from all Greenplum Database hosts.

Some Docker containers might exist on a host if the containers were not managed by PL/Container. You might need to remove the containers with Docker commands. These `docker` commands manage Docker containers and images on a local host.

- The command `docker ps -a` lists all containers on a host. The command `docker stop` stops a container.
- The command `docker images` lists the images on a host.
- The command `docker rmi` removes images.
- The command `docker rm` removes containers.

Remove PL/Container Support for a Database

To remove support for PL/Container, drop the extension from the database. Use the `psql` utility with `DROP EXTENION` command (using `-c`) to remove PL/Container from `mytest` database.

```
psql -d mytest -c 'DROP EXTENSION plcontainer CASCADE;'
```

The `CASCADE` keyword drops PL/Container-specific functions and views.

Uninstall the PL/Container Language Extension

If no databases have `plcontainer` as a registered language, uninstall the Greenplum Database PL/Container language extension with the `gppkg` utility.

1. Use the Greenplum Database `gppkg` utility with the `-r` option to uninstall the PL/Container language extension. This example uninstalls the PL/Container language extension on a Linux system:

   ```
   $ gppkg -r plcontainer-2.1.1
   ```

   You can run the `gppkg` utility with the options `-q --all` to list the installed extensions and their versions.

2. Reload `greenplum_path.sh`.

   ```
   $ source $GPHOME/greenplum_path.sh
   ```

3. Restart the database.

   ```
   $ gpstop -ra
   ```

Using PL/Container

This topic covers further details on:

- PL/Container Resource Management
- PL/Container Functions

PL/Container Resource Management

The Docker containers and the Greenplum Database servers share CPU and memory resources on the same hosts. In the default case, Greenplum Database is unaware of the resources consumed by running PL/Container instances. You can use Greenplum Database resource groups to control overall CPU and memory resource usage for running PL/Container instances.

PL/Container manages resource usage at two levels - the container level and the runtime level. You can control container-level CPU and memory resources with the `memory_mb` and `cpu_share` settings that you configure for the PL/Container runtime. `memory_mb` governs the memory resources available to each container instance. The `cpu_share` setting identifies the relative weighting of a container's CPU usage compared to other containers. See ../../utility_guide/admin_utilities/plcontainer-configuration.html for further details.

You cannot, by default, restrict the number of executing PL/Container container instances, nor can you restrict the total amount of memory or CPU resources that they consume.

Using Resource Groups to Manage PL/Container Resources

With PL/Container 1.2.0 and later, you can use Greenplum Database resource groups to manage and limit the total CPU and memory resources of containers in PL/Container runtimes. For more information about enabling, configuring, and using Greenplum Database resource groups, refer to Using Resource Groups in the *Greenplum Database Administrator Guide*.

Note: If you do not explicitly configure resource groups for a PL/Container runtime, its container instances are limited only by system resources. The containers may consume resources at the expense of the Greenplum Database server.

Resource groups for external components such as PL/Container use Linux control groups (cgroups) to manage component-level use of memory and CPU resources. When you manage PL/Container resources with resource groups, you configure both a memory limit and a CPU limit that Greenplum Database applies to all container instances that share the same PL/Container runtime configuration.

When you create a resource group to manage the resources of a PL/Container runtime, you must specify `MEMORY_AUDITOR=cgroup` and `CONCURRENCY=0` in addition to the required CPU and memory limits. For example, the following command creates a resource group named `plpy_run1_rg` for a PL/Container runtime:

```
CREATE RESOURCE GROUP plpy_run1_rg WITH (MEMORY_AUDITOR=cgroup, CONCURRENCY=0,
          CPU_RATE_LIMIT=10, MEMORY_LIMIT=10);
```

PL/Container does not use the `MEMORY_SHARED_QUOTA` and `MEMORY_SPILL_RATIO` resource group memory limits. Refer to the `CREATE RESOURCE GROUP` reference page for detailed information about this SQL command.

You can create one or more resource groups to manage your running PL/Container instances. After you create a resource group for PL/Container, you assign the resource group to one or more PL/Container runtimes. You make this assignment using the `groupid` of the resource group. You can determine the `groupid` for a given resource group name from the `gp_resgroup_config` `gp_toolkit` view. For example, the following query displays the `groupid` of a resource group named `plpy_run1_rg`:

```
SELECT groupname, groupid FROM gp_toolkit.gp_resgroup_config
 WHERE groupname='plpy_run1_rg';

 groupname   |  groupid
 --------------+----------
 plpy_run1_rg |   16391
 (1 row)
```

You assign a resource group to a PL/Container runtime configuration by specifying the `-s resource_group_id=`*`rg_groupid`* option to the `plcontainer runtime-add` (new runtime) or `plcontainer runtime-replace` (existing runtime) commands. For example, to assign the `plpy_run1_rg` resource group to a new PL/Container runtime named `python_run1`:

```
plcontainer runtime-add -r python_run1 -i pivotaldata/plcontainer_python_shared:devel
-l python -s resource_group_id=16391
```

You can also assign a resource group to a PL/Container runtime using the `plcontainer runtime-edit` command. For information about the `plcontainer` command, see ../../utility_guide/admin_utilities/plcontainer.html reference page.

After you assign a resource group to a PL/Container runtime, all container instances that share the same runtime configuration are subject to the memory limit and the CPU limit that you configured for the group. If you decrease the memory limit of a PL/Container resource group, queries executing in running containers in the group may fail with an out of memory error. If you drop a PL/Container resource group while there are running container instances, Greenplum Database kills the running containers.

Configuring Resource Groups for PL/Container

To use Greenplum Database resource groups to manage PL/Container resources, you must explicitly configure both resource groups and PL/Container.

Perform the following procedure to configure PL/Container to use Greenplum Database resource groups for CPU and memory resource management:

1. If you have not already configured and enabled resource groups in your Greenplum Database deployment, configure cgroups and enable Greenplum Database resource groups as described in Using Resource Groups in the *Greenplum Database Administrator Guide*. Note: If you have previously configured and enabled resource groups in your deployment, ensure that the Greenplum Database resource group `gpdb.conf` cgroups configuration file includes a `memory { }` block as described in the previous link.
2. Analyze the resource usage of your Greenplum Database deployment. Determine the percentage of resource group CPU and memory resources that you want to allocate to PL/Container Docker containers.
3. Determine how you want to distribute the total PL/Container CPU and memory resources that you identified in the step above among the PL/Container runtimes. Identify:
    - The number of PL/Container resource group(s) that you require.
    - The percentage of memory and CPU resources to allocate to each resource group.
    - The resource-group-to-PL/Container-runtime assignment(s).
4. Create the PL/Container resource groups that you identified in the step above. For example, suppose that you choose to allocate 25% of both memory and CPU Greenplum Database resources to PL/Container. If you further split these resources among 2 resource groups 60/40, the following SQL commands create the resource groups:

    ```
    CREATE RESOURCE GROUP plr_run1_rg WITH (MEMORY_AUDITOR=cgroup, CONCURRENCY=0,
                                    CPU_RATE_LIMIT=15, MEMORY_LIMIT=15);
     CREATE RESOURCE GROUP plpy_run1_rg WITH (MEMORY_AUDITOR=cgroup, CONCURRENCY=0,
                                    CPU_RATE_LIMIT=10, MEMORY_LIMIT=10);
    ```

5. Find and note the `groupid` associated with each resource group that you created. For example:

    ```
    SELECT groupname, groupid FROM gp_toolkit.gp_resgroup_config
    WHERE groupname IN ('plpy_run1_rg', 'plr_run1_rg');

    groupname   |  groupid
    --------------+----------
    plpy_run1_rg |   16391
    plr_run1_rg  |   16393
    (1 row)
    ```

6. Assign each resource group that you created to the desired PL/Container runtime configuration. If you have not yet created the runtime configuration, use the `plcontainer runtime-add` command. If the runtime already exists, use the `plcontainer runtime-replace` or `plcontainer runtime-edit` command to add the resource group assignment to the runtime configuration. For example:

    ```
    plcontainer runtime-add -r python_run1 -i pivotaldata/plcontainer_python_shared
    :devel -l python -s resource_group_id=16391
    plcontainer runtime-replace -r r_run1 -i pivotaldata/plcontainer_r_shared:devel
     -l r -s resource_group_id=16393
    ```

    For information about the `plcontainer` command, see ../../utility_guide/admin_utilities/plcontainer.html reference page.

Notes

PL/Container logging

When PL/Container logging is enabled, you can set the log level with the Greenplum Database server configuration parameter `log_min_messages`. The default log level is `warning`. The parameter controls the PL/Container log level and also controls the Greenplum Database log level.

- PL/Container logging is enabled or disabled for each runtime ID with the `setting` attribute

`use_container_logging`. The default is no logging.

- The PL/Container log information is the information from the UDF that is run in the Docker container. By default, the PL/Container log information is sent to a system service. On Red Hat 7 or CentOS 7 systems, the log information is sent to the `journald` service.
- The Greenplum Database log information is sent to log file on the Greenplum Database master.
- When testing or troubleshooting a PL/Container UDF, you can change the Greenplum Database log level with the `SET` command. You can set the parameter in the session before you run your PL/Container UDF. This example sets the log level to `debug1`.

  ```
  SET log_min_messages='debug1' ;
  ```

  Note: The parameter `log_min_messages` controls both the Greenplum Database and PL/Container logging, increasing the log level might affect Greenplum Database performance even if a PL/Container UDF is not running.

PL/Container Functions

When you enable PL/Container in a database of a Greenplum Database system, the language `plcontainer` is registered in that database. Specify `plcontainer` as a language in a UDF definition to create and run user-defined functions in the procedural languages supported by the PL/Container Docker images.

Limitations

Review the following limitations when creating and using PL/Container PL/Python and PL/R functions:

- Greenplum Database domains are not supported.
- Multi-dimensional arrays are not supported.
- Python and R call stack information is not displayed when debugging a UDF.
- The `plpy.execute()` methods `nrows()` and `status()` are not supported.
- The PL/Python function `plpy.SPIError()` is not supported.
- Executing the `SAVEPOINT` command with `plpy.execute()` is not supported.
- The `DO` command is not supported.
- Container flow control is not supported.
- Triggers are not supported.
- `OUT` parameters are not supported.
- The Python `dict` type cannot be returned from a PL/Python UDF. When returning the Python `dict` type from a UDF, you can convert the `dict` type to a Greenplum Database user-defined data type (UDT).

Using PL/Container functions

A UDF definition that uses PL/Container must have the these items.

- The first line of the UDF must be `# container: ID`
- The `LANGUAGE` attribute must be `plcontainer`

The *ID* is the name that PL/Container uses to identify a Docker image. When Greenplum Database executes a UDF on a host, the Docker image on the host is used to start a Docker container that runs the UDF. In the XML configuration file `plcontainer_configuration.xml`, there is a `runtime` XML element that contains a corresponding `id` XML element that specifies the Docker

container startup information. See ../../utility_guide/admin_utilities/plcontainer-configuration.html#topic_sk1_gdq_dw for information about how PL/Container maps the *ID* to a Docker image. See #function_examples for example UDF definitions.

The PL/Container configuration file is read only on the first invocation of a PL/Container function in each Greenplum Database session that runs PL/Container functions. You can force the configuration file to be re-read by performing a `SELECT` command on the view `plcontainer_refresh_config` during the session. For example, this `SELECT` command forces the configuration file to be read.

```
SELECT * FROM plcontainer_refresh_config;
```

Running the command executes a PL/Container function that updates the configuration on the master and segment instances and returns the status of the refresh.

```
 gp_segment_id | plcontainer_refresh_local_config
 --------------+---------------------------------
 1 | ok
 0 | ok
-1 | ok
(3 rows)
```

Also, you can show all the configurations in the session by performing a `SELECT` command on the view `plcontainer_show_config`. For example, this `SELECT` command returns the PL/Container configurations.

```
SELECT * FROM plcontainer_show_config;
```

Running the command executes a PL/Container function that displays configuration information from the master and segment instances. This is an example of the start and end of the view output.

```
INFO:  plcontainer: Container 'plc_py_test' configuration
 INFO:  plcontainer:     image = 'pivotaldata/plcontainer_python_shared:devel'
 INFO:  plcontainer:     memory_mb = '1024'
 INFO:  plcontainer:     use container network = 'no'
 INFO:  plcontainer:     use container logging  = 'no'
 INFO:  plcontainer:     shared directory from host '/usr/local/greenplum-db/./bin/plc
ontainer_clients' to container '/clientdir'
 INFO:  plcontainer:     access = readonly

 ...

 INFO:  plcontainer: Container 'plc_r_example' configuration  (seg0 slice3 192.168.180
.45:40000 pid=3304)
 INFO:  plcontainer:     image = 'pivotaldata/plcontainer_r_without_clients:0.2'  (seg
0 slice3 192.168.180.45:40000 pid=3304)
 INFO:  plcontainer:     memory_mb = '1024'  (seg0 slice3 192.168.180.45:40000 pid=330
4)
 INFO:  plcontainer:     use container network = 'no'  (seg0 slice3 192.168.180.45:400
00 pid=3304)
 INFO:  plcontainer:     use container logging  = 'yes'  (seg0 slice3 192.168.180.45:4
0000 pid=3304)
 INFO:  plcontainer:     shared directory from host '/usr/local/greenplum-db/bin/plcon
tainer_clients' to container '/clientdir'  (seg0 slice3 192.168.180.45:40000 pid=3304)
 INFO:  plcontainer:         access = readonly  (seg0 slice3 192.168.180.45:40000 pid=
3304)
 gp_segment_id | plcontainer_show_local_config
 --------------+-------------------------------
  0 | ok
 -1 | ok
  1 | ok
```

The PL/Container function `plcontainer_containers_summary()` displays information about the currently running Docker containers.

```
SELECT * FROM plcontainer_containers_summary();
```

If a normal (non-superuser) Greenplum Database user runs the function, the function displays information only for containers created by the user. If a Greenplum Database superuser runs the function, information for all containers created by Greenplum Database users is displayed. This is sample output when 2 containers are running.

```
 SEGMENT_ID |                           CONTAINER_ID                           |   UP_
TIME    |  OWNER   | MEMORY_USAGE(KB)
 ------------+------------------------------------------------------------------+-----
---------+---------+-----------------
 1          | 693a6cb691f1d2881ec0160a44dae2547a0d5b799875d4ec106c09c97da422ea | Up 8
seconds | gpadmin | 12940
 1          | bc9a0c04019c266f6d8269ffe35769d118bfb96ec634549b2b1bd2401ea20158 | Up 2
minutes | gpadmin | 13628
 (2 rows)
```

When Greenplum Database executes a PL/Container UDF, Query Executer (QE) processes start Docker containers and reuse them as needed. After a certain amount of idle time, a QE process quits and destroys its Docker containers. You can control the amount of idle time with the Greenplum Database server configuration parameter `gp_vmem_idle_resource_timeout`. Controlling the idle time might help with Docker container reuse and avoid the overhead of creating and starting a Docker container.

Warning: Changing `gp_vmem_idle_resource_timeout` value, might affect performance due to resource issues. The parameter also controls the freeing of Greenplum Database resources other than Docker containers.

Examples

The values in the `# container` lines of the examples, `plc_python_shared` and `plc_r_shared`, are the `id` XML elements defined in the `plcontainer_config.xml` file. The `id` element is mapped to the `image` element that specifies the Docker image to be started. If you configured PL/Container with a different ID, change the value of the `# container` line. For information about configuring PL/Container and viewing the configuration settings, see ../../utility_guide/admin_utilities/plcontainer-configuration.html.

This is an example of PL/Python function that runs using the `plc_python_shared` container that contains Python 2:

```
CREATE OR REPLACE FUNCTION pylog100() RETURNS double precision AS $$
 # container: plc_python_shared
 import math
 return math.log10(100)
 $$ LANGUAGE plcontainer;
```

This is an example of a similar function using the `plc_r_shared` container:

```
CREATE OR REPLACE FUNCTION rlog100() RETURNS text AS $$
# container: plc_r_shared
return(log10(100))
$$ LANGUAGE plcontainer;
```

If the `# container` line in a UDF specifies an ID that is not in the PL/Container configuration file, Greenplum Database returns an error when you try to execute the UDF.

About PL/Container Running PL/Python

In the Python language container, the module `plpy` is implemented. The module contains these methods:

- `plpy.execute(stmt)` - Executes the query string `stmt` and returns query result in a list of dictionary objects. To be able to access the result fields ensure your query returns named fields.

- `plpy.prepare(stmt[, argtypes])` - 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.
- `plpy.execute(plan[, argtypes])` - Executes a prepared plan.
- `plpy.debug(msg)` - Sends a DEBUG2 message to the Greenplum Database log.
- `plpy.log(msg)` - Sends a LOG message to the Greenplum Database log.
- `plpy.info(msg)` - Sends an INFO message to the Greenplum Database log.
- `plpy.notice(msg)` - Sends a NOTICE message to the Greenplum Database log.
- `plpy.warning(msg)` - Sends a WARNING message to the Greenplum Database log.
- `plpy.error(msg)` - Sends an ERROR message to the Greenplum Database log. An ERROR message raised in Greenplum Database causes the query execution process to stop and the transaction to rollback.
- `plpy.fatal(msg)` - Sends a FATAL message to the Greenplum Database log. A FATAL message causes Greenplum Database session to be closed and transaction to be rolled back.
- `plpy.subtransaction()` - Manages `plpy.execute` calls in an explicit subtransaction. See Explicit Subtransactions in the PostgreSQL documentation for additional information about `plpy.subtransaction()`.

If an error of level `ERROR` or `FATAL` is raised in a nested Python function call, the message includes the list of enclosing functions.

The Python language container supports these string quoting functions that are useful when constructing ad-hoc queries.

- `plpy.quote_literal(string)` - Returns the string quoted to be used as a string literal in an SQL statement string. Embedded single-quotes and backslashes are properly doubled. `quote_literal()` returns null on null input (empty input). If the argument might be null, `quote_nullable()` might be more appropriate.
- `plpy.quote_nullable(string)` - Returns the string quoted to be used as a string literal in an SQL statement string. If the argument is null, returns `NULL`. Embedded single-quotes and backslashes are properly doubled.
- `plpy.quote_ident(string)` - Returns the string quoted to be used as an identifier in an SQL statement string. Quotes are added only if necessary (for example, if the string contains non-identifier characters or would be case-folded). Embedded quotes are properly doubled.

When returning text from a PL/Python function, PL/Container converts a Python unicode object to text in the database encoding. If the conversion cannot be performed, an error is returned.

PL/Container does not support this Greenplum Database PL/Python feature:

- Multi-dimensional arrays.

Also, the Python module has two global dictionary objects that retain the data between function calls. They are named GD and SD. GD is used to share the data between all the function running within the same container, while SD is used for sharing the data between multiple calls of each separate function. Be aware that accessing the data is possible only within the same session, when the container process lives on a segment or master. Be aware that for idle sessions Greenplum Database terminates segment processes, which means the related containers would be shut down and the data from GD and SD lost.

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

For information about the `plpy` methods, see https://www.postgresql.org/docs/9.4/plpython-database.htm.

About PL/Container Running PL/Python with Python 3

PL/Container for Greenplum Database 5 supports Python version 3.6+. PL/Container for Greenplum

Database 6 supports Python 3.7+.

If you want to use PL/Container to run the same function body in both Python2 and Python3, you must create 2 different user-defined functions.

Keep in mind that UDFs that you created for Python 2 may not run in PL/Container with Python 3. The following Python references may be useful:

- Changes to Python - What's New in Python 3
- Porting from Python 2 to 3 - Porting Python 2 Code to Python 3

About PL/Container Running PL/R

In the R language container, the module `pg.spi` is implemented. The module contains these methods:

- `pg.spi.exec(stmt)` - Executes the query string `stmt` and returns query result in R `data.frame`. To be able to access the result fields make sure your query returns named fields.
- `pg.spi.prepare(stmt[, argtypes])` - 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.
- `pg.spi.execp(plan[, argtypes])` - Execute a prepared plan.
- `pg.spi.debug(msg)` - Sends a DEBUG2 message to the Greenplum Database log.
- `pg.spi.log(msg)` - Sends a LOG message to the Greenplum Database log.
- `pg.spi.info(msg)` - Sends an INFO message to the Greenplum Database log.
- `pg.spi.notice(msg)` - Sends a NOTICE message to the Greenplum Database log.
- `pg.spi.warning(msg)` - Sends a WARNING message to the Greenplum Database log.
- `pg.spi.error(msg)` - Sends an ERROR message to the Greenplum Database log. An ERROR message raised in Greenplum Database causes the query execution process to stop and the transaction to rollback.
- `pg.spi.fatal(msg)` - Sends a FATAL message to the Greenplum Database log. A FATAL message causes Greenplum Database session to be closed and transaction to be rolled back.

PL/Container does not support this PL/R feature:

- Multi-dimensional arrays.

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

For information about the `pg.spi` methods, see http://www.joeconway.com/plr/doc/plr-spi-rsupport-funcs-normal.html

Docker References

Docker home page https://www.docker.com/

Docker command line interface https://docs.docker.com/engine/reference/commandline/cli/

Dockerfile reference https://docs.docker.com/engine/reference/builder/

For CentOS, see Docker site installation instructions for CentOS.

For a list of Docker commands, see the Docker engine Run Reference.

Installing Docker on Linux systems https://docs.docker.com/engine/installation/linux/centos/

Control and configure Docker with systemd https://docs.docker.com/engine/admin/systemd/

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Greenplum PL/Java Language Extension

This section contains an overview of the Greenplum Database PL/Java language.

Parent topic: Greenplum Database Reference Guide

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 1.5.0. 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 server 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-1.4.3-gp5-rhelosversion-x86_64.gppkg
   ```

3. Ensure that 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`. This example changes the environment variable to use `/usr/lib/jvm`.

  ```
  JAVA_HOME=/usr/lib/jvm
  ```

- Set the `LD_LIBRARY_PATH` to include the directory with the 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/jre/lib/amd64/server`. This example adds the directory to the environment variable.

  ```
  LD_LIBRARY_PATH=$GPHOME/lib:$GPHOME/ext/python/lib:$JAVA_HOME/jre/lib/amd64/server:$LD_LIBRARY_PATH
  ```

This example `gpscp` command copies the file to all hosts specified in the file `gphosts_file`.

```
$ gpscp -f gphosts_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
   ```

Enabling PL/Java and Installing JAR Files

Perform the following steps as the Greenplum Database administrator `gpadmin`.

1. Enable PL/Java in a database by executing the `CREATE EXTENSION` command to register the language. For example, this command enables PL/Java in the `testdb` database:

   ```
   $ psql -d testdb -c 'CREATE EXTENSION pljava;'
   ```

   Note: The PL/Java `install.sql` script, used in previous releases to register the language, is deprecated.

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
```

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

Use the `DROP EXTENSION` command to remove support for PL/Java from a database. For example, this command disables the PL/Java language in the `testdb` database:

```
$ psql -d testdb -c 'DROP EXTENSION pljava;'
```

The default command fails if any existing objects (such as functions) depend on the language. Specify the `CASCADE` option to also drop all dependent objects, including functions that you created with PL/Java.

Note: The PL/Java `uninstall.sql` script, used in previous releases to remove the language registration, is deprecated.

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-1.4.3
```

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
```

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 `java.lang.System` class:

```
CREATE FUNCTION getsysprop(VARCHAR)
  RETURNS VARCHAR
  AS 'java.lang.System.getProperty'
  LANGUAGE java;
```

Run the following command to return the Java `user.home` property:

```
SELECT getsysprop('user.home');
```

Type Mapping

Scalar types are mapped in a straight forward way. This table lists the current mappings.

Table 1. PL/Java data type mapping

PostgreSQL	Java
bool	boolean
char	byte
int2	short
int4	int
int8	long
varchar	java.lang.String
text	java.lang.String
bytea	byte[ ]
date	java.sql.Date

Table 1. PL/Java data type mapping

PostgreSQL	Java
time	java.sql.Time (stored value treated as local time)
timetz	java.sql.Time
timestamp	java.sql.Timestamp (stored value treated as local time)
timestamptz	java.sql.Timestamp
complex	java.sql.ResultSet
setof complex	java.sql.ResultSet

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.

```
CREATE FUNCTION trueIfEvenOrNull(integer)
  RETURNS bool
  AS 'foo.fee.Fum.trueIfEvenOrNull(java.lang.Integer)'
  LANGUAGE java;
```

The Java code would be similar to this:

```
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:

```
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:

```
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:

```
public static String useComplexTest(ResultSet complexTest)
throws SQLException
{
  int base = complexTest.getInt(1);
  int incbase = complexTest.getInt(2);
  Timestamp ctime = complexTest.getTimestamp(3);
  return "Base = \"" + base +
    "\", incbase = \"" + incbase +
    "\", ctime = \"" + ctime + "\"";
}
```

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.

```
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:

```
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`:

```
CREATE FUNCTION javatest.getSystemProperties()
  RETURNS SETOF varchar
  AS 'foo.fee.Bar.getNames'
  IMMUTABLE LANGUAGE java;
```

This simple Java method returns an iterator:

```
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:

```
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.

```
public class Fum implements ResultSetProvider
{
  private final int m_base;
  private final int m_increment;
  public Fum(int base, int increment)
  {
    m_base = base;
    m_increment = increment;
  }
  public boolean assignRowValues(ResultSet receiver, int
currentRow)
  throws SQLException
  {
    // Stop when we reach 12 rows.
    //
    if(currentRow >= 12)
      return false;
    receiver.updateInt(1, m_base);
    receiver.updateInt(2, m_base + m_increment * currentRow);
    receiver.updateTimestamp(3, new
Timestamp(System.currentTimeMillis()));
```

```
    return true;
  }
  public void close()
  {
   // Nothing needed in this example
  }
  public static ResultSetProvider listComplexTests(int base,
int increment)
  throws SQLException
  {
    return new Fum(base, increment);
  }
}
```

The `listComplextTests` 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 RestulSet 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:

```
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:

```
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").cr
eateStatement();
    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:

```
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:

```
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 2. PL/Java Logging Levels

java.util.logging.Level	Greenplum Database Level
SEVERE ERROR	ERROR
WARNING	WARNING
CONFIG	LOG
INFO	INFO
FINE	DEBUG1
FINER	DEBUG2
FINEST	DEBUG3

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 where 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 be 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.

```
{
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`:

```
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:

```
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:

```
> \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.

PL/Java 1.5.0 release - https://github.com/tada/pljava/tree/REL1_5_STABLE.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Greenplum Database Reference Guide

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 EXTENSION` command. For example, run the following command as the `gpadmin` user to register the trusted PL/Perl language for the database named `testdb`:

```
$ psql -d testdb -c 'CREATE EXTENSION plperl;'
```

Removing PL/Perl Support

To remove support for PL/Perl from a database, run the SQL `DROP EXTENSION` command. 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 -c 'DROP EXTENSION plperl;'
```

The default command fails if any existing objects (such as functions) depend on the language. Specify the `CASCADE` option 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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Greenplum MADlib Extension for Analytics

This chapter includes the following information:

- About MADlib
- About Deep Learning
- Installing MADlib
- Upgrading MADlib
- Uninstalling MADlib
- Examples
- References

Parent topic: Greenplum Database Reference Guide

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.

About Deep Learning

Deep learning is a type of machine learning, originally inspired by biology of the brain, that uses a class of algorithms called artificial neural networks. Given the important use cases that can be effectively addressed with deep learning, it is starting to become a more important part of enterprise

computing.

Deep learning support for Keras and TensorFlow was added to Apache MADlib starting with the 1.16 release. Refer to the following documents for more information about deep learning on Greenplum using Apache MADlib:

- MADlib user documentation
- Supported libraries and configuration instructions

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 atabase 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. Unpack the MADlib distribution package. For example:

   ```
   $ tar xzvf madlib-1.17.0-gp5-rhel7-x86_64.tar.gz
   ```

4. Install the software package by running the `gppkg` command. For example:

   ```
   $ gppkg -i ./madlib-1.17.0-gp5-rhel7-x86_64/madlib-1.17.0-gp5-rhel7-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 `testdb` database 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.

For information about the upgrade paths that MADlib supports, see the MADlib support and upgrade matrix in the MADlib FAQ page.

Upgrading a MADlib Package

To upgrade MADlib, run the `gppkg` utility with the `-u` option. This command upgrades an installed MADlib package to MADlib 1.17.

```
$ gppkg -u madlib-1.17.0-gp5-rhel7-x86_64.gppkg
```

Note: Upgrading from MADlib version 1.15 using `gppkg -U` does not work because of a change in the post-uninstall script that was introduced in version 1.15.1. If you are upgrading from MADlib version 1.15 to version 1.15.1 or newer, follow these steps:

1. Remove the existing MADlib version 1.15 rpm (this does not affect the Greenplum Database installation):

   ```
   gppkg -r madlib-1.15.0
   ```

2. Manually remove the remaining MADlib files:

   ```
   rm -rf /usr/local/greenplum-db-5.13.0/madlib/Versions
   ```

3. Install the newer MADlib version. For example:

   ```
   $ gppkg -i ./madlib-1.17.0-gp5-rhel7-x86_64.gppkg
   ```

4. Upgrade the MADlib functions in each database. For example:

   ```
   madpack -p greenplum -c gpadmin@mdw:5432/testdb upgrade
   ```

Upgrading MADlib Functions

After you upgrade the MADlib package, you run the `madpack upgrade` command to upgrade the MADlib functions in Greenplum Database.

Note: The upgrade to MADlib 1.13 has a minor issue with some leftover `knn` functions.

Before running the `madpack` command to upgrade to MADlib 1.13, run these `psql` commands as the `gpadmin` user to drop the leftover functions from the databases where MADlib is installed.

```
psql db_name -c "DROP FUNCTION IF EXISTS schema.knn(VARCHAR);"
psql db_name -c "DROP FUNCTION IF EXISTS schema.knn();"
```

db_name is the name of the database. *schema* is the name of the MADlib schema.

See the MADlib Installation Guide.

The `madpack upgrade` command upgrades the MADlib functions in the database schema. This example command upgrades the MADlib functions in the schema `madlib` of the Greenplum Database `test`.

```
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 database. For example, this command removes MADlib objects from the database `testdb`.

```
$ 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 1.17.

```
$ gppkg -r madlib-1.17.0-gp5-rhel7-x86_64
```

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.

```
$ 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:

```
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.

```
SELECT madlib.linregr_train (
   'regr_example',         -- source table
```

```
   'regr_example_model',   -- output model table
   'y',                    -- dependent variable
   'ARRAY[1, x1, x2]'      -- independent variables
);
```

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:

```
SELECT * FROM regr_example_model;
-[ RECORD 1 ]------------+------------------------
coef                     | {0.111111111111127,1.14814814814815,1.01851851851852}
r2                       | 0.968612680477111
std_err                  | {1.49587911309236,0.207043331249903,0.346449758034495}
t_stats                  | {0.0742781352708591,5.54544858420156,2.93987366103776}
p_values                 | {0.952799748147436,0.113579771006374,0.208730790695278}
condition_no             | 22.650203241881
num_rows_processed       | 4
num_missing_rows_skipped | 0
variance_covariance      | {{2.23765432098598,-0.257201646090342,-0.437242798353582},
                           {-0.257201646090342,0.042866941015057,0.0342935528120456},
                           {-0.437242798353582,0.0342935528120457,0.12002743484216}}
```

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:

```
SELECT regr_example.*,
        madlib.linregr_predict ( ARRAY[1, x1, x2], m.coef ) as predict,
        y - madlib.linregr_predict ( ARRAY[1, x1, x2], m.coef ) as residual
FROM regr_example, regr_example_model m;
 id | y  | x1 | x2 |     predict      |      residual
----+----+----+----+------------------+--------------------
  1 |  5 |  2 |  3 | 5.46296296296297 | -0.462962962962971
  3 |  6 |  4 |  1 | 5.72222222222224 |  0.277777777777762
  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.

```
    pre    |  post  |      support
-----------+--------+-------------------
 {diapers} | {beer} | 0.714285714285714
 {chips}   | {beer} | 0.428571428571429
(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.

   ```
   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.

   ```
   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.

   ```
   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.

   ```
   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.

```
SELECT * FROM example_classified;
 key | class | nb_prob
-----+-------+---------
   1 |     1 |     0.4
   1 |     2 |     0.6
   3 |     1 |     0.5
   3 |     2 |     0.5
   2 |     1 |    0.25
   2 |     2 |    0.75
(6 rows)
```

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.

```
  day | play | outlook  | temperature | humidity | wind
-----+------+----------+-------------+----------+--------
 2   | No   | Sunny    | Hot         | High     | Strong
 4   | Yes  | Rain     | Mild        | High     | Weak
 6   | No   | Rain     | Cool        | Normal   | Strong
 8   | No   | Sunny    | Mild        | High     | Weak
10   | Yes  | Rain     | Mild        | Normal   | Weak
12   | Yes  | Overcast | Mild        | High     | Strong
14   | No   | Rain     | Mild        | High     | Strong
 1   | No   | Sunny    | Hot         | High     | Weak
 3   | Yes  | Overcast | Hot         | High     | Weak
 5   | Yes  | Rain     | Cool        | Normal   | Weak
 7   | Yes  | Overcast | Cool        | Normal   | Strong
 9   | Yes  | Sunny    | Cool        | Normal   | Weak
11   | Yes  | Sunny    | Mild        | Normal   | Strong
13   | Yes  | Overcast | Hot         | Normal   | Weak
(14 rows)
```

1. Create the training table.

```
DROP TABLE IF EXISTS weather_example;
CREATE TABLE weather_example (
   day int,
   play int,
   attrs int[]
);
INSERT INTO weather_example VALUES
```

```
   ( 2, 0, '{1,1,1,1}'), -- sunny, hot, high, strong
   ( 4, 1, '{3,2,1,2}'), -- rain, mild, high, weak
   ( 6, 0, '{3,3,2,1}'), -- rain, cool, normal, strong
   ( 8, 0, '{1,2,1,2}'), -- sunny, mild, high, weak
   (10, 1, '{3,2,2,2}'), -- rain, mild, normal, weak
   (12, 1, '{2,2,1,1}'), -- etc.
   (14, 0, '{3,2,1,1}'),
   ( 1, 0, '{1,1,1,2}'),
   ( 3, 1, '{2,1,1,2}'),
   ( 5, 1, '{3,3,2,2}'),
   ( 7, 1, '{2,3,2,1}'),
   ( 9, 1, '{1,3,2,2}'),
   (11, 1, '{1,2,2,1}'),
   (13, 1, '{2,1,2,2}');
```

2. Create the model from the training table.

```
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:

```
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
     0 |    3 |     2 |   1 |        2
     0 |    1 |     2 |   0 |        3
     1 |    1 |     1 |   2 |        3
     1 |    1 |     3 |   3 |        3
     1 |    3 |     1 |   3 |        2
     0 |    4 |     1 |   3 |        2
     0 |    2 |     3 |   1 |        3
     0 |    2 |     1 |   2 |        3
     1 |    2 |     2 |   4 |        3
     1 |    4 |     2 |   6 |        2
(20 rows)
```

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)
```

References

MADlib web site is at http://madlib.apache.org/.

MADlib documentation is at http://madlib.apache.org/documentation.html.

PivotalR is a first class R package that enables users to interact with data resident in Greenplum Database and MADLib using an R client.

About MADlib, R, and PivotalR

The R language is an open-source language that is used for statistical computing. PivotalR is an R package that enables users to interact with data resident in Greenplum Database using the R client. Using PivotalR requires that MADlib is installed on the Greenplum Database.

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.

The R package for PivotalR can be found at https://cran.r-project.org/web/packages/PivotalR/index.html.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

Greenplum Partner Connector API

With the Greenplum Partner Connector API (GPPC API), you can write portable Greenplum

Database user-defined functions (UDFs) in the C and C++ programming languages. Functions that you develop with the GPPC API require no recompilation or modification to work with older or newer Greenplum Database versions.

Functions that you write to the GPPC API can be invoked using SQL in Greenplum Database. The API provides a set of functions and macros that you can use to issue SQL commands through the Server Programming Interface (SPI), manipulate simple and composite data type function arguments and return values, manage memory, and handle data.

You compile the C/C++ functions that you develop with the GPPC API into a shared library. The GPPC functions are available to Greenplum Database users after the shared library is installed in the Greenplum Database cluster and the GPPC functions are registered as SQL UDFs.

Note: The Greenplum Partner Connector is supported for Greenplum Database versions 4.3.5.0 and later.

This topic contains the following information:

Parent topic: Greenplum Database Reference Guide

Using the GPPC API

The GPPC API shares some concepts with C language functions as defined by PostgreSQL. Refer to C-Language Functions in the PostgreSQL documentation for detailed information about developing C language functions.

The GPPC API is a wrapper that makes a C/C++ function SQL-invokable in Greenplum Database. This wrapper shields GPPC functions that you write from Greenplum Database library changes by normalizing table and data manipulation and SPI operations through functions and macros defined by the API.

The GPPC API includes functions and macros to:

- Operate on base and composite data types.
- Process function arguments and return values.

- Allocate and free memory.
- Log and report errors to the client.
- Issue SPI queries.
- Return a table or set of rows.
- Process tables as function input arguments.

Requirements

When you develop with the GPPC API:

- You must develop your code on a system with the same hardware and software architecture as that of your Greenplum Database hosts.
- You must write the GPPC function(s) in the C or C++ programming languages.
- The function code must use the GPPC API, data types, and macros.
- The function code must *not* use the PostgreSQL C-Language Function API, header files, functions, or macros.
- The function code must *not* `#include` the `postgres.h` header file or use `PG_MODULE_MAGIC`.
- You must use only the GPPC-wrapped memory functions to allocate and free memory. See Memory Handling.
- Symbol names in your object files must not conflict with each other nor with symbols defined in the Greenplum Database server. You must rename your functions or variables if you get error messages to this effect.

Header and Library Files

The GPPC header files and libraries are installed in `$GPHOME`:

- $GPHOME/include/gppc.h - the main GPPC header file
- $GPHOME/include/gppc_config.h - header file defining the GPPC version
- $GPHOME/lib/libgppc.[a, so, so.1, so.1.2] - GPPC archive and shared libraries

Data Types

The GPPC functions that you create will operate on data residing in Greenplum Database. The GPPC API includes data type definitions for equivalent Greenplum Database SQL data types. You must use these types in your GPPC functions.

The GPPC API defines a generic data type that you can use to represent any GPPC type. This data type is named `GppcDatum`, and is defined as follows:

```
typedef int64_t GppcDatum;
```

The following table identifies each GPPC data type and the SQL type to which it maps.

SQL Type	GPPC Type	GPPC Oid for Type
boolean	GppcBool	GppcOidBool
char (single byte)	GppcChar	GppcOidChar
int2/smallint	GppcInt2	GppcOidInt2
int4/integer	GppcInt4	GppcOidInt4
int8/bigint	GppcInt8	GppcOidInt8
float4/real	GppcFloat4	GppcOidFloat4

SQL Type	GPPC Type	GPPC Oid for Type
float8/double	GppcFloat8	GppcOidFloat8
text	*GppcText	GppcOidText
varchar	*GppcVarChar	GppcOidVarChar
char	*GppcBpChar	GppcOidBpChar
bytea	*GppcBytea	GppcOidBytea
numeric	*GppcNumeric	GppcOidNumeric
date	GppcDate	GppcOidDate
time	GppcTime	GppcOidTime
timetz	*GppcTimeTz	GppcOidTimeTz
timestamp	GppcTimestamp	GppcOidTimestamp
timestamptz	GppcTimestampTz	GppcOidTimestampTz
anytable	GppcAnyTable	GppcOidAnyTable
oid	GppcOid	

The GPPC API treats text, numeric, and timestamp data types specially, providing functions to operate on these types.

Example GPPC base data type declarations:

```
GppcText       message;
GppcInt4       arg1;
GppcNumeric    total_sales;
```

The GPPC API defines functions to convert between the generic `GppcDatum` type and the GPPC specific types. For example, to convert from an integer to a datum:

```
GppcInt4 num = 13;
GppcDatum num_dat = GppcInt4GetDatum(num);
```

Composite Types

A composite data type represents the structure of a row or record, and is comprised of a list of field names and their data types. This structure information is typically referred to as a tuple descriptor. An instance of a composite type is typically referred to as a tuple or row. A tuple does not have a fixed layout and can contain null fields.

The GPPC API provides an interface that you can use to define the structure of, to access, and to set tuples. You will use this interface when your GPPC function takes a table as an input argument or returns table or set of record types. Using tuples in table and set returning functions is covered later in this topic.

Function Declaration, Arguments, and Results

The GPPC API relies on macros to declare functions and to simplify the passing of function arguments and results. These macros include:

Task	Macro Signature	Description
Make a function SQL-invokable	`GPPC_FUNCTION_INFO(`*`function_name`*`)`	Glue to make function *`function_name`* SQL-invokable.
Declare a function	`GppcDatum` *`function_name`*`(GPPC_FUNCTION_ARGS)`	Declare a GPPC function named *`function_name`*; every function must have this same signature.

Task	Macro Signature	Description
Return the number of arguments	`GPPC_NARGS()`	Return the number of arguments passed to the function.
Fetch an argument	`GPPC_GETARG_<ARGTYPE>(arg_num)`	Fetch the value of argument number *arg_num* (starts at 0), where `<ARGTYPE>` identifies the data type of the argument. For example, `GPPC_GETARG_FLOAT8(0)`.
Fetch and make a copy of a text-type argument	`GPPC_GETARG_<ARGTYPE>_COPY(arg_num)`	Fetch and make a copy of the value of argument number *arg_num* (starts at 0). `<ARGTYPE>` identifies the text type (text, varchar, bpchar, bytea). For example, `GPPC_GETARG_BYTEA_COPY(1)`.
Determine if an argument is NULL	`GPPC_ARGISNULL(arg_num)`	Return whether or not argument number *arg_num* is NULL.
Return a result	`GPPC_RETURN_<ARGTYPE>(return_val)`	Return the value *return_val*, where `<ARGTYPE>` identifies the data type of the return value. For example, `GPPC_RETURN_INT4(131)`.

When you define and implement your GPPC function, you must declare it with the GPPC API using the two declarations identified above. For example, to declare a GPPC function named `add_int4s()`:

```
GPPC_FUNCTION_INFO(add_int4s);
GppcDatum add_int4s(GPPC_FUNCTION_ARGS);

GppcDatum
add_int4s(GPPC_FUNCTION_ARGS)
{
  // code here
}
```

If the `add_int4s()` function takes two input arguments of type `int4`, you use the `GPPC_GETARG_INT4(arg_num)` macro to access the argument values. The argument index starts at 0. For example:

```
GppcInt4  first_int = GPPC_GETARG_INT4(0);
GppcInt4  second_int = GPPC_GETARG_INT4(1);
```

If `add_int4s()` returns the sum of the two input arguments, you use the `GPPC_RETURN_INT8(return_val)` macro to return this sum. For example:

```
GppcInt8  sum = first_int + second_int;
GPPC_RETURN_INT8(sum);
```

The complete GPPC function:

```
GPPC_FUNCTION_INFO(add_int4s);
GppcDatum add_int4s(GPPC_FUNCTION_ARGS);

GppcDatum
add_int4s(GPPC_FUNCTION_ARGS)
{
  // get input arguments
  GppcInt4    first_int = GPPC_GETARG_INT4(0);
  GppcInt4    second_int = GPPC_GETARG_INT4(1);

  // add the arguments
  GppcInt8    sum = first_int + second_int;

  // return the sum
  GPPC_RETURN_INT8(sum);
```

```
}
```

Memory Handling

The GPPC API provides functions that you use to allocate and free memory, including text memory. You must use these functions for all memory operations.

Function Name	Description
void *GppcAlloc(size_t *num*)	Allocate *num* bytes of uninitialized memory.
void *GppcAlloc0(size_t *num*)	Allocate *num* bytes of 0-initialized memory.
void *GppcRealloc(void **ptr*, size_t *num*)	Resize pre-allocated memory.
void GppcFree(void **ptr*)	Free allocated memory.

After you allocate memory, you can use system functions such as `memcpy()` to set the data.

The following example allocates an array of `GppcDatum`s and sets the array to datum versions of the function input arguments:

```
GppcDatum  *values;
int attnum = GPPC_NARGS();

// allocate memory for attnum values
values = GppcAlloc( sizeof(GppcDatum) * attnum );

// set the values
for( int i=0; i<attnum; i++ ) {
    GppcDatum d = GPPC_GETARG_DATUM(i);
    values[i] = d;
}
```

When you allocate memory for a GPPC function, you allocate it in the current context. The GPPC API includes functions to return, create, switch, and reset memory contexts.

Function Name	Description
GppcMemoryContext GppcGetCurrentMemoryContext(void)	Return the current memory context.
GppcMemoryContext GppcMemoryContextCreate(GppcMemoryContext *parent*)	Create a new memory context under *parent*.
GppcMemoryContext GppcMemoryContextSwitchTo(GppcMemoryContext *context*)	Switch to the memory context *context*.
void GppcMemoryContextReset(GppcMemoryContext *context*)	Reset (free) the memory in memory context *context*.

Greenplum Database typically calls a SQL-invoked function in a per-tuple context that it creates and deletes every time the server backend processes a table row. Do not assume that memory allocated in the current memory context is available across multiple function calls.

Working With Variable-Length Text Types

The GPPC API supports the variable length text, varchar, blank padded, and byte array types. You must use the GPPC API-provided functions when you operate on these data types. Variable text manipulation functions provided in the GPPC API include those to allocate memory for, determine string length of, get string pointers for, and access these types:

Function Name	Description

GppcText GppcAllocText(size_t *len*) GppcVarChar GppcAllocVarChar(size_t *len*) GppcBpChar GppcAllocBpChar(size_t *len*) GppcBytea GppcAllocBytea(size_t *len*)	Allocate *len* bytes of memory for the varying length type.
size_t GppcGetTextLength(GppcText *s*) size_t GppcGetVarCharLength(GppcVarChar *s*) size_t GppcGetBpCharLength(GppcBpChar *s*) size_t GppcGetByteaLength(GppcBytea *b*)	Return the number of bytes in the memory chunk.
char *GppcGetTextPointer(GppcText *s*) char *GppcGetVarCharPointer(GppcVarChar *s*) char *GppcGetBpCharPointer(GppcBpChar *s*) char *GppcGetByteaPointer(GppcBytea *b*)	Return a string pointer to the head of the memory chunk. The string is not null-terminated.
char *GppcTextGetCString(GppcText *s*) char *GppcVarCharGetCString(GppcVarChar *s*) char *GppcBpCharGetCString(GppcBpChar *s*)	Return a string pointer to the head of the memory chunk. The string is null-terminated.
GppcText *GppcCStringGetText(const char **s*) GppcVarChar *GppcCStringGetVarChar(const char **s*) GppcBpChar *GppcCStringGetBpChar(const char **s*)	Build a varying-length type from a character string.

Memory returned by the `GppcGet<VLEN_ARGTYPE>Pointer()` functions may point to actual database content. Do not modify the memory content. The GPPC API provides functions to allocate memory for these types should you require it. After you allocate memory, you can use system functions such as `memcpy()` to set the data.

The following example manipulates text input arguments and allocates and sets result memory for a text string concatenation operation:

```
GppcText first_textstr = GPPC_GETARG_TEXT(0);
GppcText second_textstr = GPPC_GETARG_TEXT(1);

// determine the size of the concatenated string and allocate
// text memory of this size
size_t arg0_len = GppcGetTextLength(first_textstr);
size_t arg1_len = GppcGetTextLength(second_textstr);
GppcText retstring = GppcAllocText(arg0_len + arg1_len);

// construct the concatenated return string; copying each string
// individually
memcpy(GppcGetTextPointer(retstring), GppcGetTextPointer(first_textstr), arg0_len);
memcpy(GppcGetTextPointer(retstring) + arg0_len, GppcGetTextPointer(second_textstr), a
rg1_len);
```

Error Reporting and Logging

The GPPC API provides error reporting and logging functions. The API defines reporting levels equivalent to those in Greenplum Database:

```
typedef enum GppcReportLevel
{
        GPPC_DEBUG1                             = 10,
        GPPC_DEBUG2                             = 11,
        GPPC_DEBUG3                             = 12,
        GPPC_DEBUG4                             = 13,
        GPPC_DEBUG                              = 14,
        GPPC_LOG                                = 15,
```

```
        GPPC_INFO                               = 17,
        GPPC_NOTICE                             = 18,
         GPPC_WARNING                          = 19,
        GPPC_ERROR                              = 20,
} GppcReportLevel;
```

(The Greenplum Database `client_min_messages` server configuration parameter governs the current client logging level. The `log_min_messages` configuration parameter governs the current log-to-logfile level.)

A GPPC report includes the report level, a report message, and an optional report callback function.

Reporting and handling functions provide by the GPPC API include:

Function Name	Description
GppcReport()	Format and print/log a string of the specified report level.
GppcInstallReportCallback()	Register/install a report callback function.
GppcUninstallReportCallback()	Uninstall a report callback function.
GppcGetReportLevel()	Retrieve the level from an error report.
GppcGetReportMessage()	Retrieve the message from an error report.
GppcCheckForInterrupts()	Error out if an interrupt is pending.

The `GppcReport()` function signature is:

```
void GppcReport(GppcReportLevel elevel, const char *fmt, ...);
```

`GppcReport()` takes a format string input argument similar to `printf()`. The following example generates an error level report message that formats a GPPC text argument:

```
GppcText  uname = GPPC_GETARG_TEXT(1);
GppcReport(GPPC_ERROR, "Unknown user name: %s", GppcTextGetCString(uname));
```

Refer to the GPPC example code for example report callback handlers.

SPI Functions

The Greenplum Database Server Programming Interface (SPI) provides writers of C/C++ functions the ability to run SQL commands within a GPPC function. For additional information on SPI functions, refer to Server Programming Interface in the PostgreSQL documentation.

The GPPC API exposes a subset of PostgreSQL SPI functions. This subset enables you to issue SPI queries and retrieve SPI result values in your GPPC function. The GPPC SPI wrapper functions are:

SPI Function Name	GPPC Function Name	Description
SPI_connect()	GppcSPIConnect()	Connect to the Greenplum Database server programming interface.
SPI_finish()	GppcSPIFinish()	Disconnect from the Greenplum Database server programming interface.
SPI_exec()	GppcSPIExec()	Execute a SQL statement, returning the number of rows.
SPI_getvalue()	GppcSPIGetValue()	Retrieve the value of a specific attribute by number from a SQL result as a character string.
	GppcSPIGetDatum()	Retrieve the value of a specific attribute by number from a SQL result as a `GppcDatum`.
	GppcSPIGetValueByName()	Retrieve the value of a specific attribute by name from a SQL result as a character string.
	GppcSPIGetDatumByName()	Retrieve the value of a specific attribute by name from a SQL result as a `GppcDatum`.

When you create a GPPC function that accesses the server programming interface, your function should comply with the following flow:

```
GppcSPIConnect();
GppcSPIExec(...)
// process the results - GppcSPIGetValue(...), GppcSPIGetDatum(...)
GppcSPIFinish()
```

You use `GppcSPIExec()` to execute SQL statements in your GPPC function. When you call this function, you also identify the maximum number of rows to return. The function signature of `GppcSPIExec()` is:

```
GppcSPIResult GppcSPIExec(const char *sql_statement, long rcount);
```

`GppcSPIExec()` returns a `GppcSPIResult` structure. This structure represents SPI result data. It includes a pointer to the data, information about the number of rows processed, a counter, and a result code. The GPPC API defines this structure as follows:

```
typedef struct GppcSPIResultData
{
    struct GppcSPITupleTableData   *tuptable;
    uint32_t                       processed;
    uint32_t                       current;
    int                            rescode;
} GppcSPIResultData;
typedef GppcSPIResultData *GppcSPIResult;
```

You can set and use the `current` field in the `GppcSPIResult` structure to examine each row of the `tuptable` result data.

The following code excerpt uses the GPPC API to connect to SPI, execute a simple query, loop through query results, and finish processing:

```
GppcSPIResult   result;
char            *attname = "id";
char            *query = "SELECT i, 'foo' || i AS val FROM generate_series(1, 10)i ORD
ER BY 1";
bool            isnull = true;

// connect to SPI
if( GppcSPIConnect() < 0 ) {
    GppcReport(GPPC_ERROR, "cannot connect to SPI");
}

// execute the query, returning all rows
result = GppcSPIExec(query, 0);

// process result
while( result->current < result->processed ) {
    // get the value of attname column as a datum, making a copy
    datum = GppcSPIGetDatumByName(result, attname, &isnull, true);

    // do something with value

    // move on to next row
    result->current++;
}

// complete processing
GppcSPIFinish();
```

About Tuple Descriptors and Tuples

A table or a set of records contains one or more tuples (rows). The structure of each attribute of a tuple is defined by a tuple descriptor. A tuple descriptor defines the following for each attribute in

the tuple:

- attribute name
- object identifer of the attribute data type
- byte length of the attribute data type
- object identifier of the attribute modifer

The GPPC API defines an abstract type, `GppcTupleDesc`, to represent a tuple/row desriptor. The API also provides functions that you can use to create, access, and set tuple descriptors:

Function Name	Description
GppcCreateTemplateTupleDesc()	Create an empty tuple descriptor with a specified number of attributes.
GppcTupleDescInitEntry()	Add an attribute to the tuple descriptor at a specified position.
GppcTupleDescNattrs()	Fetch the number of attributes in the tuple descriptor.
GppcTupleDescAttrName()	Fetch the name of the attribute in a specific position (starts at 0) in the tuple descriptor.
GppcTupleDescAttrType()	Fetch the type object identifier of the attribute in a specific position (starts at 0) in the tuple descriptor.
GppcTupleDescAttrLen()	Fetch the type length of an attribute in a specific position (starts at 0) in the tuple descriptor.
GppcTupleDescAttrTypmod()	Fetch the type modifier object identifier of an attribute in a specific position (starts at 0) in the tuple descriptor.

To construct a tuple descriptor, you first create a template, and then fill in the descriptor fields for each attribute. The signatures for these functions are:

```
GppcTupleDesc GppcCreateTemplateTupleDesc(int natts);
void GppcTupleDescInitEntry(GppcTupleDesc desc, uint16_t attno,
                            const char *attname, GppcOid typid, int32_t typmod);
```

In some cases, you may want to initialize a tuple descriptor entry from an attribute definition in an existing tuple. The following functions fetch the number of attributes in a tuple descriptor, as well as the definition of a specific attribute (by number) in the descriptor:

```
int GppcTupleDescNattrs(GppcTupleDesc tupdesc);
const char *GppcTupleDescAttrName(GppcTupleDesc tupdesc, int16_t attno);
GppcOid GppcTupleDescAttrType(GppcTupleDesc tupdesc, int16_t attno);
int16_t GppcTupleDescAttrLen(GppcTupleDesc tupdesc, int16_t attno);
int32_t GppcTupleDescAttrTypmod(GppcTupleDesc tupdesc, int16_t attno);
```

The following example initializes a two attribute tuple descriptor. The first attribute is initialized with the definition of an attribute from a different descriptor, and the second attribute is initialized to a boolean type attribute:

```
GppcTupleDesc       tdesc;
GppcTupleDesc       indesc = some_input_descriptor;

// initalize the tuple descriptor with 2 attributes
tdesc = GppcCreateTemplateTupleDesc(2);

// use third attribute from the input descriptor
GppcTupleDescInitEntry(tdesc, 1,
         GppcTupleDescAttrName(indesc, 2),
         GppcTupleDescAttrType(indesc, 2),
         GppcTupleDescAttrTypmod(indesc, 2));

// create the boolean attribute
GppcTupleDescInitEntry(tdesc, 2, "is_active", GppcOidBool, 0);
```

The GPPC API defines an abstract type, `GppcHeapTuple`, to represent a tuple/record/row. A tuple

is defined by its tuple descriptor, the value for each tuple attribute, and an indicator of whether or not each value is NULL.

The GPPC API provides functions that you can use to set and access a tuple and its attributes:

Function Name	Description
GppcHeapFormTuple()	Form a tuple from an array of `GppcDatum`s.
GppcBuildHeapTupleDatum()	Form a `GppcDatum` tuple from an array of `GppcDatum`s.
GppcGetAttributeByName()	Fetch an attribute from the tuple by name.
GppcGetAttributeByNum()	Fetch an attribute from the tuple by number (starts at 1).

The signatures for the tuple-building GPPC functions are:

```
GppcHeapTuple GppcHeapFormTuple(GppcTupleDesc tupdesc, GppcDatum *values, bool *nulls)
;
GppcDatum    GppcBuildHeapTupleDatum(GppcTupleDesc tupdesc, GppcDatum *values, bool *n
ulls);
```

The following code excerpt constructs a `GppcDatum` tuple from the tuple descriptor in the above code example, and from integer and boolean input arguments to a function:

```
GppcDatum intarg = GPPC_GETARG_INT4(0);
GppcDatum boolarg = GPPC_GETARG_BOOL(1);
GppcDatum result, values[2];
bool nulls[2] = { false, false };

// construct the values array
values[0] = intarg;
values[1] = boolarg;
result = GppcBuildHeapTupleDatum( tdesc, values, nulls );
```

Set-Returning Functions

Greenplum Database UDFs whose signatures include `RETURNS SETOF RECORD` or `RETURNS TABLE( ... )` are set-returning functions.

The GPPC API provides support for returning sets (for example, multiple rows/tuples) from a GPPC function. Greenplum Database calls a set-returning function (SRF) once for each row or item. The function must save enough state to remember what it was doing and to return the next row on each call. Memory that you allocate in the SRF context must survive across multiple function calls.

The GPPC API provides macros and functions to help keep track of and set this context, and to allocate SRF memory. They include:

Function/Macro Name	Description
GPPC_SRF_RESULT_DESC()	Get the output row tuple descriptor for this SRF. The result tuple descriptor is determined by an output table definition or a `DESCRIBE` function.
GPPC_SRF_IS_FIRSTCALL()	Determine if this is the first call to the SRF.
GPPC_SRF_FIRSTCALL_INIT()	Initialize the SRF context.
GPPC_SRF_PERCALL_SETUP()	Restore the context on each call to the SRF.
GPPC_SRF_RETURN_NEXT()	Return a value from the SRF and continue processing.
GPPC_SRF_RETURN_DONE()	Signal that SRF processing is complete.
GppSRFAlloc()	Allocate memory in this SRF context.
GppSRFAlloc0()	Allocate memory in this SRF context and initialize it to zero.
GppSRFSave()	Save user state in this SRF context.

Function/Macro Name	Description
GppSRFRestore()	Restore user state in this SRF context.

The `GppcFuncCallContext` structure provides the context for an SRF. You create this context on the first call to your SRF. Your set-returning GPPC function must retrieve the function context on each invocation. For example:

```
// set function context
GppcFuncCallContext fctx;
if (GPPC_SRF_IS_FIRSTCALL()) {
    fctx = GPPC_SRF_FIRSTCALL_INIT();
}
fctx = GPPC_SRF_PERCALL_SETUP();
// process the tuple
```

The GPPC function must provide the context when it returns a tuple result or to indicate that processing is complete. For example:

```
GPPC_SRF_RETURN_NEXT(fctx, result_tuple);
// or
GPPC_SRF_RETURN_DONE(fctx);
```

Use a `DESCRIBE` function to define the output tuple descriptor of a function that uses the `RETURNS SETOF RECORD` clause. Use the `GPPC_SRF_RESULT_DESC()` macro to get the output tuple descriptor of a function that uses the `RETURNS TABLE( ... )` clause.

Refer to the GPPC Set-Returning Function Example for a set-returning function code and deployment example.

Table Functions

The GPPC API provides the `GppcAnyTable` type to pass a table to a function as an input argument, or to return a table as a function result.

Table-related functions and macros provided in the GPPC API include:

Function/Macro Name	Description
GPPC_GETARG_ANYTABLE()	Fetch an anytable function argument.
GPPC_RETURN_ANYTABLE()	Return the table.
GppcAnyTableGetTupleDesc()	Fetch the tuple descriptor for the table.
GppcAnyTableGetNextTuple()	Fetch the next row in the table.

You can use the `GPPC_GETARG_ANYTABLE()` macro to retrieve a table input argument. When you have access to the table, you can examine the tuple descriptor for the table using the `GppcAnyTableGetTupleDesc()` function. The signature of this function is:

```
GppcTupleDesc GppcAnyTableGetTupleDesc(GppcAnyTable t);
```

For example, to retrieve the tuple descriptor of a table that is the first input argument to a function:

```
GppcAnyTable     intbl;
GppcTupleDesc    in_desc;

intbl = GPPC_GETARG_ANYTABLE(0);
in_desc = GppcAnyTableGetTupleDesc(intbl);
```

The `GppcAnyTableGetNextTuple()` function fetches the next row from the table. Similarly, to retrieve the next tuple from the table above:

```
GppcHeapTuple    ntuple;
```

```
ntuple = GppcAnyTableGetNextTuple(intbl);
```

Limitations

The GPPC API does not support the following operators with Greenplum Database version 5.0.x:

- integer || integer
- integer = text
- text < integer

Sample Code

The gppc test directory in the Greenplum Database github repository includes sample GPPC code:

- `gppc_demo/` - sample code exercising GPPC SPI functions, error reporting, data type argument and return macros, set-returning functions, and encoding functions
- `tabfunc_gppc_demo/` - sample code exercising GPPC table and set-returning functions

Building a GPPC Shared Library with PGXS

You compile functions that you write with the GPPC API into one or more shared libraries that the Greenplum Database server loads on demand.

You can use the PostgreSQL build extension infrastructure (PGXS) to build the source code for your GPPC functions against a Greenplum Database installation. This framework automates common build rules for simple modules. If you have a more complicated use case, you will need to write your own build system.

To use the PGXS infrastructure to generate a shared library for functions that you create with the GPPC API, create a simple `Makefile` that sets PGXS-specific variables.

Note: Refer to Extension Building Infrastructure in the PostgreSQL documentation for information about the `Makefile` variables supported by PGXS.
For example, the following `Makefile` generates a shared library named `sharedlib_name.so` from two C source files named `src1.c` and `src2.c`:

```
MODULE_big = sharedlib_name
OBJS = src1.o src2.o
PG_CPPFLAGS = -I$(shell $(PG_CONFIG) --includedir)
SHLIB_LINK = -L$(shell $(PG_CONFIG) --libdir) -lgppc

PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
```

`MODULE_big` identifes the base name of the shared library generated by the `Makefile`.

`PG_CPPFLAGS` adds the Greenplum Database installation include directory to the compiler header file search path.

`SHLIB_LINK` adds the Greenplum Database installation library directory to the linker search path. This variable also adds the GPPC library (`-lgppc`) to the link command.

The `PG_CONFIG` and `PGXS` variable settings and the `include` statement are required and typically reside in the last three lines of the `Makefile`.

Registering a GPPC Function with Greenplum Database

Before users can invoke a GPPC function from SQL, you must register the function with Greenplum Database.

Registering a GPPC function involves mapping the GPPC function signature to a SQL user-defined

function. You define this mapping with the `CREATE FUNCTION .. AS` command specifying the GPPC shared library name. You may choose to use the same name or differing names for the GPPC and SQL functions.

Sample `CREATE FUNCTION ... AS` syntax follows:

```
CREATE FUNCTION sql_function_name(arg[, ...]) RETURNS return_type
  AS 'shared_library_path'[, 'gppc_function_name']
LANGUAGE C STRICT [WITH (DESCRIBE=describe_function)];
```

You may omit the shared library `.so` extension when you specify *`shared_library_path`*.

The following command registers the example `add_int4s()` function referenced earlier in this topic to a SQL UDF named `add_two_int4s_gppc()` if the GPPC function was compiled and linked in a shared library named `gppc_try.so`:

```
CREATE FUNCTION add_two_int4s_gppc(int4, int4) RETURNS int8
  AS 'gppc_try.so', 'add_int4s'
LANGUAGE C STRICT;
```

About Dynamic Loading

You specify the name of the GPPC shared library in the SQL `CREATE FUNCTION ... AS` command to register a GPPC function in the shared library with Greenplum Database. The Greenplum Database dynamic loader loads a GPPC shared library file into memory the first time that a user invokes a user-defined function linked in that shared library. If you do not provide an absolute path to the shared library in the `CREATE FUNCTION ... AS` command, Greenplum Database attempts to locate the library using these ordered steps:

1. If the shared library file path begins with the string `$libdir`, Greenplum Database looks for the file in the PostgreSQL package library directory. Run the `pg_config --pkglibdir` command to determine the location of this directory.
2. If the shared library file name is specified without a directory prefix, Greenplum Database searches for the file in the directory identified by the `dynamic_library_path` server configuration paramater value.
3. The current working directory.

Packaging and Deployment Considerations

You must package the GPPC shared library and SQL function registration script in a form suitable for deployment by the Greenplum Database administrator in the Greenplum cluster. Provide specific deployment instructions for your GPPC package.

When you construct the package and deployment instructions, take into account the following:

- Consider providing a shell script or program that the Greenplum Database administrator runs to both install the shared library to the desired file system location and register the GPPC functions.
- The GPPC shared library must be installed to the same file system location on the master host and on every segment host in the Greenplum Database cluster.
- The `gpadmin` user must have permission to traverse the complete file system path to the GPPC shared library file.
- The file system location of your GPPC shared library after it is installed in the Greenplum Database deployment determines how you reference the shared library when you register a function in the library with the `CREATE FUNCTION ... AS` command.
- Create a `.sql` script file that registers a SQL UDF for each GPPC function in your GPPC shared library. The functions that you create in the `.sql` registration script must reference the deployment location of the GPPC shared library. Include this script in your GPPC

deployment package.

- Document the instructions for running your GPPC package deployment script, if you provide one.
- Document the instructions for installing the GPPC shared library if you do not include this task in a package deployment script.
- Document the instructions for installing and running the function registration script if you do not include this task in a package deployment script.

GPPC Text Function Example

In this example, you develop, build, and deploy a GPPC shared library and register and run a GPPC function named `concat_two_strings`. This function uses the GPPC API to concatenate two string arguments and return the result.

You will develop the GPPC function on your Greenplum Database master host. Deploying the GPPC shared library that you create in this example requires administrative access to your Greenplum Database cluster.

Perform the following procedure to run the example:

1. Log in to the Greenplum Database master host and set up your environment. For example:

   ```
   $ ssh gpadmin@<gpmaster>
   gpadmin@gpmaster$ . /usr/local/greenplum-db/greenplum_path.sh
   ```

2. Create a work directory and navigate to the new directory. For example:

   ```
   gpadmin@gpmaster$ mkdir gppc_work
   gpadmin@gpmaster$ cd gppc_work
   ```

3. Prepare a file for GPPC source code by opening the file in the editor of your choice. For example, to open a file named `gppc_concat.c` using `vi`:

   ```
   gpadmin@gpmaster$ vi gppc_concat.c
   ```

4. Copy/paste the following code into the file:

   ```
   #include <stdio.h>
   #include <string.h>
   #include "gppc.h"

   // make the function SQL-invokable
   GPPC_FUNCTION_INFO(concat_two_strings);

   // declare the function
   GppcDatum concat_two_strings(GPPC_FUNCTION_ARGS);

   GppcDatum
   concat_two_strings(GPPC_FUNCTION_ARGS)
   {
       // retrieve the text input arguments
       GppcText arg0 = GPPC_GETARG_TEXT(0);
       GppcText arg1 = GPPC_GETARG_TEXT(1);

       // determine the size of the concatenated string and allocate
       // text memory of this size
       size_t arg0_len = GppcGetTextLength(arg0);
       size_t arg1_len = GppcGetTextLength(arg1);
       GppcText retstring = GppcAllocText(arg0_len + arg1_len);

       // construct the concatenated return string
       memcpy(GppcGetTextPointer(retstring), GppcGetTextPointer(arg0), arg0_len);
       memcpy(GppcGetTextPointer(retstring) + arg0_len, GppcGetTextPointer(arg1),
   arg1_len);
   ```

```
    GPPC_RETURN_TEXT( retstring );
}
```

The code declares and implements the `concat_two_strings()` function. It uses GPPC data types, macros, and functions to get the function arguments, allocate memory for the concatenated string, copy the arguments into the new string, and return the result.

5. Save the file and exit the editor.
6. Open a file named `Makefile` in the editor of your choice. Copy/paste the following text into the file:

```
MODULE_big = gppc_concat
OBJS = gppc_concat.o

PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)

PG_CPPFLAGS = -I$(shell $(PG_CONFIG) --includedir)
SHLIB_LINK = -L$(shell $(PG_CONFIG) --libdir) -lgppc
include $(PGXS)
```

7. Save the file and exit the editor.
8. Build a GPPC shared library for the `concat_two_strings()` function. For example:

```
gpadmin@gpmaster$ make all
```

The `make` command generates a shared library file named `gppc_concat.so` in the current working directory.

9. Copy the shared library to your Greenplum Database installation. You must have Greenplum Database administrative privileges to copy the file. For example:

```
gpadmin@gpmaster$ cp gppc_concat.so /usr/local/greenplum-db/lib/postgresql/
```

10. Copy the shared library to every host in your Greenplum Database installation. For example, if `seghostfile` contains a list, one-host-per-line, of the segment hosts in your Greenplum Database cluster:

```
gpadmin@gpmaster$ gpscp -v -f seghostfile /usr/local/greenplum-db/lib/postgresq
l/gppc_concat.so =:/usr/local/greenplum-db/lib/postgresql/gppc_concat.so
```

11. Open a `psql` session. For example:

```
gpadmin@gpmaster$ psql -d testdb
```

12. Register the GPPC function named `concat_two_strings()` with Greenplum Database, For example, to map the Greenplum Database function `concat_with_gppc()` to the GPPC `concat_two_strings()` function:

```
testdb=# CREATE FUNCTION concat_with_gppc(text, text) RETURNS text
  AS 'gppc_concat', 'concat_two_strings'
LANGUAGE C STRICT;
```

13. Run the `concat_with_gppc()` function. For example:

```
testdb=# SELECT concat_with_gppc( 'happy', 'monday' );
 concat_with_gppc
------------------
 happymonday
(1 row)
```

GPPC Set-Returning Function Example

In this example, you develop, build, and deploy a GPPC shared library. You also create and run a `.sql` registration script for a GPPC function named `return_tbl()`. This function uses the GPPC API to take an input table with an integer and a text column, determine if the integer column is greater than 13, and returns a result table with the input integer column and a boolean column identifying whether or not the integer is greater than 13. `return_tbl()` utilizes GPPC API reporting and SRF functions and macros.

You will develop the GPPC function on your Greenplum Database master host. Deploying the GPPC shared library that you create in this example requires administrative access to your Greenplum Database cluster.

Perform the following procedure to run the example:

1. Log in to the Greenplum Database master host and set up your environment. For example:

   ```
   $ ssh gpadmin@<gpmaster>
   gpadmin@gpmaster$ . /usr/local/greenplum-db/greenplum_path.sh
   ```

2. Create a work directory and navigate to the new directory. For example:

   ```
   gpadmin@gpmaster$ mkdir gppc_work
   gpadmin@gpmaster$ cd gppc_work
   ```

3. Prepare a source file for GPPC code by opening the file in the editor of your choice. For example, to open a file named `gppc_concat.c` using `vi`:

   ```
   gpadmin@gpmaster$ vi gppc_rettbl.c
   ```

4. Copy/paste the following code into the file:

   ```
   #include <stdio.h>
   #include <string.h>
   #include "gppc.h"

   // initialize the logging level
   GppcReportLevel level = GPPC_INFO;

   // make the function SQL-invokable and declare the function
   GPPC_FUNCTION_INFO(return_tbl);
   GppcDatum return_tbl(GPPC_FUNCTION_ARGS);

   GppcDatum
   return_tbl(GPPC_FUNCTION_ARGS)
   {
       GppcFuncCallContext fctx;
        GppcAnyTable intbl;
        GppcHeapTuple intuple;
        GppcTupleDesc in_tupdesc, out_tupdesc;
        GppcBool     resbool = false;
        GppcDatum    result, boolres, values[2];
         bool   nulls[2] = {false, false};

       // single input argument - the table
       intbl = GPPC_GETARG_ANYTABLE(0);

       // set the function context
       if (GPPC_SRF_IS_FIRSTCALL()) {
           fctx = GPPC_SRF_FIRSTCALL_INIT();
       }
       fctx = GPPC_SRF_PERCALL_SETUP();

       // get the tuple descriptor for the input table
       in_tupdesc  = GppcAnyTableGetTupleDesc(intbl);

       // retrieve the next tuple
   ```

```
    intuple = GppcAnyTableGetNextTuple(intbl);
    if( intuple == NULL ) {
      // no more tuples, conclude
      GPPC_SRF_RETURN_DONE(fctx);
    }

    // get the output tuple descriptor and verify that it is
    // defined as we expect
    out_tupdesc = GPPC_SRF_RESULT_DESC();
    if (GppcTupleDescNattrs(out_tupdesc) != 2                ||
        GppcTupleDescAttrType(out_tupdesc, 0) != GppcOidInt4 ||
        GppcTupleDescAttrType(out_tupdesc, 1) != GppcOidBool) {
        GppcReport(GPPC_ERROR, "INVALID out_tupdesc tuple");
    }

    // log the attribute names of the output tuple descriptor
    GppcReport(level, "ouput tuple descriptor attr0 name: %s", GppcTupleDescAtt
rName(out_tupdesc, 0));
    GppcReport(level, "ouput tuple descriptor attr1 name: %s", GppcTupleDescAtt
rName(out_tupdesc, 1));

    // retrieve the attribute values by name from the tuple
    bool text_isnull, int_isnull;
    GppcDatum intdat = GppcGetAttributeByName(intuple, "id", &int_isnull);
    GppcDatum textdat = GppcGetAttributeByName(intuple, "msg", &text_isnull);

    // convert datum to specific type
    GppcInt4 intarg = GppcDatumGetInt4(intdat);
    GppcReport(level, "id: %d", intarg);
    GppcReport(level, "msg: %s", GppcTextGetCString(GppcDatumGetText(textdat)))
;

    // perform the >13 check on the integer
    if( !int_isnull && (intarg > 13) ) {
        // greater than 13?
        resbool = true;
        GppcReport(level, "id is greater than 13!");
    }

    // values are datums; use integer from the tuple and
    // construct the datum for the boolean return
    values[0] = intdat;
    boolres = GppcBoolGetDatum(resbool);
    values[1] = boolres;

    // build a datum tuple and return
    result = GppcBuildHeapTupleDatum(out_tupdesc, values, nulls);
    GPPC_SRF_RETURN_NEXT(fctx, result);

}
```

The code declares and implements the `return_tbl()` function. It uses GPPC data types, macros, and functions to fetch the function arguments, examine tuple descriptors, build the return tuple, and return the result. The function also uses the SRF macros to keep track of the tuple context across function calls.

5. Save the file and exit the editor.
6. Open a file named `Makefile` in the editor of your choice. Copy/paste the following text into the file:

```
MODULE_big = gppc_rettbl
OBJS = gppc_rettbl.o

PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)

PG_CPPFLAGS = -I$(shell $(PG_CONFIG) --includedir)
SHLIB_LINK = -L$(shell $(PG_CONFIG) --libdir) -lgppc
```

```
include $(PGXS)
```

7. Save the file and exit the editor.
8. Build a GPPC shared library for the `return_tbl()` function. For example:

   ```
   gpadmin@gpmaster$ make all
   ```

   The `make` command generates a shared library file named `gppc_rettbl.so` in the current working directory.
9. Copy the shared library to your Greenplum Database installation. You must have Greenplum Database administrative privileges to copy the file. For example:

   ```
   gpadmin@gpmaster$ cp gppc_rettbl.so /usr/local/greenplum-db/lib/postgresql/
   ```

   This command copies the shared library to `$libdir`
10. Copy the shared library to every host in your Greenplum Database installation. For example, if `seghostfile` contains a list, one-host-per-line, of the segment hosts in your Greenplum Database cluster:

    ```
    gpadmin@gpmaster$ gpscp -v -f seghostfile /usr/local/greenplum-db/lib/postgresq
    l/gppc_rettbl.so =:/usr/local/greenplum-db/lib/postgresql/gppc_rettbl.so
    ```
11. Create a `.sql` file to register the GPPC `return_tbl()` function. Open a file named `gppc_rettbl_reg.sql` in the editor of your choice.
12. Copy/paste the following text into the file:

    ```
    CREATE FUNCTION rettbl_gppc(anytable) RETURNS TABLE(id int4, thirteen bool)
      AS 'gppc_rettbl', 'return_tbl'
    LANGUAGE C STRICT;
    ```
13. Register the GPPC function by running the script you just created. For example, to register the function in a database named `testdb`:

    ```
    gpadmin@gpmaster$ psql -d testdb -f gppc_rettbl_reg.sql
    ```
14. Open a `psql` session. For example:

    ```
    gpadmin@gpmaster$ psql -d testdb
    ```
15. Create a table with some test data. For example:

    ```
    CREATE TABLE gppc_testtbl( id int, msg text );
    INSERT INTO gppc_testtbl VALUES (1, 'f1');
    INSERT INTO gppc_testtbl VALUES (7, 'f7');
    INSERT INTO gppc_testtbl VALUES (10, 'f10');
    INSERT INTO gppc_testtbl VALUES (13, 'f13');
    INSERT INTO gppc_testtbl VALUES (15, 'f15');
    INSERT INTO gppc_testtbl VALUES (17, 'f17');
    ```
16. Run the `rettbl_gppc()` function. For example:

    ```
    testdb=# SELECT * FROM rettbl_gppc(TABLE(SELECT * FROM gppc_testtbl));
     id | thirteen
    ----+----------
      1 | f
      7 | f
     13 | f
     15 | t
     17 | t
     10 | f
    (6 rows)
    ```

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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.

Parent topic: Greenplum Database Reference Guide

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:

```
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) return
s int
levenshtein(text source, text target) returns int
levenshtein_less_equal(text source, text target, int ins_cost, int del_cost, int sub_c
ost, 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 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/.

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.

A newer version of this documentation is available. Use the version menu above to view the most up-to-date release of the Greenplum 5.x documentation.

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

Parent topic: Greenplum Database Reference Guide

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 9075-14:2008. 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 ABOLUTE`, 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 TABLE`s and `LOCAL TEMPORARY TABLE`s. Greenplum `TEMPORARY TABLE`s 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 TABLE`s 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:

   ```
   SELECT myfunc(args);
   ```

4. The PostgreSQL/Greenplum function definition language (`PL/PGSQL`) 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:

   ```
   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 1. SQL Support in Greenplum Database

SQL Command	Supported in Greenplum	Modifications, Limitations, Exceptions
`ALTER AGGREGATE`	YES	
`ALTER CONVERSION`	YES	
`ALTER DATABASE`	YES	
`ALTER DOMAIN`	YES	
`ALTER EXTENSION`	YES	Changes the definition of a Greenplum Database extension - based on PostgreSQL 9.6.
`ALTER FILESPACE`	YES	Greenplum Database parallel tablespace feature - not in PostgreSQL 8.3.
`ALTER FUNCTION`	YES	
`ALTER GROUP`	YES	An alias for ALTER ROLE
`ALTER INDEX`	YES	

Table 1. SQL Support in Greenplum Database

SQL Command	Supported in Greenplum	Modifications, Limitations, Exceptions
ALTER LANGUAGE	YES	
ALTER OPERATOR	YES	
ALTER OPERATOR CLASS	YES	
ALTER OPERATOR FAMILY	YES	
ALTER PROTOCOL	YES	
ALTER RESOURCE QUEUE	YES	Greenplum Database resource management feature - not in PostgreSQL.
ALTER ROLE	YES	**Greenplum Database Clauses:** RESOURCE QUEUE *queue_name* | none
ALTER SCHEMA	YES	
ALTER SEQUENCE	YES	
ALTER TABLE	YES	**Unsupported Clauses / Options:** CLUSTER ON ENABLE/DISABLE TRIGGER **Greenplum Database Clauses:** ADD | DROP | RENAME | SPLIT | EXCHANGE PARTITION | SET SUBPARTITION TEMPLATE | SET WITH (REORGANIZE=true | false) | SET DISTRIBUTED BY
ALTER TABLESPACE	YES	
ALTER TRIGGER	**NO**	
ALTER TYPE	YES	
ALTER USER	YES	An alias for ALTER ROLE
ALTER VIEW	YES	
ANALYZE	YES	
BEGIN	YES	
CHECKPOINT	YES	
CLOSE	YES	
CLUSTER	YES	
COMMENT	YES	
COMMIT	YES	
COMMIT PREPARED	**NO**	
COPY	YES	**Modified Clauses:** ESCAPE [AS] '*escape*' | 'OFF' **Greenplum Database Clauses:** [LOG ERRORS] SEGMENT REJECT LIMIT *count* [ROWS|PERCENT]

Table 1. SQL Support in Greenplum Database

SQL Command	Supported in Greenplum	Modifications, Limitations, Exceptions
CREATE AGGREGATE	YES	**Unsupported Clauses / Options:** [, SORTOP = *sort_operator*] **Greenplum Database Clauses:** [, PREFUNC = *prefunc*] **Limitations:** The functions used to implement the aggregate must be IMMUTABLE functions.
CREATE CAST	YES	
CREATE CONSTRAINT TRIGGER	**NO**	
CREATE CONVERSION	YES	
CREATE DATABASE	YES	
CREATE DOMAIN	YES	
CREATE EXTENSION	YES	Loads a new extension into Greenplum Database - based on PostgreSQL 9.6.
CREATE EXTERNAL TABLE	YES	Greenplum Database parallel ETL feature - not in PostgreSQL 8.3.
CREATE FUNCTION	YES	**Limitations:** Functions defined as STABLE or VOLATILE can be executed in Greenplum Database provided that they are executed on the master only. STABLE and VOLATILE functions cannot be used in statements that execute at the segment level.
CREATE GROUP	YES	An alias for CREATE ROLE
CREATE INDEX	YES	**Greenplum Database Clauses:** 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. CONCURRENTLY keyword not supported in Greenplum.
CREATE LANGUAGE	YES	
CREATE OPERATOR	YES	**Limitations:** The function used to implement the operator must be an IMMUTABLE function.
CREATE OPERATOR CLASS	YES	
CREATE OPERATOR FAMILY	YES	
CREATE PROTOCOL	YES	
CREATE RESOURCE QUEUE	YES	Greenplum Database resource management feature - not in PostgreSQL 8.3.
CREATE ROLE	YES	**Greenplum Database Clauses:** RESOURCE QUEUE *queue_name* | none

Table 1. SQL Support in Greenplum Database

SQL Command	Supported in Greenplum	Modifications, Limitations, Exceptions
`CREATE RULE`	YES	
`CREATE SCHEMA`	YES	
`CREATE SEQUENCE`	YES	**Limitations:** The `lastval()` and `currval()` functions are not supported. The `setval()` function is only allowed in queries that do not operate on distributed data.

`CREATE TABLE`	YES	**Unsupported Clauses / Options:**
		`[GLOBAL | LOCAL]`
		`REFERENCES`
		`FOREIGN KEY`
		`[DEFERRABLE | NOT DEFERRABLE]`
		Limited Clauses:
		`UNIQUE` or `PRIMARY KEY` constraints are only allowed on hash-distributed tables (`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.
		Greenplum Database Clauses:
		`DISTRIBUTED BY (`*`column`*`, [ ... ] ) |`
		`DISTRIBUTED RANDOMLY`
		`PARTITION BY `*`type`*` (`*`column`*` [, ...])  (`*`partition_specification`*`, [...] )`
		`WITH (appendonly=true [,compresslevel=value,blocksize=value] )`

`CREATE TABLE AS`	YES	See CREATE TABLE
`CREATE TABLESPACE`	NO	**Greenplum Database Clauses:** `FILESPACE` *filespace_name*
`CREATE TRIGGER`	NO	
`CREATE TYPE`	YES	**Limitations:** The functions used to implement a new base type must be `IMMUTABLE` functions.
`CREATE USER`	YES	An alias for CREATE ROLE
`CREATE VIEW`	YES	
`DEALLOCATE`	YES	

`DECLARE`	YES	**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.

Table 1. SQL Support in Greenplum Database

SQL Command	Supported in Greenplum	Modifications, Limitations, Exceptions
DELETE	YES	**Unsupported Clauses / Options:** RETURNING
DISCARD	YES	**Limitation:** DISCARD ALL is not supported.
DO	YES	PostgreSQL 9.0 feature
DROP AGGREGATE	YES	
DROP CAST	YES	
DROP CONVERSION	YES	
DROP DATABASE	YES	
DROP DOMAIN	YES	
DROP EXTENSION	YES	Removes an extension from Greenplum Database – based on PostgreSQL 9.6.
DROP EXTERNAL TABLE	YES	Greenplum Database parallel ETL feature - not in PostgreSQL 8.3.
DROP FILESPACE	YES	Greenplum Database parallel tablespace feature - not in PostgreSQL 8.3.
DROP FUNCTION	YES	
DROP GROUP	YES	An alias for DROP ROLE
DROP INDEX	YES	
DROP LANGUAGE	YES	
DROP OPERATOR	YES	
DROP OPERATOR CLASS	YES	
DROP OPERATOR FAMILY	YES	
DROP OWNED	**NO**	
DROP PROTOCOL	YES	
DROP RESOURCE QUEUE	YES	Greenplum Database resource management feature - not in PostgreSQL 8.3.
DROP ROLE	YES	
DROP RULE	YES	
DROP SCHEMA	YES	
DROP SEQUENCE	YES	
DROP TABLE	YES	
DROP TABLESPACE	**NO**	
DROP TRIGGER	**NO**	
DROP TYPE	YES	
DROP USER	YES	An alias for DROP ROLE
DROP VIEW	YES	
END	YES	
EXECUTE	YES	

Table 1. SQL Support in Greenplum Database

SQL Command	Supported in Greenplum	Modifications, Limitations, Exceptions
EXPLAIN	YES	
FETCH	YES	**Unsupported Clauses / Options:** LAST PRIOR BACKWARD BACKWARD ALL **Limitations:** Cannot fetch rows in a nonsequential fashion; backward scan is not supported.
GRANT	YES	
INSERT	YES	**Unsupported Clauses / Options:** RETURNING
LISTEN	**NO**	
LOAD	YES	
LOCK	YES	
MOVE	YES	See FETCH
NOTIFY	**NO**	
PREPARE	YES	
PREPARE TRANSACTION	**NO**	
REASSIGN OWNED	YES	
REINDEX	YES	
RELEASE SAVEPOINT	YES	
RESET	YES	
REVOKE	YES	
ROLLBACK	YES	
ROLLBACK PREPARED	**NO**	
ROLLBACK TO SAVEPOINT	YES	
SAVEPOINT	YES	
SELECT	YES	**Limitations:** Limited use of VOLATILE and STABLE functions in FROM or WHERE clauses Text search (Tsearch2) is not supported FETCH FIRST or FETCH NEXT clauses not supported **Greenplum Database Clauses (OLAP):** [GROUP BY *grouping_element* [, ...]] [WINDOW *window_name* AS (*window_specification*)] [FILTER (WHERE *condition*)] applied to an aggregate function in the SELECT list
SELECT INTO	YES	See SELECT

Table 1. SQL Support in Greenplum Database

SQL Command	Supported in Greenplum	Modifications, Limitations, Exceptions
SET	YES	
SET CONSTRAINTS	**NO**	In PostgreSQL, this only applies to foreign key constraints, which are currently not enforced in Greenplum Database.
SET ROLE	YES	
SET SESSION AUTHORIZATION	YES	Deprecated as of PostgreSQL 8.1 - see SET ROLE
SET TRANSACTION	YES	
SHOW	YES	
START TRANSACTION	YES	
TRUNCATE	YES	
UNLISTEN	**NO**	
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	