Greenplum
The Data Computing Division of EMC

EMC²
where information lives®

Greenplum® Performance Monitor 4.0 Administrator Guide

P/N: 300-011-542
Rev: A02

Greenplum Performance Monitor Administrator Guide 4.0 — Contents

Preface

This guide provides information about the Greenplum Performance Monitor, an optional feature of Greenplum Database used to collect, store, and analyze system and query performance metrics.

About This Guide

This guide provides information and instructions for installing and using Greenplum Performance Monitor. It also provides installation instructions for Greenplum Performance Monitor Console (the graphical web application for viewing Performance Monitor data). For information about using the Console and details about the system and query metrics you can view with this optional web-based application, click the Help link in the Console user interface.

This guide assumes knowledge of Linux/Unix system administration, database management systems, database administration, and structured query language (SQL).

- Chapter 1, "About Greenplum Performance Monitor" explains the general architecture and functionality of Greenplum Performance Monitor.
- Chapter 2, "Setting Up Greenplum Performance Monitor" provides instructions for setting up the data collection agents within Greenplum Database, and for installing the web application.
- Chapter 3, "Administering Performance Monitor" provides information on system, server, and database administration of a Greenplum Performance Monitor installation.
- Appendix A, "Utility Reference" is a reference for the setup and management utilities of Greenplum Performance Monitor (data collection agents and web application).
- Appendix B, "Configuration File Reference" is a reference of the configuration parameters for Greenplum Performance Monitor.
- Appendix C, "Performance Monitor Database Reference" is a reference of the `gpperfmon` database schema.

Document Conventions

The following conventions are used throughout the Greenplum Database documentation to help you identify certain types of information.

Text Conventions

Table 0.1 Text Conventions

Text Convention	Usage	Examples
bold	Button, menu, tab, page, and field names in GUI applications	Click **Cancel** to exit the page without saving your changes.
italics	New terms where they are defined Database objects, such as schema, table, or columns names	The *master instance* is the `postgres` process that accepts client connections. Catalog information for Greenplum Database resides in the *pg_catalog* schema.
`monospace`	File names and path names Programs and executables Command names and syntax Parameter names	Edit the `postgresql.conf` file. Use `gpstart` to start Greenplum Database.
`monospace italics`	Variable information within file paths and file names Variable information within command syntax	`/home/gpadmin/`*`config_file`* `COPY` *`tablename`* `FROM '`*`filename`*`'`
`monospace bold`	Used to call attention to a particular part of a command, parameter, or code snippet.	Change the host name, port, and database name in the JDBC connection URL: `jdbc:postgresql://`**`host:5432/mydb`**
`UPPERCASE`	Environment variables SQL commands Keyboard keys	Make sure that the Java `/bin` directory is in your `$PATH`. `SELECT * FROM` *`my_table`*`;` Press `CTRL+C` to escape.

Command Syntax Conventions

Table 0.2 Command Syntax Conventions

Text Convention	Usage	Examples
`{ }`	Within command syntax, curly braces group related command options. Do not type the curly braces.	`FROM { 'filename' | STDIN }`
`[ ]`	Within command syntax, square brackets denote optional arguments. Do not type the brackets.	`TRUNCATE [ TABLE ] name`
`...`	Within command syntax, an ellipsis denotes repetition of a command, variable, or option. Do not type the ellipsis.	`DROP TABLE name [, ...]`
`|`	Within command syntax, the pipe symbol denotes an "OR" relationship. Do not type the pipe symbol.	`VACUUM [ FULL | FREEZE ]`

`$ system_command`
`# root_system_command`
`=> gpdb_command`
`=# su_gpdb_command`
	Denotes a command prompt - do not type the prompt symbol. `$` and `#` denote terminal command prompts. `=>` and `=#` denote Greenplum Database interactive program command prompts (`psql` or `gpssh`, for example).	`$ createdb mydatabase`
		`# chown gpadmin -R /datadir`
		`=> SELECT * FROM mytable;`
		`=# SELECT * FROM pg_database;`

Getting Support

EMC support, product, and licensing information can be obtained as follows.

Product information

For documentation, release notes, software updates, or for information about EMC products, licensing, and service, go to the EMC Powerlink website (registration required) at:

http://Powerlink.EMC.com

Technical support

For technical support, go to Powerlink and choose **Support**. On the Support page, you will see several options, including one for making a service request. Note that to open a service request, you must have a valid support agreement. Please contact your EMC sales representative for details about obtaining a valid support agreement or with questions about your account.

1 About Greenplum Performance Monitor

Greenplum Performance Monitor is an optional feature that administrators can enable within Greenplum Database. Greenplum Performance Monitor allows administrators to collect query and system performance metrics from a running Greenplum Database system. Monitor data is stored within Greenplum Database. The Greenplum Performance Monitor Console (a graphical web application) can optionally be installed and used to view the collected performance monitor data.

Greenplum Performance Monitor is comprised of data collection agents that run on the master host and each segment host. The agents collect performance data about active queries and system utilization and send it to the Greenplum master at regular intervals. The data is stored in a dedicated database on the master, where it can be accessed through the Greenplum Performance Monitor Console or through SQL queries.

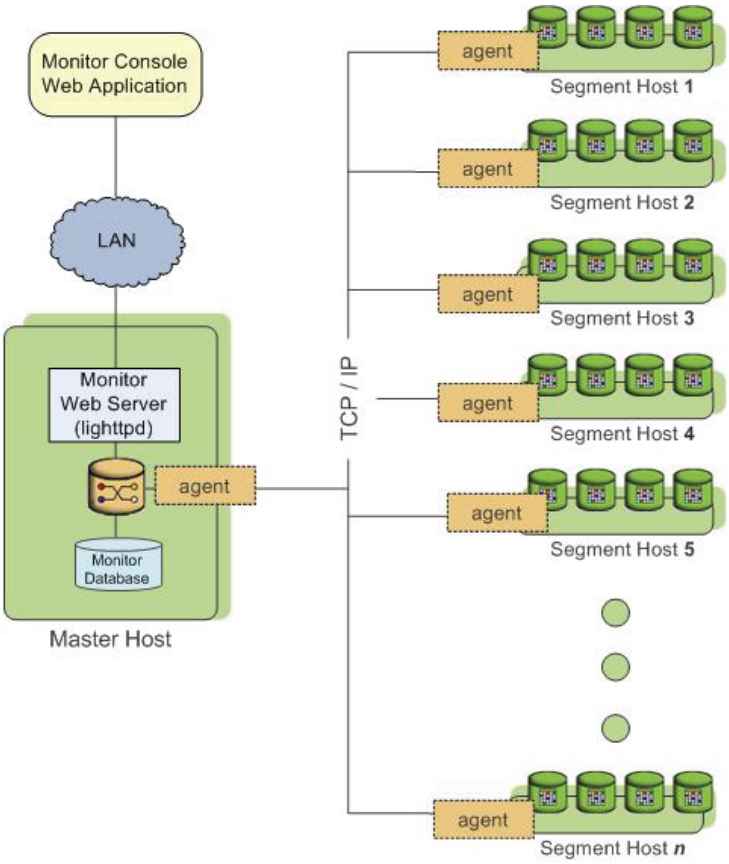

Figure 1.1 Greenplum Performance Monitor Architecture

Greenplum Performance Monitor Agents

The collection of query and system performance metrics is an optional feature of Greenplum Database that can be enabled or disabled using the `gp_enable_gpperfmon` server configuration parameter. Once enabled, the Greenplum Performance Monitor data collection agents run on all Greenplum Database hosts (master and segments), and are started and stopped along with the Greenplum Database server processes.

The master agent polls all segment agents for performance data at a configurable interval (known as the `quantum`). The master agent pulls together the data from all segments, stores it in flat files, and periodically commits the data in the files to the Greenplum Performance Monitor database.

Greenplum Performance Monitor Database

The Greenplum Performance Monitor database (`gpperfmon`) is a database within your Greenplum system dedicated to storing and serving performance data. Your Greenplum Database installation includes setup scripts to install the `gpperfmon` database.

Greenplum administrators can connect to the monitor database using client programs such as `psql` or application programming interfaces (APIs) such as JDBC and ODBC. Administrators can also use the Greenplum Performance Monitor console to view reports on current and historical performance.

The database consists of three sets of tables; *now* tables store data on currently active queries, *history* tables store data on completed queries, and *tail* tables are for data in transition. The *now* and *tail* data are stored as text files on the master file system, and accessed by Greenplum Database via external tables. The *history* tables are regular database tables stored within the `gpperfmon` database. See Appendix C, "Performance Monitor Database Reference" for the schema definitions of these tables.

Greenplum Performance Monitor Console

Greenplum Performance Monitor provides a graphical console for viewing performance metrics. With this browser-based application, you can track the performance of active queries and view historical data for queries and system metrics.

Figure 1.2 Greenplum Performance Monitor Dashboard

If you have multiple Greenplum Database instances, you can create separate monitor instances for them. Each separate console instance operates on a unique port and has its own unique configuration options. For more information, see "Installing the Greenplum Performance Monitor Console" on page 9.

Greenplum Performance Monitor Web Service

The Greenplum Performance Monitor Console queries the monitor database through a web service framework composed of a lightweight lighttpd web server and Python-based middleware. The lighttpd server is an open-source web server with a low memory footprint and light CPU load. For more information, see http://www.lighttpd.net/.

The console setup utility sets up the lighttpd web server and web service, prompting you for basic configuration information on the desired port and SSL options. Under normal conditions, the web server and web service API require minimal maintenance and administration, as described in "Web Server Administration" on page 21.

2 Setting Up Greenplum Performance Monitor

Installing and enabling Greenplum Performance Monitor is a two-part process. First you must create the Greenplum Performance Monitor database and enable Greenplum Performance Monitor data collection agents within Greenplum Database. After data collection is enabled in Greenplum Database, the next (optional) step is to install and configure the Greenplum Performance Monitor Console. (the Web application used to view the performance monitor data stored in Greenplum Database). The Greenplum Performance Monitor Console is shipped separately from your Greenplum Database installation.

Enabling the Performance Monitor Data Collection Agents

This section describes how to create the Performance Monitor database and enable the Performance Monitor data collection agents. When the data collection agents are enabled, their processes are started and stopped along with the Greenplum Database server processes (using `gpstart` and `gpstop`).

Greenplum provides a `gpperfmon_install` utility that performs the following tasks:

- Creates the Performance Monitor database (`gpperfmon`).
- Creates the Performance Monitor superuser role (`gpmon`).
- Configures Greenplum Database to accept connections from the Performance Monitor superuser role (edits the `pg_hba.conf` and `.pgpass` files).
- Sets the Performance Monitor server configuration parameters in the Greenplum Database `postgresql.conf` files.

Install the Performance Monitor database and enable Performance Monitor agents

1. Log in to the Greenplum Database master as the `gpadmin` user.

   ```
   $ su - gpadmin
   ```

2. Run the `gpperfmon_install` utility with the `--enable` option. You must supply the connection port of the Greenplum Database master, and set the password for the `gpmon` superuser that will be created. For example:

   ```
   $ gpperfmon_install --enable --password p@$$word --port 5432
   ```

3. When the utility completes, restart Greenplum Database. The data collection agents will not start until the database is restarted:

   ```
   $ gpstop -r
   ```

4. Using the `ps` command, verify that the data collection process is running on the Greenplum master. For example:

   ```
   $ ps -ef | grep gpmmon
   ```

5. Run the following command to verify that the data collection processes are writing to the Performance Monitor database. If all of the segment data collection agents are running, you should see one row per segment host.

   ```
   $ psql gpperfmon -c 'SELECT * FROM system_now;'
   ```

The data collection agents are now running, and your Greenplum Database system now has a `gpperfmon` database installed. This is the database where performance data is stored. You can connect to it as follows:

```
$ psql gpperfmon
```

To configure a standby master host (if enabled)

6. Copy the `$MASTER_DATA_DIRECTORY/pg_hba.conf` file from your primary master host to your standby master host. This ensures that the required connection options are also set on the standby master.

7. Copy your `~/.pgpass` file from your primary master host to your standby master host. This file usually resides in the `gpadmin` user's home directory.

Installing the Greenplum Performance Monitor Console

The Performance Monitor Console provides a graphical interface for viewing performance data and for administering certain aspects of monitoring. Normally installed on the Greenplum Database master host, the console is a web-based user interface accessed through a supported browser.

The Performance Monitor Console is typically installed on the Greenplum Database master host. However, you have the option to install the console on a host different from the master host. Note that this setup incurs a performance penalty due to the numerous database connections the console must open over the network.

If you have multiple Greenplum Database instances, you can create separate monitor instances for each of them. Each separate console instance operates on a unique port and has its own unique configuration options.

The Performance Monitor Console is supported for the following browsers with Adobe Flash 9.0 or higher:

- Internet Explorer for Windows XP and Vista
- Mozilla Firefox for Windows and Linux
- Apple's Safari browser for Macintosh

The Performance Monitor Console runs on a lighttpd web server. The default web server port is 28080. For more information about the web server, see "Web Server Administration" on page 21.

Installing the Performance Monitor Console involves the following high-level tasks:

- Install Performance Monitor Console — Create the software installation directory.
- Set up Greenplum Performance Monitor Console — Create and configure a Performance Monitor Console instance and its supporting web services.

Install Performance Monitor Console

1. Download the installer file from Greenplum Network. Installer files are available for RedHat (32-bit and 64-bit), Solaris 64-bit or SuSE Linux 64 bit platforms.
2. Unzip the installer file where *PLATFORM* is either `RHEL5-i386` (RedHat 32-bit), `RHEL5-x86_64` (RedHat 64-bit), `SOL-x86_64` (Solaris 64-bit), or `SuSE10-x86_64` (SuSe Linux 64 bit). For example:

   ```
   # unzip greenplum-perfmon-web-4.0.x.x-PLATFORM.zip
   ```
3. Launch the installer using `bash`. For example:

   ```
   # /bin/bash greenplum-perfmon-web-4.0.x.x-PLATFORM.bin
   ```
4. Type `yes` to accept the license agreement.
5. The installer will prompt you to provide an installation path. Press `ENTER` to accept the default install path:
   (`/usr/local/greenplum-perfmon-web-4.0.x.x`)
   or enter an absolute path to an install location. You must have write permissions to the location you specify. This location is referred to as `$GPPERFMONHOME`.
6. If you ran the installer as `root`, change the ownership of the Console installation directory to the `gpadmin` user. For example:

   ```
   # chown -R gpadmin /usr/local/greenplum-perfmon-web-4.0.x.x
   ```
7. The installation directory contains a `gpperfmon_path.sh` file with path and environment settings for the Console. You must source this file in your `gpadmin` user's startup shell profile (such as `.bashrc`).

   For example, you could add a line similar to the following to your chosen profile files:

   ```
   source /usr/local/greenplum-perfmon-web-4.0.x.x/ \
   gpperfmon_path.sh
   ```

   After editing the chosen profile file, source it as the correct user to make the changes active. For example:

   ```
   $ source ~/.bashrc
   ```
8. Configure the Console as described in "Setting up the Performance Monitor Console".

Setting up the Performance Monitor Console

The `gpperfmon` utility sets up the Performance Monitor Console on the current host. On hosts other than the Greenplum Database master host, the console experiences slower performance due to frequent connections to the `gpperfmon` database.

During the setup process, the utility prompts you for values to configure Console connections to a single Greenplum Database instance. To configure connections to multiple Greenplum Database instances, run the setup routine multiple times. To accept the displayed default values for any of these parameters at configuration time, hit the `ENTER` key.

Set up Greenplum Performance Monitor Console

1. Log in as the Greenplum administrator (`gpadmin`).
2. With the Greenplum Database running, launch the setup utility. For example:
   ```
   $ gpperfmon --setup
   ```
3. Provide an instance name for the Greenplum Database instance monitored by this Console. To monitor multiple instances, you must run the setup utility separately for each instance.
4. Select `y` or `n` to specify if the Greenplum Database master for this instance is on a remote host. Console performance is better when the Console and Greenplum Database master are on the same host.

   If the master host is remote, enter `y` and enter the hostname of the master at the prompt.

Note: Verify that you can connect to the master host from the host on which you are installing the Performance Monitor Console. Enter:

```
psql -h master_host_name -p port -U gpmon
```

If you cannot establish a connection, make sure you create a `.pgpass` file on the host running the Performance Monitor Console as described in "Configuring Authentication for the Monitor Console" on page 13.

5. Provide a display name for the instance. This name is shown in the Console user interface. This prompt does not display if the master host is remote.
6. Provide the port for the Greenplum Database master instance.
7. Provide a port number for the Performance Monitor Console web server. The default is `28080`.
8. (Optional) Enter `y` or `n` to set up SSL connections for the Performance Monitor Console. If you enter `Y`, you are prompted for the following distinguished name (DN) information used to create an unsigned certificate for the Performance Monitor Console. For example:

   ```
   Country Name (2 letter code) [GB]:US
   State or Province Name (full name) [Berkshire]:California
   Locality Name (eg, city) [Newbury]:San Mateo
   Organization Name (eg, company) [My Company Ltd]:Greenplum
   Organizational Unit Name (eg, section) []:Engineering
   Common Name (eg, your name or your server's hostname) []:mdw1
   ```

```
Email Address []:gpadmin@greenplum.com
```

Note: Because database login information is sent over the network, Greenplum recommends using SSL to encrypt these communications.

9. Start and log into the Console. See "Connecting to the Greenplum Performance Monitor Console".

   You can also configure authentication so that other database users can log into the Console, as described in "Configuring Authentication for the Monitor Console" on page 13.

Connecting to the Greenplum Performance Monitor Console

Start the Greenplum Performance Monitor Console by entering:

```
gpperfmon --start
```

If you do not specify an instance name, all Performance Monitor Console instances are started. To start a particular instance, you can specify the name of the instance. For example:

```
gpperfmon --start "instance_name"
```

See "Starting and Stopping Greenplum Performance Monitor" on page 19 for a complete list of commands.

After the instance is running, you can open the Performance Monitor Console in a supported browser using the correct hostname and port. For example, to open a monitor instance running on port 28080 on the local host with SSL, use the following web address:

```
https://localhost:28080/
```

If connecting to a monitor instance using SSL on port 28080, enter:

```
https://monitor_console_hostname:28080/
```

At the login prompt, enter the user name and password of a Greenplum Database role that has been properly configured to allow authentication to Greenplum Performance Monitor (for example, `gpmon`), then click **Login**. This opens the **Dashboard** page of the Performance Monitor Console, which provides a graphical system snapshot and a summary view of active queries. See the Performance Monitor Console online help for more information.

You must be a Greenplum Database administrator to fully use the Greenplum Performance Monitor Console. Administrators can view information for all queries as well as system metrics, while regular database users can only monitor their own queries.

Configuring Authentication for the Monitor Console

When you installed Greenplum Performance Monitor database and enabled the data collection agents, a `gpmon` superuser was created by the installation utility. This is the database role used to manage the Performance Monitor components and data within Greenplum Database. The `gpmon` role is configured to use md5-encrypted password authentication to connect to Greenplum Database.

Typically, you will not be connecting to the Performance Monitor Console as `gpmon`, and instead connect as another database user (such as `gpadmin`). The Performance Monitor Console is configured by default to require md5-encrypted password authentication for all database users who want to log into the Performance Monitor Console.

To set up md5-encrypted password authentication for a database user who will login to the Console, do the following steps:

1. Make sure the database role has an md5-encrypted password set. Passwords can be assigned using `CREATE ROLE` or `ALTER ROLE`. For example:

   ```
   =# ALTER ROLE gpadmin WITH ENCRYPTED PASSWORD
   'password1234';
   ```

2. Edit `$MASTER_DATA_DIRECTORY/pg_hba.conf` to specify `md5` authentication for connections made by any role to the `gpperfmon` database. For example:

   ```
   host     gpperfmon     all     0.0.0.0/0     md5
   ```

3. In the `gpadmin` user's home directory on the master host, add an entry to the `.pgpass` file that has the following connection information:

   master_hostname`:`*master_port*`:`*dbname*`:`*rolename*`:`*password*

   For example (where * means any configured master hostname):

   ```
   *:5432:gpperfmon:gpadmin:password1234
   ```

4. Save and close the `.pgpass` file.

5. Use `gpstop` to reload the `pg_hba.conf` file:

   ```
   $ gpstop -u
   ```

Using Trust Authentication

You can bypass password authentication by using `trust` authentication. Trust authentication allows database users to login to the Performance Monitor Console without authenticating, and is recommended only for testing purposes. It should not be used in production Greenplum Database environments. Greenplum recommends using password authentication with SSL for production installations.

To allow trust access

1. Set `allow_trust_logon` to `yes` in the `gpperfmonui.conf` file (Greenplum Performance Monitor configuration file) located in:

   `$GPPERFMONHOME/instances/`*instance_name*

   where *instance_name* is the name of the instance that you specified when you ran `gpperfmon --setup`.

2. Edit `$MASTER_DATA_DIRECTORY/pg_hba.conf` to specify `trust` authentication for connections made by any role to the `gpperfmon` database. For example:

   ```
   host     gpperfmon     all     0.0.0.0/0     trust
   ```

3. Use `gpstop` to reload the `pg_hba.conf` file:

   ```
   $ gpstop -u
   ```

Using SSL Encryption

If you enable SSL at setup time, the installer creates a self-signed certificate and uses OpenSSL encryption for all connections to the monitor web server.

Because this is a self-signed certificate, supported browsers may need to be configured to accept or allow it as a security exception. This does not detract from the added security gained from encrypting communications between the browser client and the web server.

Optionally, you can obtain and use a certificate signed by a trusted certificate authority such as Verisign. If you use a trusted certificate, edit the `lighttpd.conf` file (located in `$GPPERFMONHOME/instances/`*`instance_name`*`/conf`), and set the `ssl.pemfile` to point to the location of the certificate. For example:

```
ssl.pemfile = "$GPPERFMONHOME/instances/instance_name/conf/cert.pem"
```

For more information on the `lighttpd.conf` file, see "Web Server Administration" on page 21.

Setting up Performance Monitor Console on a Standby Master

1. Install the Performance Monitor Console software on the standby master host using the instructions in "Installing the Greenplum Performance Monitor Console" on page 9.

2. Set up a new Performance Monitor Console instance on the master host using the instructions in "Setting up the Performance Monitor Console" on page 10. During the setup process, make sure to specify the following options specific to the standby master:

   - Specify a *different* instance name than the console instance running on the primary master host.
   - When prompted for the Greenplum Performance Monitor port, specify the *same* value as the primary master (the standby master must always use the same port as the primary).
   - If you configured the Greenplum Performance Monitor Console for SSL connections, make sure the `ssl.pemfile` parameter in the web server configuration file points to a valid certificate for the standby master host. You cannot use the same certificate file as the primary master host.

3. If you fail over to your standby master host, the data collection agents are restarted automatically upon activation of the standby. However, you must manually start the Performance Monitor Console using `gpperfmon --start`.

About the Performance Monitor Installation

The installation and setup procedures create a software installation directory and a directory containing files and folders to support each Greenplum Performance Monitor Console instance.

Software Installation Directory

The following files and first-level subdirectories are copied into the installation folder that you specified when you installed Greenplum Performance Monitor Console. This location is referred to as `$GPPERFMONHOME`.

- `gpperfmon_path.sh` — file containing environment variables for the monitor
- **bin** — program files for Greenplum Performance Monitor
- **docs** — documentation, including administration guide and release notes
- **etc** — contains `openssl.conf` file
- **ext** — Python directory and files
- **instances** — contains a subdirectory of resources for each Greenplum Database instance monitored by the console
- **lib** — library files for Greenplum Performance Monitor
- **www** — web service and user interface files

Instances Directory

The `$GPPERFMONHOME/instances` directory contains subdirectories named for each instance created during console setup. The `conf` subdirectory contains configuration files that you can edit. Other files and folders are used by the web services for the instance, and should not be modified or deleted.

Each subdirectory contains the following files and first-level subdirectories:

- `lighttpd.pid` -- file containing an updated process ID for the web server process for the instance
- **conf** — console and web server configuration files, `gpperfmonui.conf` and `lighttpd.conf`
- **logs** — logs for the web server for this instance
- `perfmon.fastcgi.socket-0` — dynamically updated socket information, which cannot be viewed or updated
- **sessions** — files storing information about session state
- **tmp** — temporary files used by the web server
- **web** — symbolic links to web server files in the installation directory

Querying the Performance Monitor Database

Greenplum Performance Monitor database stores system and query data collected by Performance Monitor agents that run on the master host and each segment host. This information provides immediate visibility into key areas affecting database performance, and is typically accessed through the Greenplum Performance Monitor Console.

However, you can also use direct SQL queries and views on performance data. Before performing any of the tasks in this section, make sure the Greenplum Performance Monitor is complete installed and configured, as described in "Installing the Greenplum Performance Monitor Console" on page 9, and "Setting up the Performance Monitor Console" on page 10.

Viewing Historical System Metrics using SQL

Historical tables in the monitor database store all system data collected by the monitor. You can analyze this data to identify trends in the performance of the Greenplum Database system over time.

With a default `quantum` setting of 15 seconds, the monitor collects very large amounts of historical system data. To view meaningful groupings of system data records, you can view the data by selected time intervals with a SQL view (as described below).

The following provides an example SQL view for viewing aggregated system metrics. In this example view named *system_metrics_aggregate_1min*, rows of system metrics from *system_history* are averaged over one-minute intervals and aggregated across all segments in the array:

```
DROP VIEW IF EXISTS system_metrics_aggregate_1min;

CREATE VIEW system_metrics_aggregate_1min AS (
    SELECT date_trunc('minute', ctime) AS sample_time,
    hostname, avg(mem_total) AS mem_total,
    avg(mem_used) AS mem_used,
    avg(mem_actual_used) AS mem_actual_used,
    avg(mem_actual_free) AS mem_actual_free,
    avg(swap_total) AS swap_total, avg(swap_used) AS swap_used,
    avg(swap_page_in) AS swap_page_in,
    avg(swap_page_out) AS swap_page_out,
    avg(cpu_user) AS cpu_user, avg(cpu_sys) AS cpu_sys,
    avg(cpu_idle) AS cpu_idle,
    avg(load0) AS load0, avg(load1) AS load1,
    avg(load2) AS load2,
    avg(disk_ro_rate) AS disk_ro_rate,
    avg(disk_wo_rate) AS disk_wo_rate,
    avg(disk_rb_rate) AS disk_rb_rate,
    avg(disk_wb_rate) AS disk_wb_rate,
    avg(net_rp_rate) AS net_rp_rate,
    avg(net_wp_rate) AS net_wp_rate,
    avg(net_rb_rate) AS net_rb_rate,
    avg(net_wb_rate) AS net_wb_rate

    FROM system_history
```

```
    GROUP BY hostname, sample_time
);
```

To change the interval for averaging results, adjust the value specified for `date_trunc`. Valid values are `microseconds`, `milliseconds`, `second`, `minute`, `hour`, `day`, `week`, `month`, `quarter`, `year`, `decade`, `century` and `millennium`.

Note: You use can also the Performance Monitor Console to view historical system metric. See the Performance Monitor Console online help for more information.

Viewing Historical Query Metrics using SQL

Historical tables in the monitor database store all query and iterator data collected by the monitor. You can analyze this data to identify trends in the performance of Greenplum Database over time.

Calculating the average, minimum and maximum values for system-wide query statistics aggregated over time can yield a useful view of database utilization. The following information provides an example SQL view for viewing query statistics in this way.

In this example view named *database_metrics_1hour*, rows of statistics from the table *database_history* are averaged over hour-long intervals:

```
DROP VIEW if exists database_metrics_1hour;

CREATE VIEW database_metrics_1hour AS (

    SELECT date_trunc('hour', ctime) AS sample_time,
    avg(queries_total) AS queries_total,
    min(queries_total) AS queries_total_min,
    max(queries_total) AS queries_total_max,
    avg(queries_running) AS queries_running,
    min(queries_running) AS queries_running_min,
    max(queries_running) AS queries_running_max,
    avg(queries_queued) AS queries_queued,
    min(queries_queued) AS queries_queued_min,
    max(queries_queued) AS queries_queued_max

    FROM database_history

    GROUP BY sample_time );
```

To change the interval for averaging results, adjust the value specified for `date_trunc`. Valid values are `microseconds`, `milliseconds`, `second`, `minute`, `hour`, `day`, `week`, `month`, `quarter`, `year`, `decade`, `century` and `millennium`.

Note: You use can also the Performance Monitor Console to view historical query metric. See the Performance Monitor Console online help for more information.

Uninstalling Greenplum Performance Monitor

To uninstall, you must stop both the Performance Monitor Console and disable the Performance Monitor data collection agents. Optionally, you may also want to remove any data associated with Greenplum Performance Monitor by removing your Performance Monitor Console installation and the Performance Monitor database.

To uninstall the Performance Monitor Console

1. Stop Performance Monitor Console if it is currently running. For example:
   ```
   $ gpperfmon --stop
   ```
2. Remove the monitor installation directory. For example:
   ```
   $ rm -rf /usr/local/greenplum-perfmon-web-4.0.x.x
   ```

To disable the Performance Monitor Agents

1. Log in to the master host as the Greenplum administrative user (`gpadmin`):
   ```
   su - gpadmin
   ```
2. Edit `$MASTER_DATA_DIRECTORY/postgresql.conf` and disable the Performance Monitor data collection agents:
   ```
   gp_enable_gpperfmon = off
   ```
3. Remove or comment out the `gpmon` entries in `pg_hba.conf`. For example:
   ```
   #local      gpperfmon      gpmon      md5
   #host       gpperfmon      gpmon    0.0.0.0/0    md5
   ```
4. Drop the monitor superuser role from the database. For example:
   ```
   $ psql template1 -c 'DROP ROLE gpmon;'
   ```
5. Restart Greenplum Database:
   ```
   $ gpstop -r
   ```
6. Clean up any uncommitted monitor data and log files that reside on the master file system:
   ```
   $ rm $MASTER_DATA_DIRECTORY/gpperfmon/data/*
   $ rm $MASTER_DATA_DIRECTORY/gpperfmon/logs/*
   ```
7. If you do not want to keep your historical Performance Monitor data, drop the `gpperfmon` database:
   ```
   $ dropdb gpperfmon
   ```

3 Administering Performance Monitor

After installation and startup, Greenplum Performance Monitor requires minimal maintenance by administrators. Tasks for administration of the web server or the monitor database are performed with native tools or Performance Monitor utilities, as described in this chapter.

Starting and Stopping Greenplum Performance Monitor

Starting and Stopping the Monitor Agents

Whenever the Greenplum Database server configuration parameter `gp_enable_gpperfmon` is enabled in the master `postgresql.conf` file, the Performance Monitor agents will run and collect data. The monitor agents are automatically stopped and started together along with Greenplum Database.

If you wish to disable the Performance Monitor data collection agents, you must disable the gp_enable_gpperfmon parameter, and restart Greenplum Database.

Starting and Stopping the Monitor Console

You can start, stop and restart Greenplum Performance Monitor Console instances with the following commands:

```
$ gpperfmon --start ["instance name"]
$ gpperfmon --stop ["instance name"]
$ gpperfmon --restart ["instance name"]
```

If you do not specify an instance name, all instances are started, stopped or restarted at once. You can check the status of instances using:

```
$ gpperfmon --status ["instance name"]
```

Configuring Greenplum Performance Monitor

Configuration parameters for Greenplum Performance Monitor are stored in the following configuration files.

Agent Configuration

Changes to these files require a restart of Greenplum Database (`gpstop -r`).

- `$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf`
- `$MASTER_DATA_DIRECTORY/postgresql.conf`

Console Configuration

Changes to these files require a restart of Performance Monitor Console (`gpperfmon --restart`).

- `$GPPERFMONHOME/instances/`*`instance_name`*`/conf/gpperfmonui.conf`
- `$GPPERFMONHOME/instances/`*`instance_name`*`/conf/lighttpd.conf`

See Appendix B, "Configuration File Reference" for a description of the configuration parameters in these files.

Performance Monitor Agent Administration

This section describes basic agent administration tasks, including adding monitor hosts and viewing agent log files.

Adding and Removing Monitored Hosts

Segment agents on new hosts are detected automatically by the master agent. Whenever `gp_enable_gpperfmon` is enabled on the master, the master monitor agent automatically detects, starts, and begins harvesting data from new segment agents.

To verify the addition of a new monitored host, you can check for the new hostname in the Greenplum Performance Monitor Console **System Metrics** view described in the Greenplum Performance Monitor Console online help. Alternately, you can query the *system_now* table for the row containing current metrics for the host. For example:

```
=# SELECT * FROM system_now WHERE hostname='new_hostname';
```

Viewing and Maintaining Master Agent Log Files

Log messages for the master agent are written to `$MASTER_DATA_DIRECTORY/gpperfmon/logs/gpmmon.log` by default. To change the log file location, edit the `log_location` parameter in `gpperfmon.conf`.

On the segment hosts, agent log messages are written to a `gpsmon.log` file in the segment instance's data directory. For a host with multiple segments, the agent log file is located in the data directory of the first segment, as listed in the *gp_configuration* table by *dbid*. If the segment agent is unable to log into this directory, it will log messages to the home directory of the user running the monitor (typically `gpadmin`).

Configuring Log File Rollover

At higher logging levels, the size of the log files may grow dramatically. To prevent the log files from growing to excessive size, you can add an optional log rollover parameter to `gpperfmon.conf`. The value of this parameter is measured in bytes. For example:

```
max_log_size = 10485760
```

With this setting, the log files will grow to 10MB before the system rolls over the log file. The timestamp is added to the log file name when it is rolled over. Administrators must periodically clean out old log files that are no longer needed.

Performance Monitor Database Administration

Data collected by the Performance Monitor agents is stored in a dedicated database called `gpperfmon` within Greenplum Database. This database requires the typical database maintenance tasks such as clean up of old historical data and periodic `VACUUM ANALYZE`.

See "Performance Monitor Database Reference" on page 31 for a reference of the tables and views in the `gpperfmon` database.

Connecting to the Performance Monitor Database

Database administrators can connect directly to the `gpperfmon` database using any Greenplum Database-compatible client program (such as `psql`). For example:

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

Backing Up and Restoring the Monitor Database

The *history* tables of the `gpperfmon` database can be backed up and restored using the Greenplum Database parallel backup and restore utilities (`gp_dump`, `gp_restore`, `gpcrondump`, `gpdbrestore`). See the *Greenplum Database Administrator Guide* for more information.

Because the monitor database has a low number of tables, you may prefer to devise a backup plan using the table-level backup features of `gp_dump`. For example, you can create scripts to run `gp_dump` to back up the monthly partitions of the historical data tables on a monthly schedule. Alternately, you can back up your monitor database at the database level.

Maintaining the Historical Data Tables

All of the **_history* tables stored in the `gpperfmon` database are partitioned into monthly partitions. A January 2010 partition is created at installation time as a template partition (it can be deleted once some current partitions are created). The performance monitor agents automatically create new partitions in two month increments as needed. Administrators must periodically drop partitions for the months that are no longer needed in order to maintain the size of the `gpperfmon` database.

See the *Greenplum Database Administrator Guide* for more information on dropping partitions of a partitioned table.

Web Server Administration

The Lighttpd web server and web service middleware are installed in the `www` directory of your Greenplum Performance Monitor installation. For detailed information on Lighttpd administration, see http://www.lighttpd.net/.

Configuring the Web Server

The lighttpd web server configuration file is stored in `$GPPERFMONHOME/instances/instance_name/conf/lighttpd.conf`. Some of the parameters in this configuration file are set by the `gpperfmon` setup utility, including the web server port and SSL options. See "Web Server Parameters (lighttpd.conf)" on page 29 for a description of the parameters in this file.

Viewing and Maintaining Web Server Log Files

Web server access and error logs are written to `$GPPERFMONHOME/instances/instance_name/logs`. These two logs are:

- `lighttpd-access.log`
- `lighttpd-error.log`

If you experience errors viewing the Greenplum Performance Monitor Console, refer to these logs for more information.

To prevent web server logs from growing to excessive size, you can set up log file rotation using logrotate or cronolog, both of which are widely used with lighttpd.

A. Utility Reference

Greenplum provides the following utilities to install and maintain Greenplum Performance Monitor:

- gpperfmon_install - Enables the data collection agents.
- gpperfmon - Sets up and manages the web application.

gpperfmon_install

Installs the gpperfmon database and optionally enables the data collection agents.

Synopsis

```
gpperfmon_install
      [--enable --password gpmon_password --port gpdb_port]
      [--pgpass path_to_file]
      [--verbose]

gpperfmon_install --help | -h | -?
```

Description

The gpperfmon_install utility automates the steps required to enable the performance monitor data collection agents. You must be the Greenplum Database administrative user (gpadmin) in order to run this utility. If using the --enable option, Greenplum Database must be restarted after the utility completes.

When run without any options, the utility will just create the gpperfmon database (the database used to store performance monitor data). When run with the --enable option, the utility will also run the following additional tasks necessary to enable the performance monitor data collection agents:

1. Creates the gpmon superuser role in Greenplum Database. The performance monitor data collection agents require this role to connect to the database and write their data. The gpmon superuser role uses MD5-encrypted password authentication by default. Use the --password option to set the gpmon superuser's password. Use the --port option to supply the port of the Greenplum Database master instance.

2. Updates the $MASTER_DATA_DIRECTORY/pg_hba.conf file. The utility will add the following line to the host-based authentication file (pg_hba.conf). This allows the gpmon user to locally connect to any database using MD5-encrypted password authentication:
   ```
   local     all     gpmon     md5
   ```

3. Updates the password file (`.pgpass`). In order to allow the data collection agents to connect as the `gpmon` role without a password prompt, you must have a password file that has an entry for the `gpmon` user. The utility add the following entry to your password file (if the file does not exist, the utility will create it):
   `*:5432:gpperfmon:gpmon:`*`gpmon_password`*
   If your password file is not located in the default location (`~/.pgpass`), use the `--pgpass` option to specify the file location.

4. Sets the server configuration parameters for performance monitor. The following parameters must be enabled in order for the data collection agents to begin collecting data. The utility will set the following parameters in the Greenplum Database `postgresql.conf` configuration files:
   `gp_enable_gpperfmon=on` (in all `postgresql.conf` files)
   `gpperfmon_port=8888` (in all `postgresql.conf` files)
   `gp_external_enable_exec=on` (in the master `postgresql.conf` file)

Options

`--enable`

In addition to creating the `gpperfmon` database, performs the additional steps required to enable the performance monitor data collection agents. When `--enable` is specified the utility will also create and configure the `gpmon` superuser account and set the performance monitor server configuration parameters in the `postgresql.conf` files.

`--password` ***`gpmon_password`***

Required if `--enable` is specified. Sets the password of the `gpmon` superuser.

`--port` ***`gpdb_port`***

Required if `--enable` is specified. Specifies the connection port of the Greenplum Database master.

`--pgpass` ***`path_to_file`***

Optional if `--enable` is specified. If the password file is not in the default location of `~/.pgpass`, specifies the location of the password file.

`--verbose`

Sets the logging level to verbose.

`--help | -h | -?`

Displays the online help.

Examples

Create the `gpperfmon` database only:

```
$ su - gpadmin
$ gpperfmon_install
```

Create the `gpperfmon` database, create the `gpmon` superuser, and enable the performance monitor agents:

```
$ su - gpadmin
$ gpperfmon_install --enable --password p@$$word --port 5432
$ gpstop -r
```

See Also

`gpstop`

gpperfmon

Configures and manages instances of the Performance Monitor Console.

Synopsis

```
gpperfmon --setup [instance_name]
        | --start [instance_name]
        | --stop [instance_name]
        | --restart [instance_name]
        | --status [instance_name]
        | --upgrade
```

Description

The `gpperfmon` utility sets up and configures Performance Monitor Console instances, starts and stops instances, and provides status information. Also, `gpperfmon` facilitates upgrading pre-4.0.2 versions of the Performance Monitor Console.

For all actions, including starting and stopping, you can specify a console instance name. If you do not specify a name, the action applies to all existing console instances.

Options

`--setup`

Configures console components on the installation host. With this option, `gpperfmon` prompts for values to configure the components and writes the values to `gpperfmonui.conf` and `lighttpd.conf`. For more information on these configuration parameters, see Appendix B, “Configuration File Reference”.

`--start`

Starts the specified instance (or all instances by default) and its associated web service.

--stop

Stops the specified instance (or all instances by default) and its associated web service.

--restart

Restarts the specified instance (or all instances by default) and its associated web service.

--status

Displays the status, either `Running` or `Stopped`, of the web service.

--upgrade

Upgrades your Performance Monitor Console instances from 3.3.x to 4.0.2 by copying your old console instances from `$MASTER_DATA_DIRECTORY/gpperfmon` to the new `$GPPERFMONHOME/instances` location.

--version

Displays the version of the `gpperfmon` utility and the `lighttpd` web service.

Examples

Start the utility in order to install and configure a new instance:

```
$ gpperfmon --setup
```

Check the status of the monitor and web service:

```
$ gpperfmon --status
```

B. Configuration File Reference

Configuration parameters for Greenplum Performance Monitor are stored in these files:

- `$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf` — stores configuration parameters for the Greenplum Performance Monitor agents.
- `$GPPERFMONHOME/instances/`*`instance_name`*`/conf/gpperfmonui.conf` and `lighttpd.conf` — stores configuration parameters for the Performance Monitor web application and web server.
- `$MASTER_DATA_DIRECTORY/postgresql.conf` — stores configuration parameters to enable the Greenplum Performance Monitor feature for Greenplum Database.

Any system user with write permissions to these directories can edit these configuration files.

Performance Monitor Agent Parameters

The `$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf` stores configuration parameters for the Performance Monitor agents. For configuration changes to these options to take effect, you must save `gpperfmon.conf` and then restart Greenplum Database (`gpstop -r`).

To enable the Performance Monitor agents within Greenplum Database, you must also set the "Greenplum Database Server Configuration Parameters" on page 29.

quantum

Specifies the time in seconds between updates from performance monitor agents on all segments. Valid values are 10, 15, 20, 30, and 60. Defaults to 15 seconds.

If you prefer a less granular view of performance, or want to collect and analyze minimal amounts of data for system metrics, choose a higher quantum. To collect data more frequently, choose a lower value.

min_query_time

Specifies the minimum query run time in seconds for statistics collection. The monitor logs all queries that run longer than this value in the `queries_history` table. For queries with shorter run times, no historical data is collected. Defaults to 20 seconds.

If you know that you want to collect data for all queries, you can set this parameter to a low value. Setting the minimum query run time to zero, however, collects data even for the numerous queries run by the monitor itself, creating a large amount of data that may not be useful.

min_detailed_query_time

Specifies the minimum iterator run time in seconds for statistics collection. The monitor logs all iterators that run longer than this value in the `iterators_history` table. For iterators with shorter run times, no data is collected. Minimum value is 10 seconds.

This parameter's value must always be equal to, or greater than, the value of `min_query_time`. Setting `min_detailed_query_time` higher than `min_query_time` allows you to log detailed query plan iterator data only for especially complex, long-running queries, while still logging basic query data for shorter queries.

Given the complexity and size of iterator data, you may want to adjust this parameter according to the size of data collected. If the *iterators_** tables are growing to excessive size without providing useful information, you can raise the value of this parameter to log iterator detail for fewer queries.

log_location

Specifies a directory location for monitor log files. Default is `$MASTER_DATA_DIRECTORY/gpperfmon/logs`.

max_log_size

This parameter is not included in `gpperfmon.conf`, but it may be added to this file for use with Greenplum Performance Monitor.

To prevent the log files from growing to excessive size, you can add the `max_log_size` parameter to `gpperfmon.conf`. The value of this parameter is measured in bytes. For example:

```
max_log_size = 10485760
```

With this setting, the log files will grow to 10MB before the system rolls over to a new log file.

Performance Monitor Console Parameters

Each instance of the Performance Monitor Console has two configuration files located in `$GPPERFMONHOME/instances/`*`instance_name`*`/conf`. The web application file is `gpperfmonui.conf` and the web server file is `lighttpd.conf`.

After editing these files, reload the configuration by restarting the Performance Monitor Console instance (`gpperfmon --restart "`*`instance_name`*`"`).

Web Application Parameters (gpperfmonui.conf)

allow_trust_logon

Determines whether to allow access to the Greenplum Performance Monitor Console for any user listed in `pg_hba.conf` using the `trust` authentication method. When set to `yes`, trusted database users are allowed to login to the monitor console without a password. By default, this parameter is commented out.

server_name

Specifies the instance name displayed on the login page of the Greenplum Performance Monitor Console. This value can be any name you want to display to users, expressed as a text string. Defaults to the instance name you specified when setting up the Performance Monitor Console.

master_port

Specifies the port number of the Greenplum Database master that this instance is monitoring.

Web Server Parameters (lighttpd.conf)

This file has several configuration parameters, however these are the parameters most likely to change for a Performance Monitor Console installation. For more information on the other parameters in this configuration file, see the lighttpd documentation.

server.port

Sets the web server port number. The default HTTP port is 28080.

ssl.engine

Determines whether SSL encryption is used for client connections. Valid values are `enable` and `disable`. If you enable SSL at installation time, this is set to `enable`.

ssl.pemfile

Specifies the path to the PEM file for SSL support. If you enable SSL at installation time, this parameter points to a self-signed certificate. If you use a trusted signed certificate, you must specify it with this parameter.

Greenplum Database Server Configuration Parameters

The following parameters must be uncommented and set in the server configuration file (`postgresql.conf`) in order to enable the Performance Monitor data collection agents. gp_enable_gpperfmon and gpperfmon_port must be set in both the master and segment `postgresql.conf` files. gp_enable_gpperfmon and gp_enable_gpperfmon only need to be set in the master `postgresql.conf` file.

After changing these settings, Greenplum Database must be restarted for the changes to take effect.

gp_enable_gpperfmon

Turns on the Performance Monitor data collection agent for a segment. Must be set in all `postgresql.conf` files (master and all segments).

gpperfmon_port

The default port for the monitor agents is 8888, but you can set this parameter to a different port if you like (master and all segments).

gp_gpperfmon_send_interval

Sets the frequency in seconds that the Greenplum Database server processes send query execution updates to the Performance Monitor agent processes.

gp_external_enable_exec

This parameter is enabled by default and must remain enabled. It allows the use of external tables that execute OS commands or scripts on the segment hosts. The Performance Monitor agents use this type of external tables to collect current system metrics from the segments.

C. Performance Monitor Database Reference

This is the schema reference for the tables and views in the Greenplum Performance Monitor database (`gpperfmon`).

The database consists of three sets of tables; *now* tables store data on currently active queries, *history* tables store data on completed queries, and *tail* tables are for data in transition. The *now* and *tail* data are stored as text files on the master file system, and accessed by Greenplum Database via external tables. The *history* tables are regular database tables stored within the `gpperfmon` database.

There are five categories of tables:

- The database_* tables store query workload information for a Greenplum Database instance.
- The iterators_* tables store information about query plan iterators and their metrics. A query iterator refers to a node or operation in a query plan.
- The queries_* tables store high-level query status information.
- The segment_* tables store memory allocation statistics for the Greenplum Database segment instances.
- The system_* tables store system utilization metrics.

The Performance Monitor 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 iterators_*_rollup set of views summarize the query iterator metrics across all segments in the system.
- The memory_info view shows per-host memory information from the *system_history* and *segment_history* tables.

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 `$MASTER_DATA_DIRECTORY/gpperfmon/data`. Current query workload data is stored in *database_now* during the period between data collection from the monitor agents and automatic commitment to the *database_history* table.
- *database_tail* is an external table whose data files are stored in `$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. Administrators must drop old partitions for the months that are no longer needed.

Table D.1 gpperfmon database - database_* tables

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 queue at data collection time.

iterators_*

The *iterators_** tables store information about query plan iterators and their metrics. A query iterator refers to a node or operation in a query plan. For example, a sequential scan operation on a table may be one type of iterator in a particular query plan.

The *tmid*, *ssid* and *ccnt* columns are the composite key that uniquely identifies a particular query. These columns can be used to join with the queries_* data tables.

There are three iterator tables, all having the same columns:

- *iterators_now* is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. Current query plan iterator data is stored in *iterators_now* during the period between data collection from the monitor agents and automatic commitment to the *iterators_history* table.
- *iterators_tail* is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. This is a transitional table for query plan iterator data that has been cleared from *iterators_now* but has not yet been committed to *iterators_history*. It typically only contains a few minutes worth of data.
- *iterators_history* is a regular table that stores historical query plan iterator data. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed. Administrators must drop old partitions for the months that are no longer needed.

See also the *iterator_rollup* views for summary metrics of the query plan iterator data.

Table E.1 gpperfmon database - iterators_* tables

column	type	description
ctime	timestamp	Time this row was created.
tmid	int	A time identifier for a particular query. All iterator records associated with the query will have the same *tmid*.
ssid	int	The session id as shown by the `gp_session_id` parameter. All iterator records associated with the query will have the same *ssid*.
ccnt	in	The command number within this session as shown by `gp_command_count` parameter. All iterator records associated with the query will have the same *ccnt*.
segid	int	The segment ID (*dbid* from *gp_segment_configuration*).
pid	int	The `postgres` process ID for this iterator.
nid	int	The query plan node ID from the Greenplum slice plan.
pnid	int	The parent query plan node ID from the Greenplum slice plan.
hostname	varchar(64)	Segment hostname.

Table E.1 gpperfmon database - iterators_* tables

column	type	description
ntype	varchar(64)	The iterator operation type. Possible values are: `Adaptive Join, Hash Aggregate, Aggregate, Group Aggregate, Append, Append-Only Scan, BitmapAnd, Bitmap Heap Scan, Bitmap Index Scan, BitmapOr, External Scan, Function Scan, Group, Hash, Hash Join, Hash Left Join, Hash Full Join, Hash Right Join, Hash EXISTS Join, Index Scan, Limit, Materialize, Merge Join, Merge Left Join, Merge Full Join, Merge Right Join, Merge EXISTS Join, Redistribute Motion, Broadcast Motion, Gather Motion, Nested Loop, Nested Loop Left Join, Nested Loop Left Anti Semi Join, Nested Loop Full Join, Nested Loop Right Join, Result, Seq Scan, SetOp, SetOp Intersect, SetOp Intersect All, SetOp Except, SetOp Except All, Shared Scan, Sort, Subquery Scan, Tid Scan, Unique, Values Scan, Window.`
nstatus	varchar(64)	The status of this iterator. Possible values are: `Initialize, Executing and Finished.`
tstart	timestamp	Start time for the iterator.
tduration	int	Duration of the execution.
pmemsize	bigint	Work memory allocated by the Greenplum planner to this iterator's query process.
pmemmax	bigint	Maximum work memory allocated by the Greenplum planner to this iterator's query process.
memsize	bigint	OS memory allocated to this iterator's process.
memresid	bigint	Resident memory allocated to this iterator's process (as opposed to shared memory).
memshare	bigint	Shared memory allocated to this iterator's process.
cpu_elapsed	bigint	Total CPU usage of the process executing the iterator.
cpu_currpct	float	The percentage of CPU currently being utilized by this iterator process. This value is always zero for historical (completed) iterators.
phase	varchar(64)	This column is not implemented at time of writing.
rowsout	bigint	The actual number of rows output by the iterator.
rowsout_est	bigint	The query planner's estimate of rows output by the iterator.
m0_name	varchar(64)	Each operation in a query plan (ntype) has metrics associated with it. For all operations, this metric name is `Rows In`.
m0_unit	varchar(64)	The unit of measure for this metric. For all operations (ntype), this unit of measure is `Rows`.
m0_val	bigint	The value of this metric.
m0_est	bigint	The estimated value of this metric.

Table E.1 gpperfmon database - iterators_* tables

column	type	description

m1_name	varchar(64)	Each operation in a query plan (ntype) has metrics associated with it. This metric name is as follows for the corresponding plan node operations:
		`Hash Aggregate - Agg Total Spill Tuples`
		`Aggregate - Agg Total Spill Tuples`
		`Group Aggregate - Agg Total Spill Tuples`
		`Append - Append Current Input Source`
		`Append-Only Scan - Rescans`
		`Bitmap Heap Scan - Bitmap Heap Scan Pages`
		`Hash - Hash Spill Batches`
		`Hash Join - Hash Join Spill Batches`
		`Hash Left Join - Hash Join Spill Batches`
		`Hash Left Anti Semi Join - Hash Join Spill Batches`
		`Hash Full Join - Hash Join Spill Batches`
		`Hash Right Join - Hash Join Spill Batches`
		`Hash EXISTS Join - Hash Join Spill Batches`
		`Index Scan - Index Scan Restore Pos`
		`Merge Join - Merge Join Inner Tuples`
		`Merge Left Join - Merge Join Inner Tuples`
		`Merge Left Anti Semi Join - Merge Join Inner Tuples`
		`Merge Full Join - Merge Join Inner Tuples`
		`Merge Right Join - Merge Join Inner Tuples`
		`Merge EXISTS Join - Merge Join Inner Tuples`
		`Nested Loop - Nested Loop Inner Tuples`
		`Nested Loop Left Join - Nested Loop Inner Tuples`
		`Nested Loop Left Anti Semi Join - Nested Loop Inner Tuples`
		`Nested Loop Full Join - Nested Loop Inner Tuples`
		`Nested Loop Right Join - Nested Loop Inner Tuples`
		`Seq Scan - SeqScan Page Stats`
		`Shared Scan - Shared Scan Restore Pos`
		`Sort - Sort Memory Usage`
		`Subquery Scan - Subquery Scan Rescan`

Table E.1 gpperfmon database - iterators_* tables

column	type	description

m1_unit	varchar(64)	The unit of measure for this metric. This unit is as follows for the corresponding plan node operations:
		`Hash Aggregate - Tuples`
		`Aggregate - Tuples`
		`Group Aggregate - Tuples`
		`Append - Inputs`
		`Append-Only Scan - Rescans`
		`Bitmap Heap Scan - Pages`
		`Hash - Batches`
		`Hash Join - Batches`
		`Hash Left Join - Batches`
		`Hash Full Join - Batches`
		`Hash Right Join - Batches`
		`Hash EXISTS Join - Batches`
		`Index Scan - Restores`
		`Merge Join - Tuples`
		`Merge Left Join - Tuples`
		`Merge Full Join - Tuples`
		`Merge Right Join - Tuples`
		`Merge EXISTS Join - Tuples`
		`Nested Loop - Tuples`
		`Nested Loop Left Join - Tuples`
		`Nested Loop Full Join - Tuples`
		`Nested Loop Right Join - Tuples`
		`Seq Scan - Pages`
		`Shared Scan - Restores`
		`Sort - Bytes`
		`Subquery Scan - Rescans`

m1_val	bigint	The value of this metric.
m1_est	bigint	The estimated value of this metric.

Table E.1 gpperfmon database - iterators_* tables

column	type	description

m2_name	varchar(64)	Each operation in a query plan (ntype) has metrics associated with it. This metric name is as follows for the corresponding plan node operations:
		`Hash Aggregate - Agg Total Spill Bytes`
		`Aggregate - Agg Total Spill Bytes`
		`Group Aggregate -Agg Total Spill Bytes`
		`Hash - Hash Spill Tuples`
		`Hash Join - Hash Join Spill Tuples`
		`Hash Left Join - Hash Join Spill Tuples`
		`Hash Left Anti Semi Join - Hash Join Spill Tuples`
		`Hash Full Join - Hash Join Spill Tuples`
		`Hash Right Join - Hash Join Spill Tuples`
		`Hash EXISTS Join - Hash Join Spill Tuples`
		`Index Scan - Index Scan Rescan`
		`Merge Join - Merge Join Outer Side Tuples`
		`Merge Left Join - Merge Join Outer Tuples`
		`Merge Left Anti Semi Join - Merge Join Outer Tuples`
		`Merge Full Join - Merge Join Outer Tuples`
		`Merge Right Join - Merge Join Outer Tuples`
		`Merge EXISTS Join - Merge Join Outer Tuples`
		`Nested Loop - Nested Loop Outer Tuples`
		`Nested Loop Left Join - Nested Loop Outer Tuples`
		`Nested Loop Left Anti Semi Join - Nested Loop Outer Tuples`
		`Nested Loop Full Join - Nested Loop Outer Tuples`
		`Nested Loop Right Join - Nested Loop Outer Tuples`
		`Seq Scan - SeqScan Restore Pos`
		`Shared Scan - Shared Scan Rescan`
		`Sort - Sort Spill Tuples`

Table E.1 gpperfmon database - iterators_* tables

column	type	description

m2_unit	varchar(64)	The unit of measure for this metric. This unit is as follows for the corresponding plan node operations:
		`Hash Aggregate - Bytes`
		`Aggregate - Bytes`
		`Group Aggregate - Bytes`
		`Hash - Tuples`
		`Hash Join - Tuples`
		`Hash Left Join - Tuples`
		`Hash Full Join - Tuples`
		`Hash Right Join - Tuples`
		`Hash EXISTS Join - Tuples`
		`Index Scan - Rescans`
		`Merge Join - Tuples`
		`Merge Left Join - Tuples`
		`Merge Full Join - Tuples`
		`Merge Right Join - Tuples`
		`Merge EXISTS Join - Tuples`
		`Nested Loop - Tuples`
		`Nested Loop Left Join - Tuples`
		`Nested Loop Full Join - Tuples`
		`Nested Loop Right Join - Tuples`
		`Seq Scan - Restores`
		`Shared Scan - Rescans`
		`Sort - Tuples`

m2_val	bigint	The value of this metric.
m2_est	bigint	The estimated value of this metric.

m3_name	varchar(64)	Each operation in a query plan (ntype) has metrics associated with it. This metric name is as follows for the corresponding plan node operations:
		`Hash Aggregate - Agg Total Spill Batches`
		`Aggregate - Agg Total Spill Batches`
		`Group Aggregate - Agg Total Spill Batches`
		`Hash - Hash Spill Bytes`
		`Hash Join - Hash Join Spill Bytes`
		`Hash Left Join - Hash Join Spill Bytes`
		`Hash Left Anti Semi Join - Hash Join Spill Bytes`
		`Hash Full Join - Hash Join Spill Bytes`
		`Hash Right Join - Hash Join Spill Bytes`
		`Hash EXISTS Join - Hash Join Spill Bytes`
		`Seq Scan - SeqScan Rescan`
		`Sort - Sort Spill Bytes`

Table E.1 gpperfmon database - iterators_* tables

column	type	description

m3_unit	varchar(64)	The unit of measure for this metric. This unit is as follows for the corresponding plan node operations:
		`Hash Aggregate - Batches`
		`Aggregate - Batches`
		`Group Aggregate - Batches`
		`Hash - Bytes`
		`Hash Join - Bytes`
		`Hash Left Join - Bytes`
		`Hash Full Join - Bytes`
		`Hash Right Join - Bytes`
		`Hash EXISTS Join - Bytes`
		`Seq Scan - Rescans`
		`Sort - Bytes`

m3_val	bigint	The value of this metric.
m3_est	bigint	The estimated value of this metric.

m4_name	varchar(64)	Each operation in a query plan (ntype) has metrics associated with it. This metric name is as follows for the corresponding plan node operations:
		`Hash Aggregate - Agg Total Spill Pass`
		`Aggregate - Agg Total Spill Pass`
		`Group Aggregate -Agg Total Spill Pass`
		`Sort - Sort Spill Pass`

m4_unit	varchar(64)	The unit of measure for this metric. This unit is as follows for the corresponding plan node operations:
		`Hash Aggregate - Passes`
		`Aggregate - Passes`
		`Group Aggregate - Passes`
		`Sort - Passes`

m4_val	bigint	The value of this metric.
m4_est	bigint	The estimated value of this metric.

m5_name	varchar(64)	Each operation in a query plan (ntype) has metrics associated with it. This metric name is as follows for the corresponding plan node operations:
		`Hash Aggregate - Agg Current Spill Pass Read Tuples`
		`Aggregate - Agg Current Spill Pass Read Tuples`
		`Group Aggregate - Agg Current Spill Pass Read Tuples`
		`Sort - Sort Current Spill Pass Tuples`

Table E.1 gpperfmon database - iterators_* tables

column	type	description

m5_unit	varchar(64)	The unit of measure for this metric. This unit is as follows for the corresponding plan node operations:
		`Hash Aggregate - Tuples`
		`Aggregate - Tuples`
		`Group Aggregate - Tuples`
		`Sort - Tuples`

m5_val	bigint	The value of this metric.
m5_est	bigint	The estimated value of this metric.

m6_name	varchar(64)	Each operation in a query plan (ntype) has metrics associated with it. This metric name is as follows for the corresponding plan node operations:
		`Hash Aggregate - Agg Current Spill Pass Read Bytes`
		`Aggregate - Agg Current Spill Pass Read Bytes`
		`Group Aggregate - Agg Current Spill Pass Read Bytes`
		`Sort - Sort Current Spill Pass Bytes`

m6_unit	varchar(64)	The unit of measure for this metric. This unit is as follows for the corresponding plan node operations:
		`Hash Aggregate - Bytes`
		`Aggregate - Bytes`
		`Group Aggregate - Bytes`
		`Sort - Bytes`

m6_val	bigint	The value of this metric.
m6_est	bigint	The estimated value of this metric.

m7_name	varchar(64)	Each operation in a query plan (ntype) has metrics associated with it. This metric name is as follows for the corresponding plan node operation:
		`Hash Aggregate - Agg Current Spill Pass Tuples`
		`Aggregate - Agg Current Spill Pass Tuples`
		`Group Aggregate - Agg Current Spill Pass Tuples`

m7_unit	varchar(64)	The unit of measure for this metric. This unit is as follows for the corresponding plan node operation:
		`Hash Aggregate - Tuples`
		`Aggregate - Tuples`
		`Group Aggregate - Tuples`

m7_val	bigint	The value of this metric.
m7_est	bigint	The estimated value of this metric.

Table E.1 gpperfmon database - iterators_* tables

column	type	description

m8_name	varchar(64)	Each operation in a query plan (ntype) has metrics associated with it. This metric name is as follows for the corresponding plan node operation:
		`Hash Aggregate - Agg Current Spill Pass Bytes`
		`Aggregate - Agg Total Spill Pass Bytes`
		`Group Aggregate -Agg Total Spill Pass Bytes`

m8_unit	varchar(64)	The unit of measure for this metric. This unit is as follows for the corresponding plan node operation:
		`Hash Aggregate - Bytes`
		`Aggregate - Bytes`
		`Group Aggregate - Bytes`

m8_val	bigint	The actual value of this metric.
m8_est	bigint	The estimated value of this metric.

m9_name	varchar(64)	Each operation in a query plan (ntype) has metrics associated with it. This metric name is as follows for the corresponding plan node operation:
		`Hash Aggregate - Agg Current Spill Pass Batch`
		`Aggregate - Agg Total Spill Pass Batch`
		`Group Aggregate -Agg Total Spill Pass Batch`

m9_unit	varchar(64)	The unit of measure for this metric. This unit is as follows for the corresponding plan node operation:
		`Hash Aggregate - Batches`
		`Aggregate - Batches`
		`Group Aggregate - Batches`

m9_val	bigint	The actual value of this metric.
m9_est	bigint	The estimated value of this metric.
m10_name - m15_name	varchar(64)	The iterator name (ntype) associated with this metric. Metrics m10 through m15 are currently not used.
m10_unit - m15_unit	varchar(64)	The unit of measure for this metric. Metrics m10 through m15 are currently not used.
m10_value - m15_value	bigint	The actual value of this metric. Metrics m10 through m15 are currently not used.
m10_est - m15_est	bigint	The estimated value of this metric. Metrics m10 through m15 are currently not used.
t0_name	varchar(64)	This column is a label for t0_val. Its value is always `Name`.
t0_val	varchar(128)	The name of the relation being scanned by an iterator. This metric is collected only for iterators that perform scan operations such as a sequential scan or function scan.

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. These columns can be used to join with the iterators_* tables.

There are three queries tables, all having the same columns:

- *queries_now* is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. Current query status is stored in *queries_now* during the period between data collection from the monitor agents and automatic commitment to the *queries_history* table.
- *queries_tail* is an external table whose data files are stored in `$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. Administrators must drop old partitions for the months that are no longer needed.

Table F.1 gpperfmon database - queries_* tables

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	in	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)	Database role name that issued this query.
db	varchar(64)	Name of the database queried.
cost	integer	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	bigin	Rows out for the query.

Table F.1 gpperfmon database - queries_* tables

column	type	description
cpu_elapsed	bigin	CPU usage for this query across all segments, measured in milliseconds. This is the sum of the CPU usage values taken from all active primary segments in the database array. Note that the Greenplum Performance Monitor logs the value 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 values for `min_query_time` and `min_detailed_query`, and these values are 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.
skew_cpu	float	Coefficient of variation for `cpu_elapsed` of iterators across all segments for this query, multiplied by 100. For example, a value of .95 is rendered as 95.
skew_rows	float	Coefficient of variation for `rows_in` of iterators across all segments for this query, multiplied by 100. For example, a value of .95 is rendered 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.

segment_*

The *segment_** tables contain memory allocation statistics for the Greenplum Database segment instances. This tracks the amount of memory consumed by all `postgres` processes of a particular segment instance, and the remaining amount of memory available to a segment as per the setting of the `postgresql.conf` configuration parameter: `gp_vmem_protect_limit`. Query processes that cause a segment to exceed this limit will be cancelled in order to prevent system-level out-of-memory errors. See the *Greenplum Database Administrator Guide* for more information about this parameter.

There are three segment tables, all having the same columns:

- *segment_now* is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. Current memory allocation data is stored in *segment_now* during the period between data collection from the monitor agents and automatic commitment to the *segment_history* table.
- *segment_tail* is an external table whose data files are stored in `$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. Administrators must drop old partitions for the months that are no longer needed.

A particular segment instance is identified by its *hostname* and *dbid* (the unique segment identifier as per the *gp_segment_configuration* system catalog table).

Table G.1 gpperfmon database - segment_* tables

column	type	description
ctime	timestamp(0) without time zone	The time the row was created.
dbid	integer	The segment ID (*dbid* from *gp_segment_configuration*).
hostname	character varying(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 `gp_vmem_protect_limit` parameter.

See also the views *memory_info* and *dynamic_memory_info* for aggregated memory allocation and utilization by host.

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 `$MASTER_DATA_DIRECTORY/gpperfmon/data`. Current system utilization data is stored in *system_now* during the period between data collection from the monitor agents and automatic commitment to the *system_history* table.
- *system_tail* is an external table whose data files are stored in `$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. Administrators must drop old partitions for the months that are no longer needed.

Table H.1 gpperfmon database - system_* tables

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.

Table H.1 gpperfmon database - system_* tables

column	type	description
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.
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.

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 will allow the query processes of a single segment instance to consume before it starts cancelling processes. This limit is set by the `gp_vmem_protect_limit` server configuration parameter, and is evaluated on a per-segment basis.

Table I.1 gpperfmon database - dynamic_memory_info view

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 as set by the `gp_vmem_protect_limit` parameter.

iterators_*_rollup

The *iterators_*_rollup* set of views aggregate the metrics stored in the *iterators_** tables. A query *iterator* refers to a *node* or *operation* in a query plan. For example, a sequential scan operation may be one type of iterator in a particular query plan. For each iterator in a query plan, the *iterators_** tables store the metrics collected from each segment instance. The *iterators_*_rollup* views summarize the query iterator metrics across all segments in the system.

The *tmid*, *ssid* and *ccnt* columns are the composite key that uniquely identifies a particular query.

There are three iterators rollup views, all having the same columns:

- *The iterators_now_rollup* view shows iterator data from the *interators_now* table aggregated across all segments in the system.
- The *iterators_tail_rollup* view shows iterator data from the *interators_tail* table aggregated across all segments in the system.
- The *iterators_history_rollup* shows iterator data from the *interators_history* table aggregated across all segments in the system.

See also the *iterators_** tables for more information about the query plan iterator types and the metrics collected for each iterator.

Table J.1 gpperfmon database - iterators_*_rollup views

column	type	description
sample_time	timestamp	The ctime from the associated iterators_* table.
tmid	int	A time identifier for a particular query. All iterator records associated with the query will have the same *tmid*.
ssid	int	The session id as shown by the `gp_session_id` parameter. All iterator records associated with the query will have the same *ssid*.
ccnt	int	The command number within this session as shown by `gp_command_count` parameter. All iterator records associated with the query will have the same *ccnt*.
nid	int	The ID of this query plan node from the slice plan.
pnid	int	The pnid (slice plan parent node ID) from the associated iterators_* table.
ntype	text	The ntype (node/iterator type) from the associated iterators_* table.
nstatus	text	The accumulated status of this iterator. Possible values are: `Initialize`, `Executing`, or `Finished`.
tstart	timestamp	The average start time for this iterator.
tduration	numeric	The average execution time for this iterator.
pmemsize	numeric	The average work memory allocated by the Greenplum planner to this iterator's query processes.
pmemmax	numeric	The average of the maximum planner work memory used by this iterator's query processes.

Table J.1 gpperfmon database - iterators_*_rollup views

column	type	description
memsize	numeric	The average OS memory allocated to this iterator's processes.
memresid	numeric	The average resident memory allocated to this iterator's processes (as opposed to shared memory).
memshare	numeric	The average shared memory allocated to this iterator's processes.
cpu_elapsed	numeric	Sum of the CPU usage of all segment processes executing this iterator.
cpu_currpct	double precision	The current average percentage of CPU utilization used by this iterator's processes. This value is always zero for historical (completed) iterators.
phase	text	This column is not implemented.
rows_out	numeric	The total number of actual rows output for this iterator on all segments.
rows_out_est	numeric	The total number of output rows for all segments as estimated by the query planner.
skew_cpu	numeric	Coefficient of variation for cpu_elapsed of iterators across all segments for this query, multiplied by 100. For example, a value of .95 is rendered as 95.
skew_rows	numeric	Coefficient of variation for rows_out of iterators across all segments for this query, multiplied by 100. For example, a value of .95 is rendered as 95.
m0	text	The name (m0_name), unit of measure (m0_unit), average actual value (m0_val), and average estimated value (m0_est) for this iterator metric across all segments. The m0 metric is always *rows* for all iterator types.
m1	text	The name (m1_name), unit of measure (m1_unit), average actual value (m1_val), and average estimated value (m1_est) for this iterator metric across all segments.
m2	text	The name (m2_name), unit of measure (m2_unit), average actual value (m2_val), and average estimated value (m2_est) for this iterator metric across all segments.
m3	text	The name (m3_name), unit of measure (m3_unit), average actual value (m3_val), and average estimated value (m3_est) for this iterator metric across all segments.
m4	text	The name (m4_name), unit of measure (m4_unit), average actual value (m4_val), and average estimated value (m4_est) for this iterator metric across all segments.
m5	text	The name (m5_name), unit of measure (m5_unit), average actual value (m5_val), and average estimated value (m5_est) for this iterator metric across all segments.
m6	text	The name (m6_name), unit of measure (m6_unit), average actual value (m6_val), and average estimated value (m6_est) for this iterator metric across all segments.

Table J.1 gpperfmon database - iterators_*_rollup views

column	type	description
m7	text	The name (m7_name), unit of measure (m7_unit), average actual value (m7_val), and average estimated value (m7_est) for this iterator metric across all segments.
m8	text	The name (m8_name), unit of measure (m8_unit), average actual value (m8_val), and average estimated value (m8_est) for this iterator metric across all segments.
m9	text	The name (m9_name), unit of measure (m9_unit), average actual value (m9_val), and average estimated value (m9_est) for this iterator metric across all segments.
m10 - m15	text	Metrics m10 through m15 are not currently used by any iterator types.
t0	text	The name of the relation (t0_val) being scanned by this iterator. This metric is collected only for iterators that perform scan operations such as a sequential scan or function scan.

memory_info

The *memory_info* view shows per-host memory information from the *system_history* and *segment_history* tables. This allows administrators to compare total memory available on a segment host, total memory used on a segment host, and dynamic memory used by query processes.

Table K.1 gpperfmon database - memory_info view

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.
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 as set by the `gp_vmem_protect_limit` parameter.

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

O

P

Q

R

S

T

U

V

W