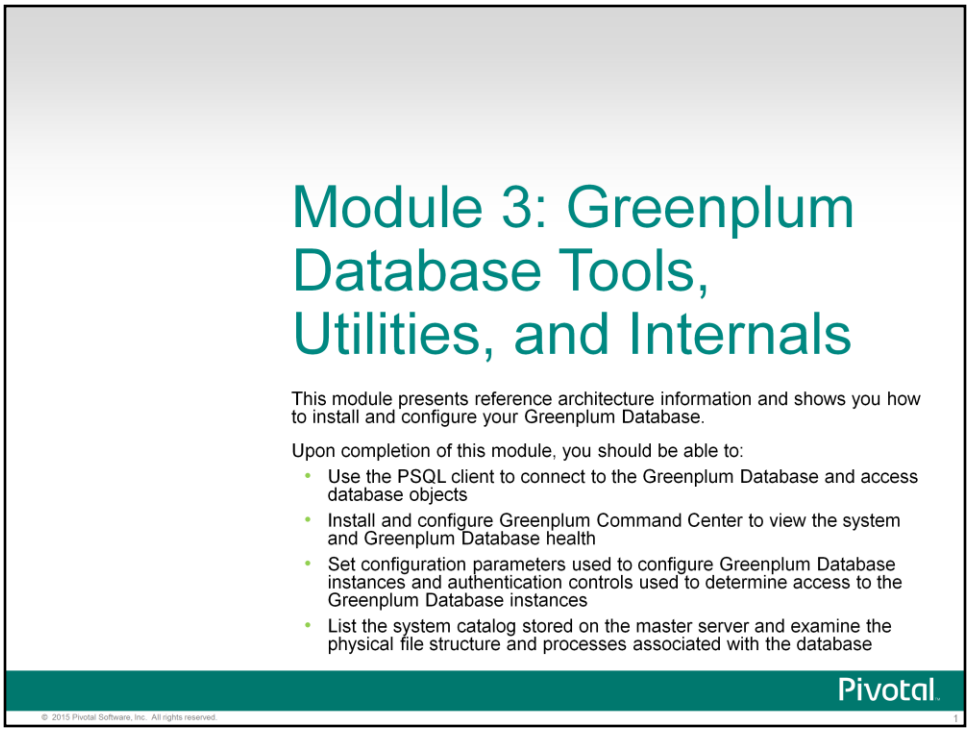
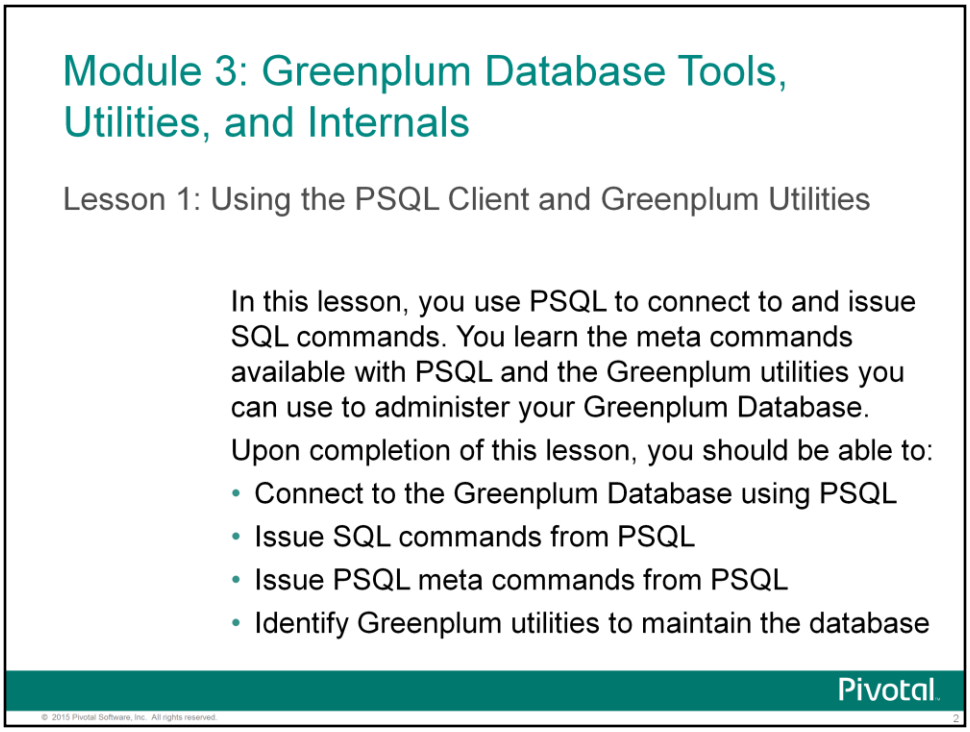


In this module, you access the Greenplum Database with the PSQL client and examine several database objects. You will also examine the Greenplum Command Center tool that is available for installation to help you monitor the Greenplum environment, examine queries that are currently running, and allow you to see how system resources are being consumed.

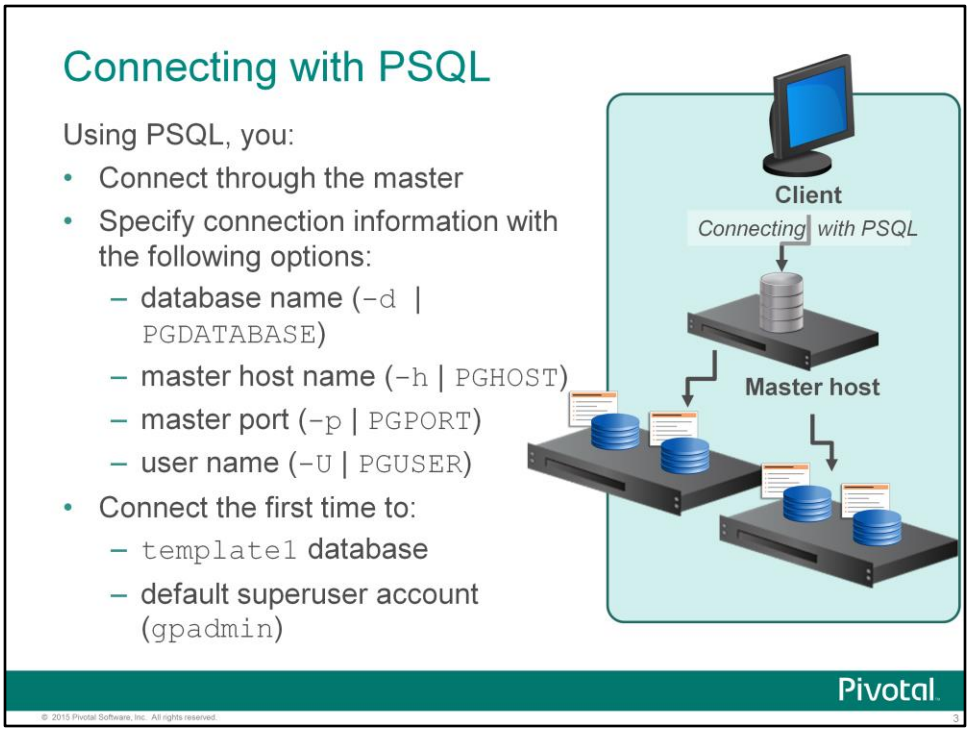
You will become familiar with the configuration parameters that you may modify from time to time, based on changes to the environment. It is therefore important for you to understand some basic internal concepts for the Greenplum environment.



Once the database has been installed, you should verify that you can connect to the Greenplum Database while familiarizing yourself with commands to access the database and database objects.

In this lesson, you:

- Use PSQL to connect to the database
- Execute SQL commands interactively and in batch mode
- Examine the PSQL meta commands that allow you to access database objects
- Identify commands to access database objects



PSQL:

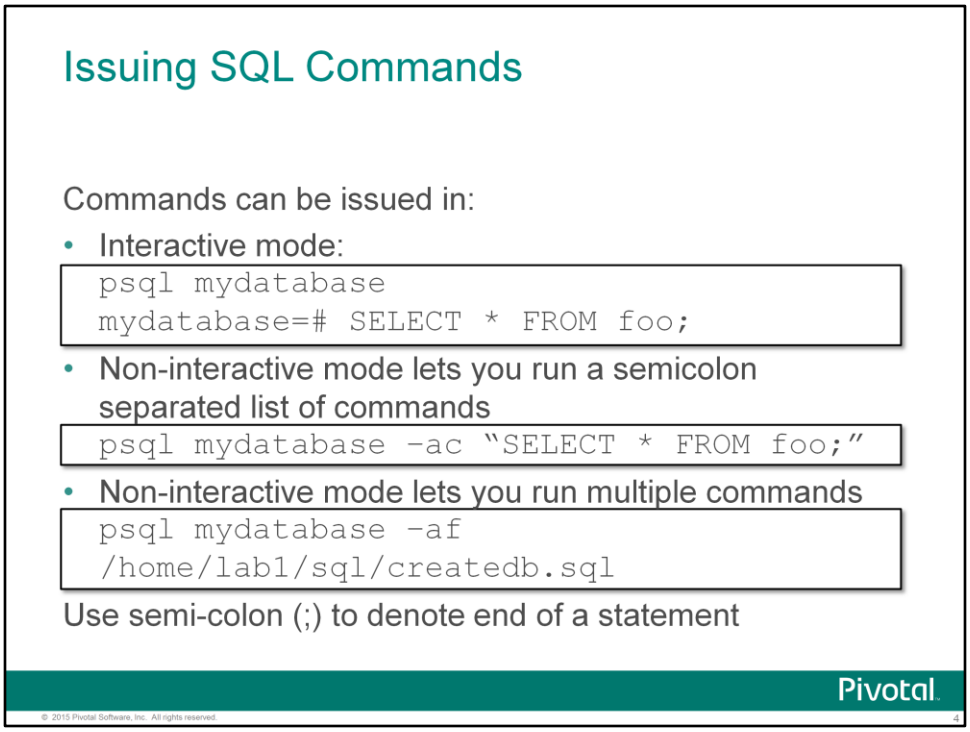
- Is a front-end terminal-based connection to PostgreSQL
- Allows you to interactively and non-interactively issue SQL queries
- Provides a number of meta commands to facilitate script writing

Using PSQL, you connect to the Greenplum Database by connecting to the master to issue any SQL commands. You do not connect to the segments directly.

Use `psql` options to connect to a specific database with a specific user or set environment variables with your default settings. Options to the `psql` command supersede environment variables. The options and alternative variables are:

- `-d` *`database_name`* allows you to specify the database you want to connect to. The alternative variable you set is `PGDATABASE`.
- `-h` *`hostname`* lets you specify the host to which you want to connect. The alternative variable is `PGHOST`.
- `-p` *`port_number`* lets you specify the port number for the database to connect to. The alternative variable is `PGPORT`.
- `-u` *`user_name`* lets you specify the user name to connect to in the database. The alternative variable is `PGUSER`.

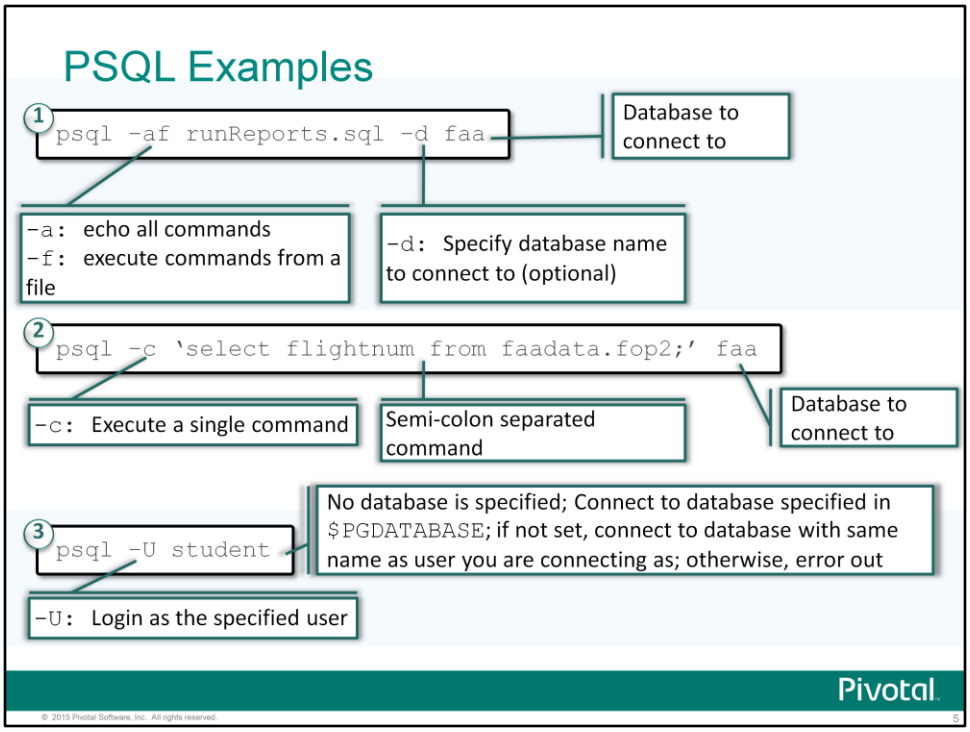
If you do not specify these options, by default, you will be automatically connected to `template1` database that is created by default when installing a PostgreSQL database. You will connect as the default superuser account, `gpadmin`.



There are three ways to execute SQL commands using `psql`:

- In Interactive Mode, you start a PSQL session and enter your commands at the SQL prompt
- In Non-Interactive Mode, you execute a single statement using the `-c` option
- In non-interactive mode, you specify the `-f` option to run SQL statements from a file

An SQL statement must end with a `;` (semicolon) to both denote the end of the SQL statement and to execute that statement.



Here are some examples of the PSQL command.

The first example is run in non-interactive mode. All of the commands are stored in the SQL file, runReports.sql. The `-a` option echoes all commands provided, while the `-f` option allows you to specify a file name that contains the commands and exit once complete. The `-d` option allows you to specify the database you are connecting to when executing the command. This is followed by the database name. You do not have to use the `-d` option when specifying a database name.

In the second example, you specify a quoted semi-colon separated list of commands to execute by using the `-c` option. This is followed by the database name, which in this example, is not preceded by the `-d` option.

In the last example, the user specified with the `-U` option, is connecting to the database, though a database name is not specified on the command line. If the database name is not specified, the command will use the value set in the `PGDATABASE` variable. If that is not set, psql will attempt to connect to a database with the same name as the user you are connecting to the database as. If that fails, then the command will error out.

Common PSQL Meta-Commands

Commonly used PSQL meta-commands include:

Meta-Command	Description
`\?`	Help on psql meta-commands
`\h`	Help on SQL command syntax
`\dt`	Show tables
`\dtS`	Show system tables
`\dg` `\du`	Show roles
`\l`	Show databases
`\c db_name`	Connect to the specified database
`\dn`	Show schemas
`\q`	Quit psql

Pivotal

PSQL meta-commands are commands that are processed by psql itself. Anything you enter in PSQL that begins with an unquoted backslash (\) is a PSQL meta-command. The commands are used for administration and scripting.

The table shows a list of common meta-commands to connect to a database, list database objects, and obtain help from PSQL.

Greenplum Client Utility Applications

Greenplum provides a wide range of client applications to help administer the database, including:

Utilities	Description
`createdb`	Create a new Greenplum Database
`createlang`	Define a new procedural language
`createuser`	Define a new database role with login privileges
`dropdb`	Remove or drop a database
`droplang`	Remove or drop a procedural language
`dropuser`	Remove or drop a role
`gppkg`	Install Greenplum Database extensions, such as PL/R and PL/Java
`pg_config`	Retrieves information about the installed version of Greenplum Database
`reindexdb`	Reindex a database
`vacuumdb`	Garbage collect and analyze a database

Pivotal

In addition to the `psql` command, Greenplum provides a number of client applications in the `$GPHOME/bin` directory of your Greenplum master host installation. These commands, executed on the command line, allow you to manage different aspects of the database. The table shows the list of commonly used client applications.

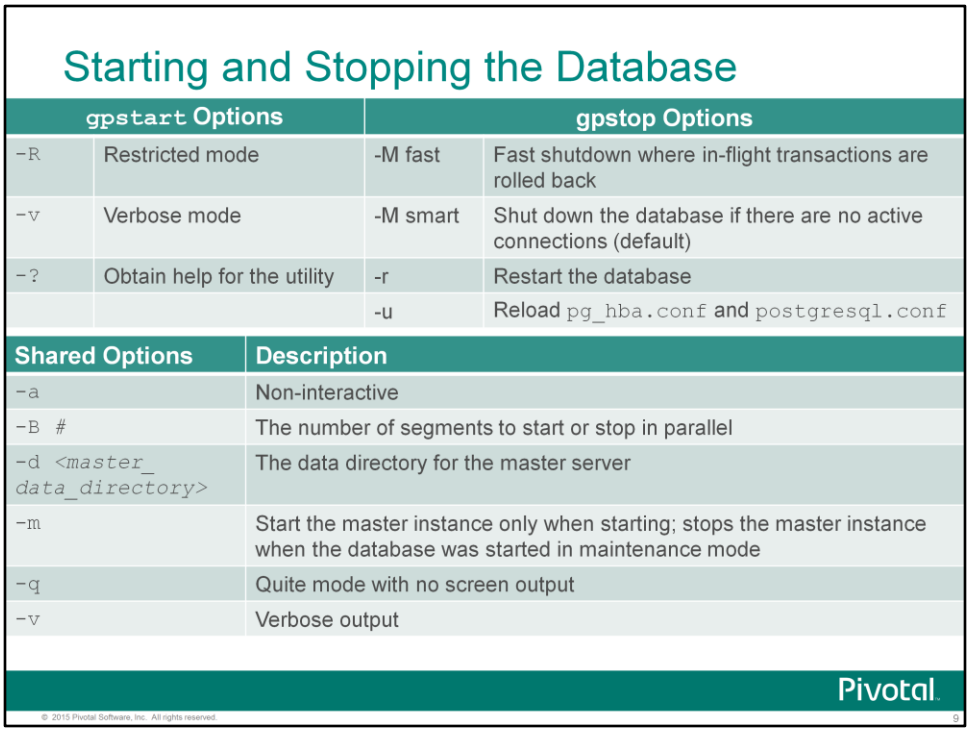
You will revisit these commands in greater detail as you continue through the course.

Greenplum Database Utilities

Utilities	Description
`gpstart`	Starts the database
`gpstop`	Stops or restarts the database and re-read configuration files
`gpstate`	Obtain the status of the database
`gpconfig`	Obtain or set database parameter values (GUCs)
`gpscp`	Secure copy between multiple hosts
`gpssh`	Secure access to multiple hosts
`gplogfilter`	Search Greenplum Database log files for specified entries

Pivotal

The commands listed are used for during everyday administrative and maintenance tasks for the database. We will discuss several of the commands here while you will encounter the remainder throughout the course.

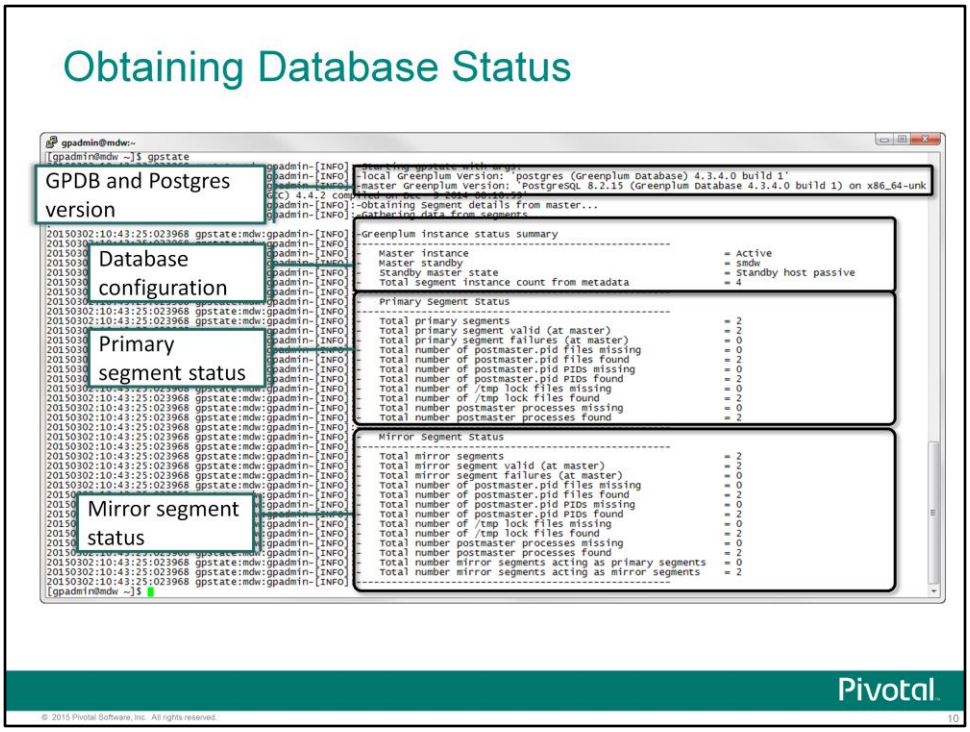


The `gpstart` and `gpstop` commands are used to start and stop the database respectively.

The `gpstart` command starts all database server processes on all servers and segments defined in the cluster. To start the database, you must either specify the location of the data directory on the master server on the command line using the `-d` option or set the `MASTER_DATA_DIRECTORY` environment variable. When performing maintenance tasks, you can start the master server only using the `-m` option, or start the database in restricted mode, using the `-R` option.

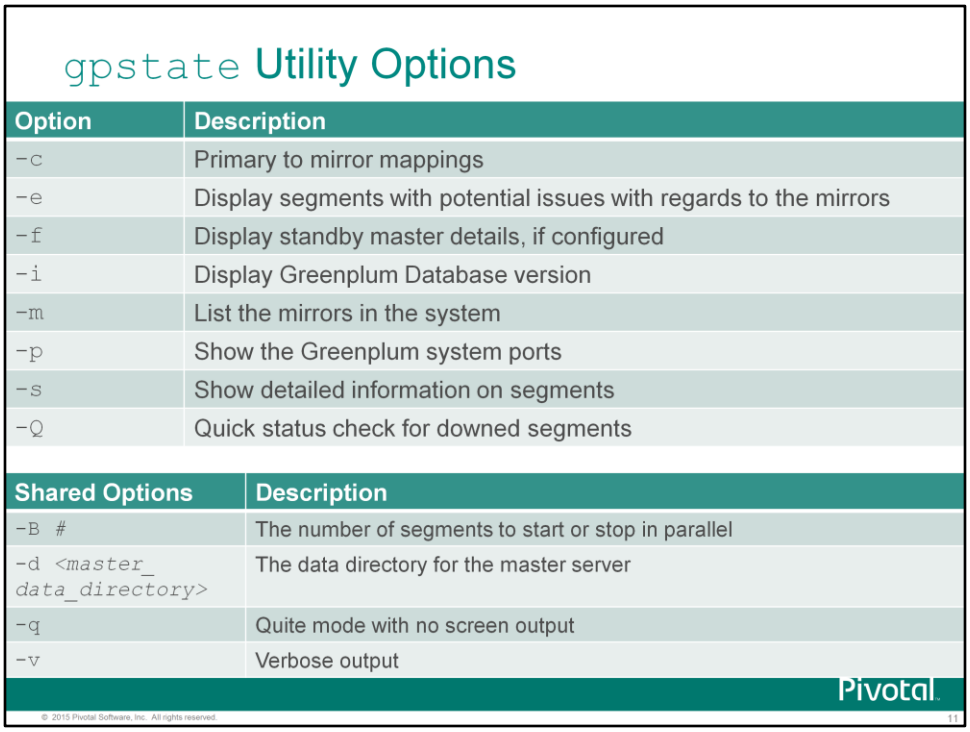
The `gpstop` utility is used to both stop and restart the database with the `-r` option in the latter case. All database server processes are stopped in parallel, but, as with the `gpstart` command, you can specify a maximum number of segments to start or stop in parallel. By default, the database is stopped using smart mode, where it is stopped when there are no active connections. You can force a database shutdown using the `-M fast` option. In addition to shutting down segments and the master servers, this option will also rollback any in-flight transactions.

In addition to stopping and restarting the database, the gpstop command also re-reads two Greenplum Database configuration files: `pg_hba.conf` and `postgreql.conf`. The `pg_hba.conf` configuration file controls access and authorization to the database, whereas the `postgresql.conf` file contains Greenplum Database configuration parameters. These will be discussed in greater details later in this course.



The gpstate utility is used to show the current state of the Greenplum Database. Without any options, or by using the `-b` option, `gpstate` displays:

- The overall state of the Greenplum Database instance
- Greenplum Database version information
- Master, primary, and segment configuration information



The `gpstate` utility lets you drill down into specific areas of your environment.

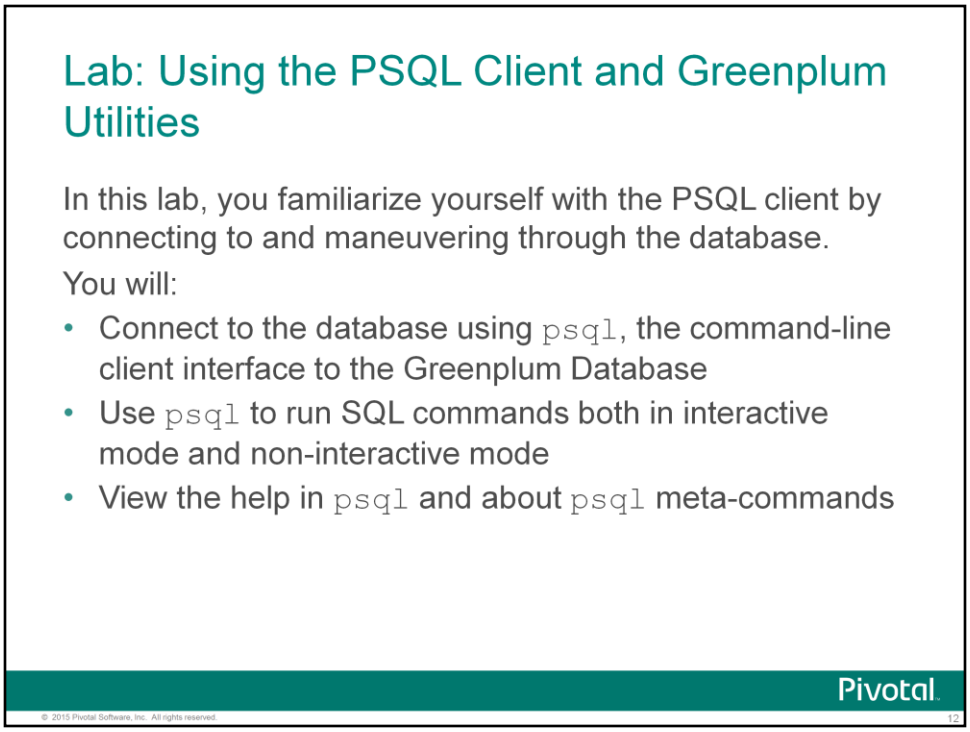
Issue the `gpstate -c` command to see the mirror to primary mapping for your environment. This, along with `gpstate -s`, is helpful when troubleshooting failed mirror issues. The –e option offers additional information on failed segments by displaying the status of the mirror segment and whether or not it is in its preferred role. A mirror in its preferred role will display as a mirror. However, if a primary has failed and the mirror has taken over for it, it will display that it is a primary segment. This is therefore not its preferred role. The option will also display if the segments are running normally or are synchronizing. The `-m` option lists a subset of the `-c` option, displaying the mirroring method, information on the mirrors themselves, and their data directory. It also shows whether or not a mirror is synchronized and if it is active, in which it is servicing requests, or passive, where it is behaving as a mirror.

The `gpstate -f` command displays information on the standby server, including its data directory, port, process ID, and status. It also provides information on the replication status, whether it is in sync with the master.

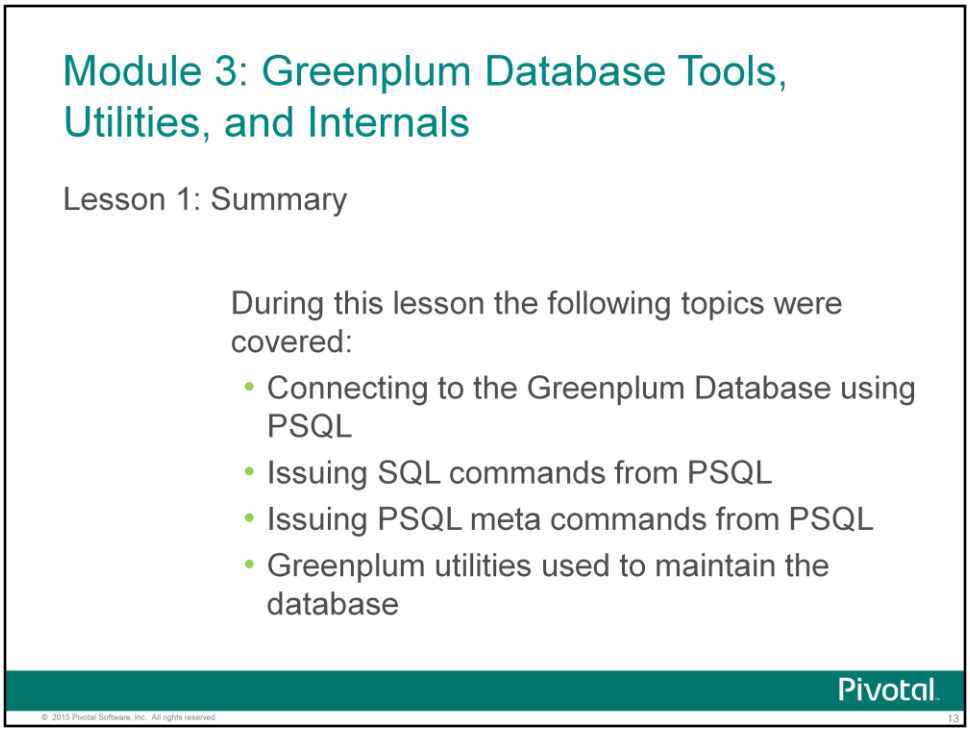
Several options provide a quick view of the system, including:

- `-i`: This option displays the GPDB and Postgres version information
- `-p`: This option displays the port numbers used throughout the GPDB environment
- `-Q`: This option displays a quick status check on the environment with a focus on down segments.

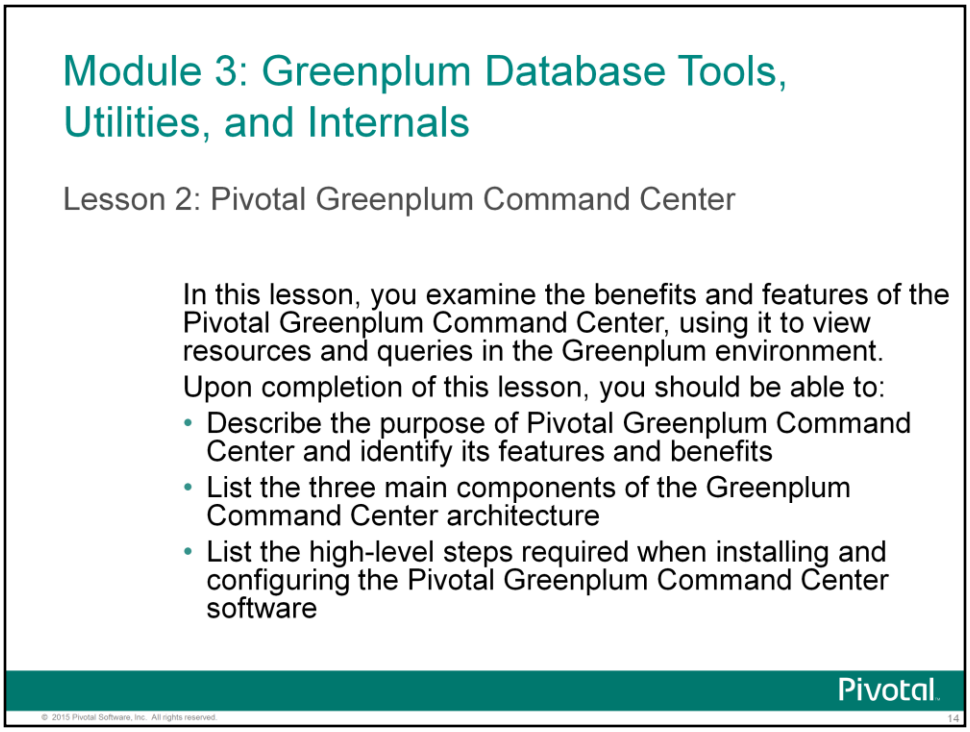
Just as with other utilities, the `gpstate` command also supports the `-B`, `-d`, `-q`, and `-v` options.



In this lab, you familiarize yourself with the PSQL client by connecting to and maneuvering through the database.



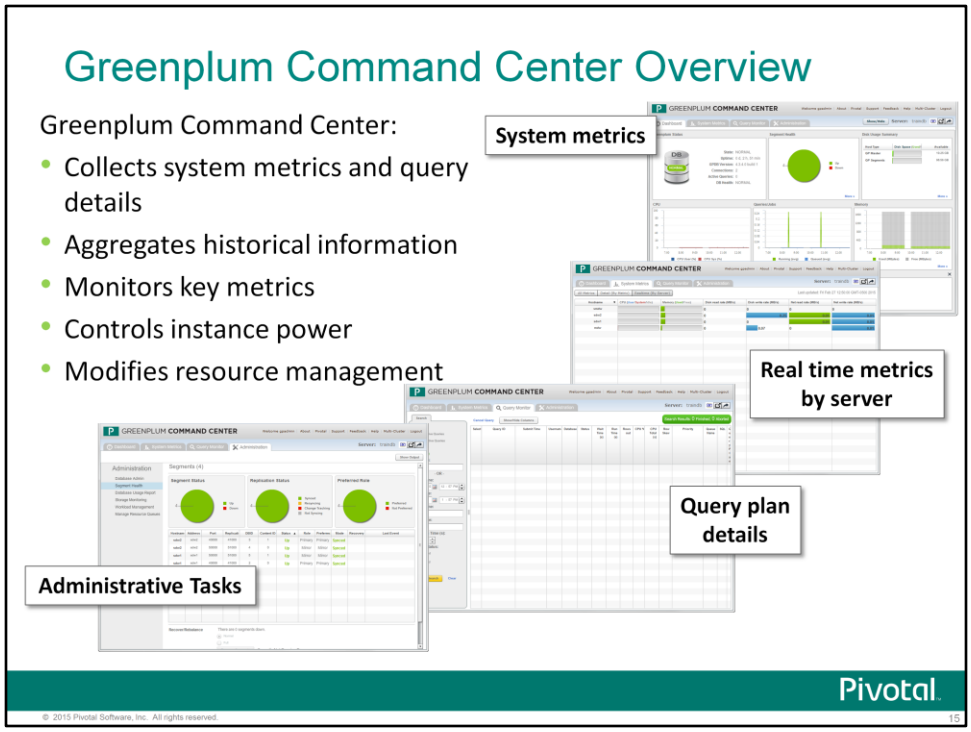
This lesson covered the how to access the Greenplum Database using the PSQL client. It also covered obtaining help on meta and SQL commands and issuing these commands from the client. An overview of several Greenplum-specific utilities was also covered, including creating and dropping databases and users from the UNIX command line.



Pivotal Greenplum Command Center, also known as Greenplum Command Center, enables you to monitor several aspects of the Greenplum environment, allowing you to view how resources are being consumed by Greenplum users, as well as to take an in-depth look at the queries currently executing.

In this lesson, you will:

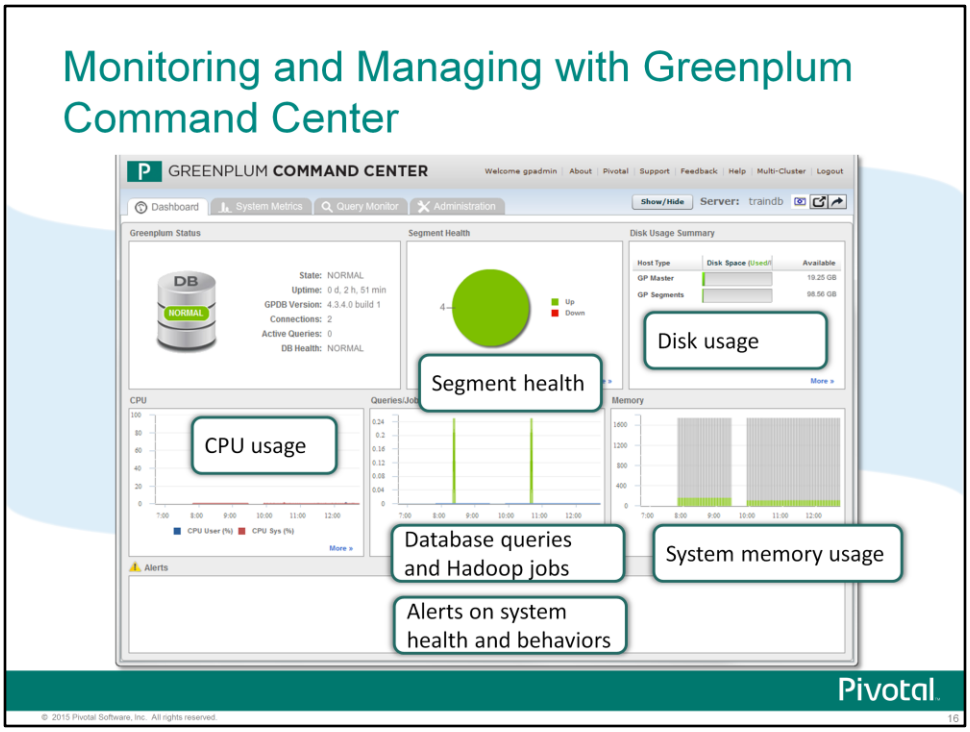
- Examine the overall purpose, features, and benefits of Greenplum Command Center.
- Identify the three main architectural components of Greenplum Command Center as well as examine the communication pathway among these components.
- Examine the steps that will be required to install the Greenplum Command Center server and console.



You can use Greenplum Command Center to collect query and system performance metrics from a running Greenplum Database system. You can also perform several administrative tasks with this tool. You can track how the system performs over time as the Greenplum Command Center also stores historical information on system metrics.

Greenplum Command Center uses monitoring agents to collect information on all servers within the DCA system. The type of information collected includes:

- CPU utilization
- Queries running and queued
- Memory utilization
- System load
- Disk I/O activity
- Network throughput
- Swap activity
- Real-time performance per server
- Detailed information on each query
- Health monitoring details on each server and switch within the Pivotal DCA as well as the status of all components within those systems. This is available only when monitoring Greenplum Database on a Pivotal DCA.



Greenplum Command Center is a monitoring and management interface for the Greenplum environment. It lets administrators not only monitor system metrics and access alerts, but also provides interactive dashboards for accessing detailed information on system health as well as monitoring and managing configuration parameters that affect workloads.

The user interface provides access to dashboards that allow you to view varying aspects of system and database behaviors.

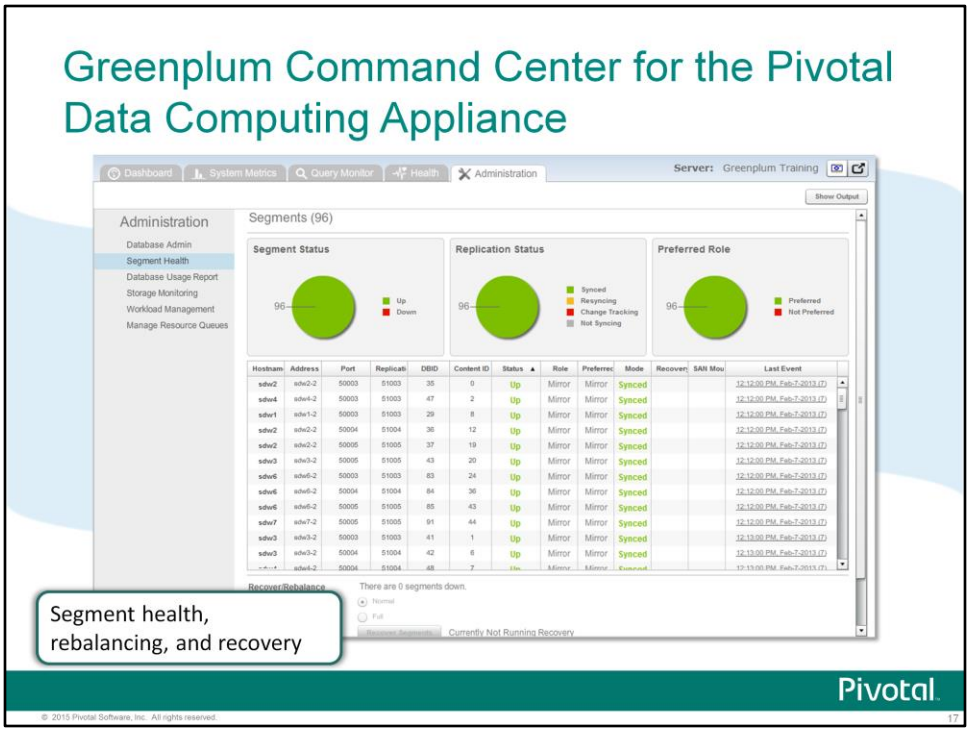
Greenplum Command Center is available with each release of the Greenplum Database software starting with GPDB 4.2 service pack 1. It is also available with each release of the Pivotal DCA software starting with the same release.

The Dashboard is your first tab after logging into Greenplum Command Center and provides an overview of how the system is performing in real-time along with some historical trends.

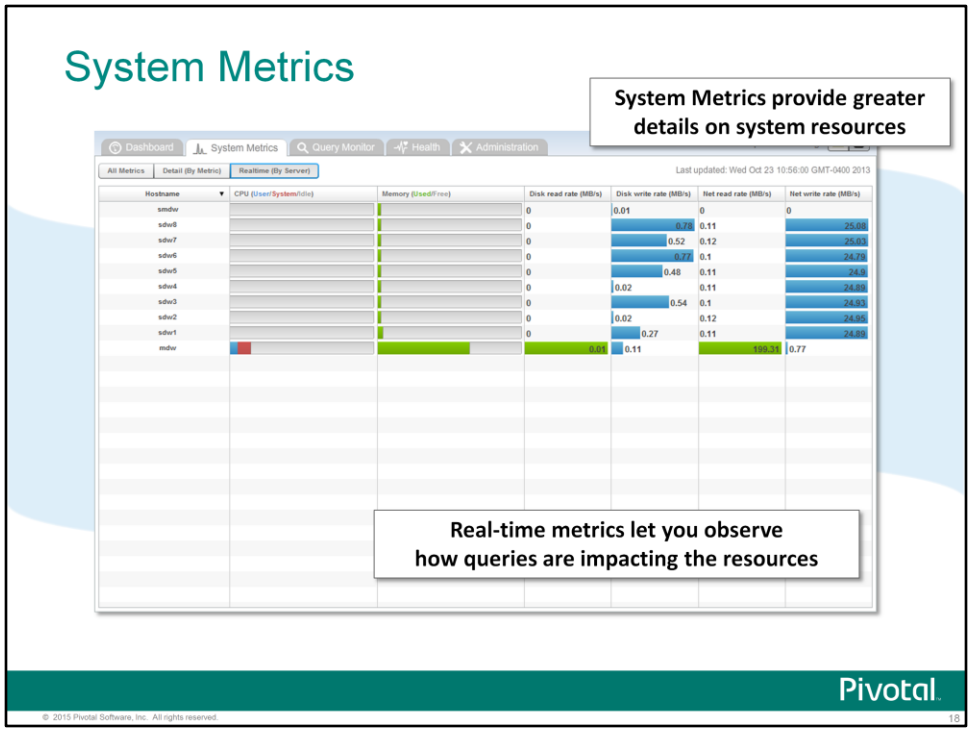
You can obtain information on the CPU, any queries that are running or queued, the memory that is currently being used, overall load, disk, swap, and network performance.

You can also obtain information on active queries and the resources that they are consuming. From this page, you can drill-down into greater detail on each query.

When used to monitor and manage an environment installed on the Pivotal DCA, Greenplum Command Center provides an overview of the infrastructure, including overall and component health.



Greenplum Command Center can be installed to monitor a Greenplum Database software-only environment or a Greenplum Database environment on the Pivotal Data Computing Appliance. When installed for the Pivotal Data Computing Appliance, it provides a hardware monitoring view that lets you track metrics on the Greenplum Database segment servers, master, or standby server as well as the Hadoop worker and administration nodes. Real-time metrics on disk reads and writes, network reads and writes, and CPU and memory usage are available. In addition, overall segment health and replication or mirror health is monitored.



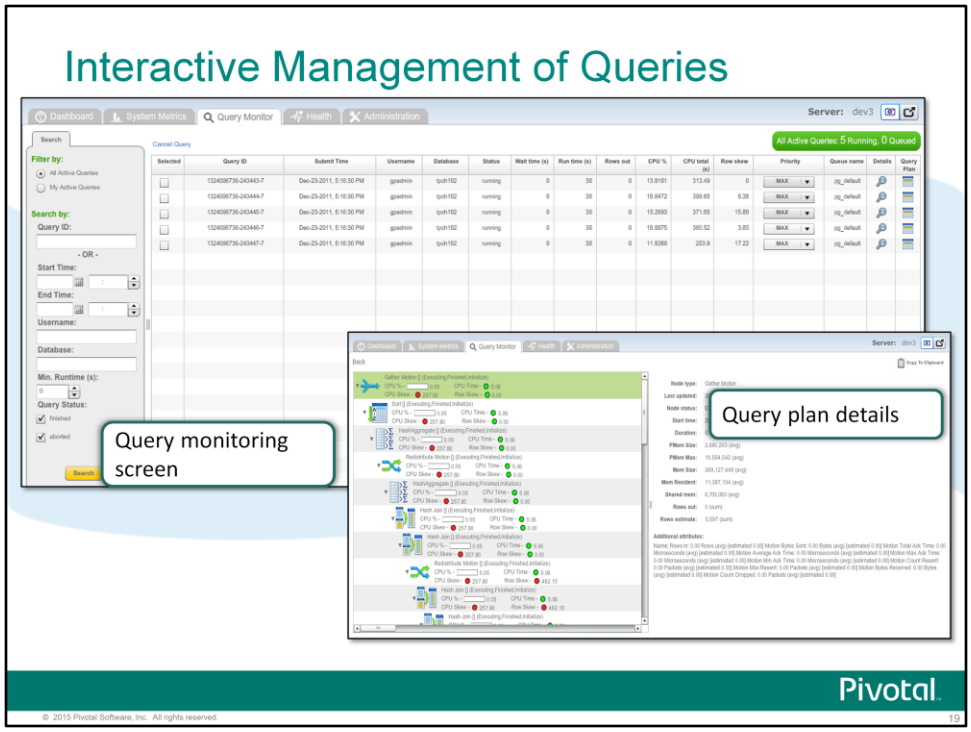
The System Metrics tab allows you to see greater details on system resources, again showing the CPU resources, swap, network, disk I/O, memory usage, and system load. You can additionally view details by a specific metric or you can examine the real-time performance of the system as it is being used.

The System Metrics tab will show historical information for a time period you specify, starting from an hour and extending up to a year, if the information is available. All data is stored in a separate database within the Greenplum environment and is available until manually purged.

The System Metrics tab can give you a good understanding of when usage of the Greenplum Database is at a peak and what the impact at those times can be.

A great benefit to using the real-time metrics is to examine how one or more queries are affecting the resources. Information on the CPU resources, memory, disk read and write rate, and the network read and write rate gives you a clear picture on how queries are affecting the system. This can show if you have a balanced environment or if one or more servers tend to be working harder than others.

By displaying and tracking this information, you can determine if your data is balanced appropriately for the type of jobs that is being executed. When one set of segment servers appears to be dedicating more CPU cycles to a query, it tends to show that there is a processing skew. The same for disk read and write rates. Overall, the huge benefit that Greenplum offers is the ability to balance the load on systems based on the type of queries that are run. This tool helps you to visualize performance and act on it if necessary.



With Greenplum Command Center, you can manage active queries on the Query Monitor tab. Previously with the Greenplum Command Center console, you could only view queries. With Greenplum Command Center, you can view the resources that a query is consuming and the user who initiated the query. By monitoring the query, you can determine whether or not the query is awaiting access to the active running queue, cancel queries, and view detailed information on the query plan associated with the query.

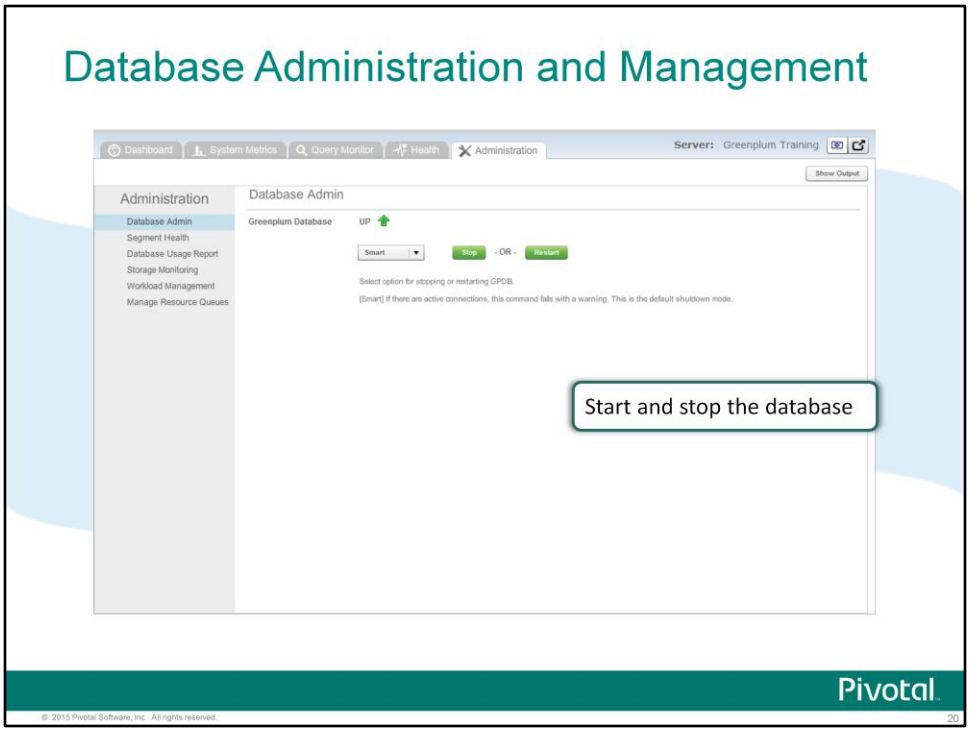
The Query Monitor tab lets you see which queries are actively running and queued and allows you to get detailed information on each of those queries. This is particularly effective when working with long running queries. Shorter queries may actually finish too quickly to have much, if any, information captured or even available in real-time analysis.

The active queries mini-dashboard allows you to see how much resources the query is using, the user issuing the query, and the database they are connecting to, the run time, CPU, and row skew. We will cover details on skewing later in the course.

Clicking the details icon lets you see the actual query content and system information on that query, including its priority in the run queue, and the application that was used to issue the command.

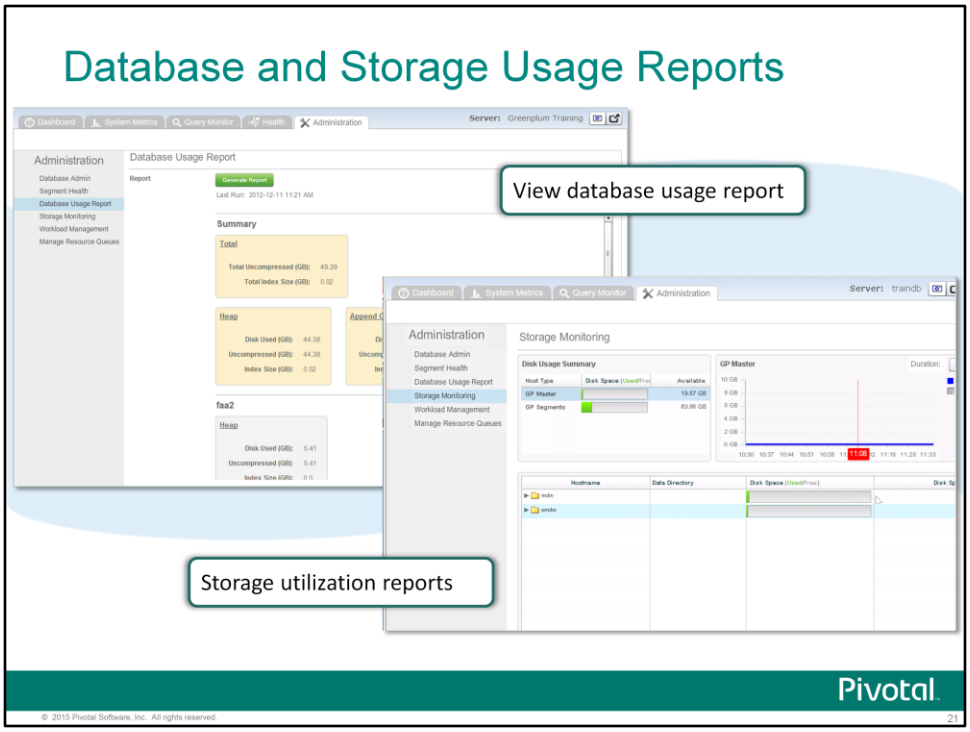
You are not only exposed to the explain plan for the query, but details on the query are displayed.

Now that we have discussed the main features and benefits of the Greenplum Command Center, let us take a look at the architecture.



Some of the database administration tasks can be performed using the Greenplum Command Center console. You can:

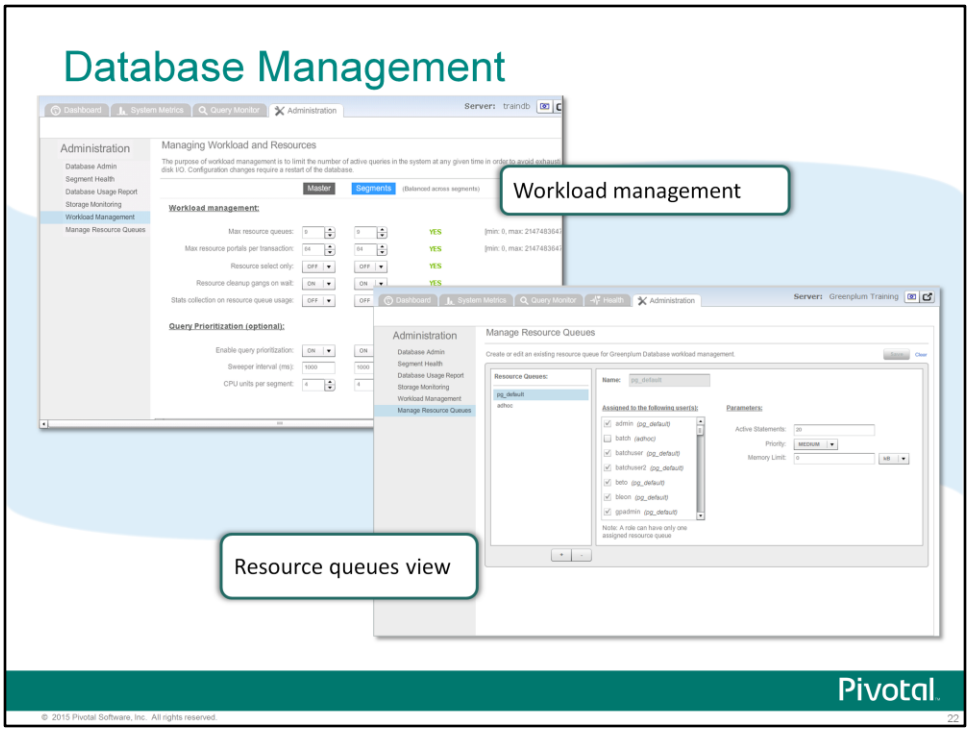
- Start and stop the database using the Database Admin screen. Smart, fast, and immediate shutdown are supported.
- Segment server rebalancing and recovery are accessed through the Segment Health screen. The visual indicators make it easier to see which segments are not acting in their primary roles, which segments are down, and provides an method of recovering the segment.



You can monitor storage usage for the database by viewing database utilization reports and current storage utilization reports.

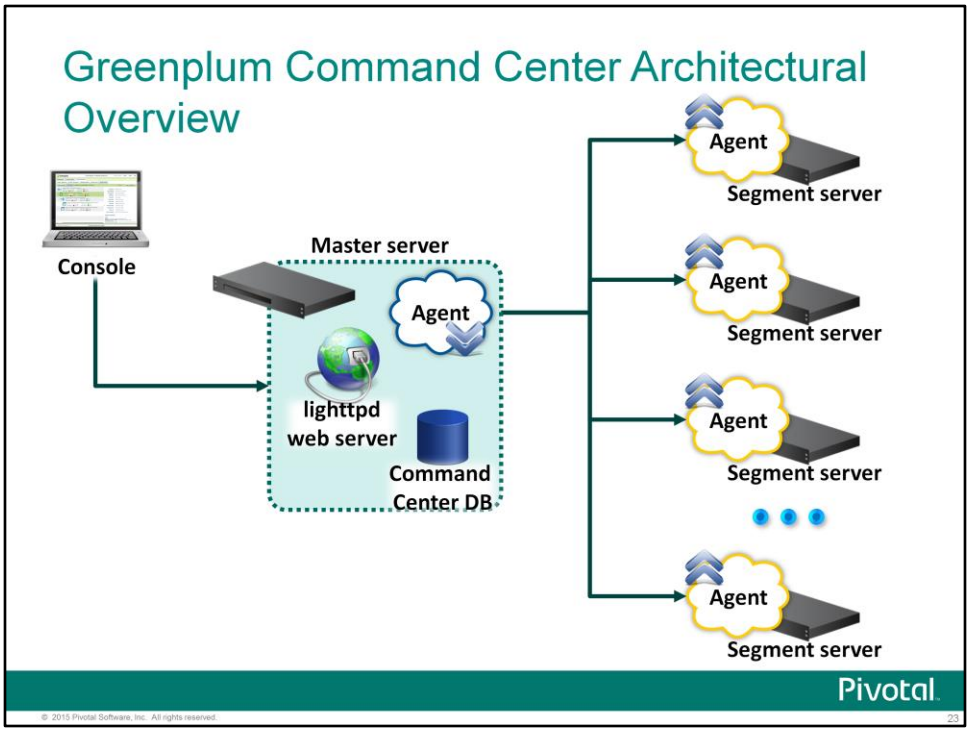
The database usage report lets you view how the disk is being utilized by the database, including the amount of disk space consumed by heap or regular tables, append-only tables, and column-oriented tables.

Storage monitoring lets you view the amount of disk space utilized by each server within the cluster. By actively keeping track of the disk space on each segment, an administrator can work to keep disk utilization below 70%. Filespaces are also easily identified within the storage utilization reports along with the actual filesystem or directory to which the filespace is associated.



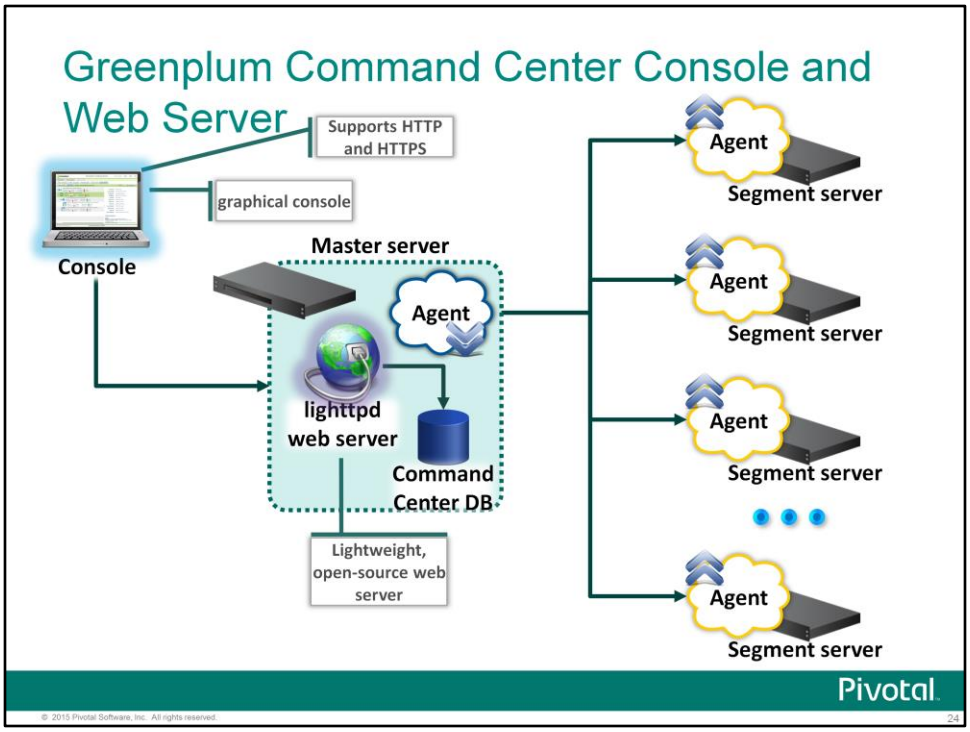
The Workload Management screen allows you to view and make changes to the Greenplum Database configuration parameters for managing workload and resources. The parameters are displayed at both the master and local, or segment, levels and are classified according to their potential effect on the database to make it easier for you to understand which parameters affect a specific area of the Greenplum Database.

You can also view the resource queues that exist within the environment as well as which queries are currently running within those queues. Management of the resource queues through the Resource Queues screen will be available in future releases of the Greenplum Command Center.



Greenplum Command Center is comprised of:

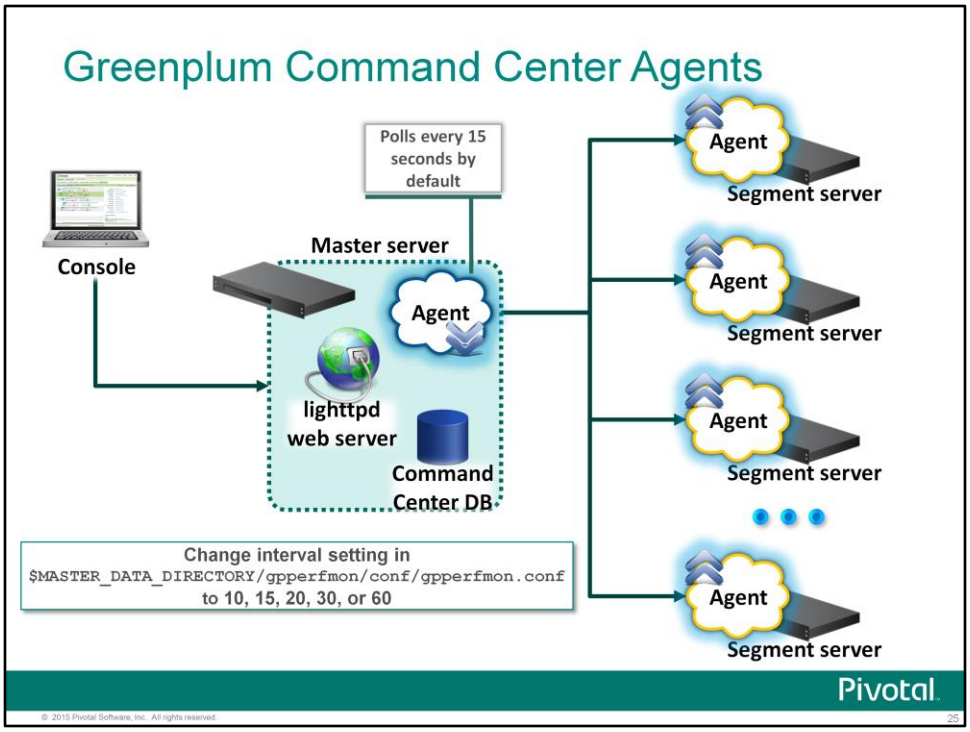
- Greenplum Command Center Console, a graphical browser interface which features the Greenplum Command Center Web Service, a lightweight lighttpd web server.
- Greenplum Command Center data collection agents which run on the master and segment servers.
- Greenplum Command Center database, `gpperfmon`, which is dedicated to storing and serving performance data.



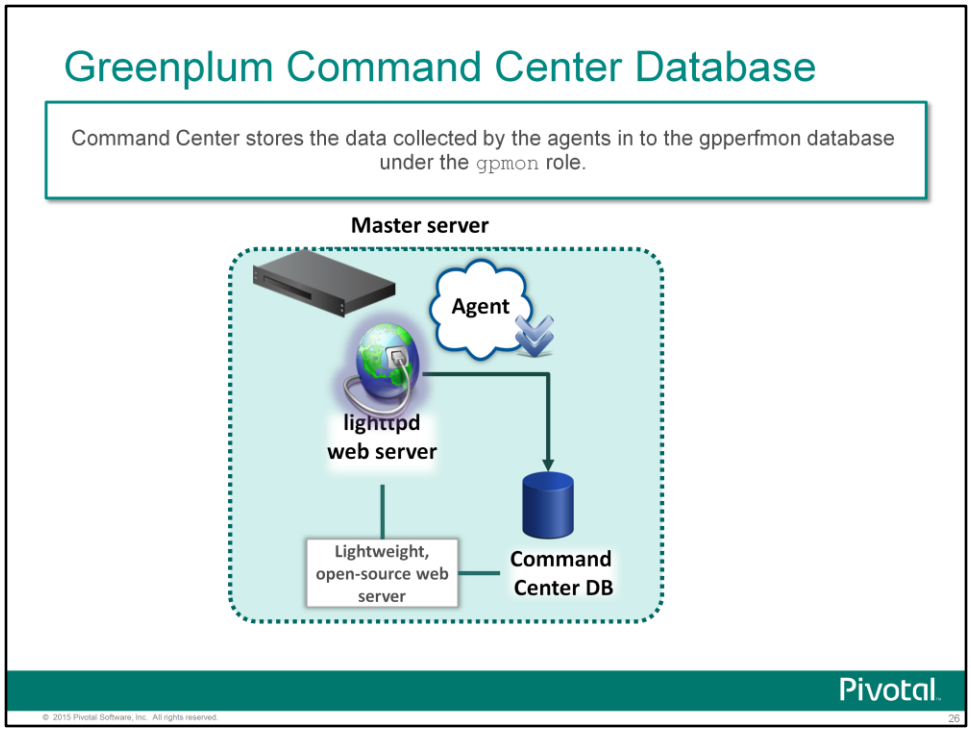
The Greenplum Command Center Console consists of a flash-based graphical console accessed through a browser and used to view performance metrics for all servers in the cluster. You can access separate Greenplum instances from the console by assigning each instance its own unique port ID.

The console queries the monitor database through a web service framework that consists of a lightweight lighttpd web server and a Python-based middleware infrastructure. This web service framework is an open-source web server with low-memory and low-CPU load requirements.

The console can be accessed through HTTP or HTTPS protocols that you enable when installing and configuring the console. The console can be installed on the master server or on a remote server that can access the master server.



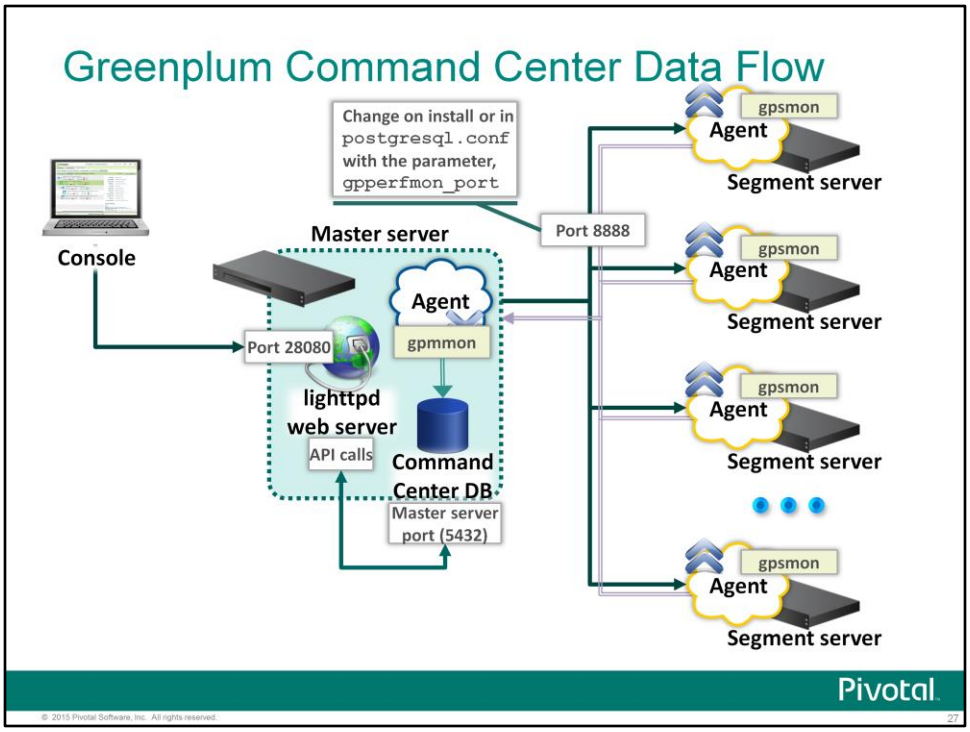
The Greenplum Command Center agent on the master server polls all agents on other servers in the DCA for performance data. By default, this occurs every 15 seconds. The interval can be changed by updating the `quantum` variable in the `gpperfmon.conf` file stored in the `$MASTER_DATA_DIRECTORY/gpperfmon/conf` directory. Decreasing the interval lets you collect additional data for system metrics, allowing you to capture shorter running queries. Keep in mind however that decreasing the interval can potentially crowd out other queries.



The Greenplum Performance Database, `gpperfmon`, stores and serves performance data collected by the agents. Data is stored across three types of tables:

- Tables with `_now` appended to the end of the table name are external tables that pull information from data files stored in the `$MASTER_DATA_DIRECTORY/gpperfmon/data` directory. The files contain data on current metrics collected during the period between when data are collected from the monitor agents to when the same data are committed to the database.
- Tables with `_tail` appended to the name are external tables that contain transitional information cleared from the *`now`* tables that have not yet been committed to the database itself. These tables typically contain a few minutes worth of data.
- Tables with `_history` appended to the name are regular tables that are partitioned into monthly partitions and contain all the information gathered from the *`tail`* tables.

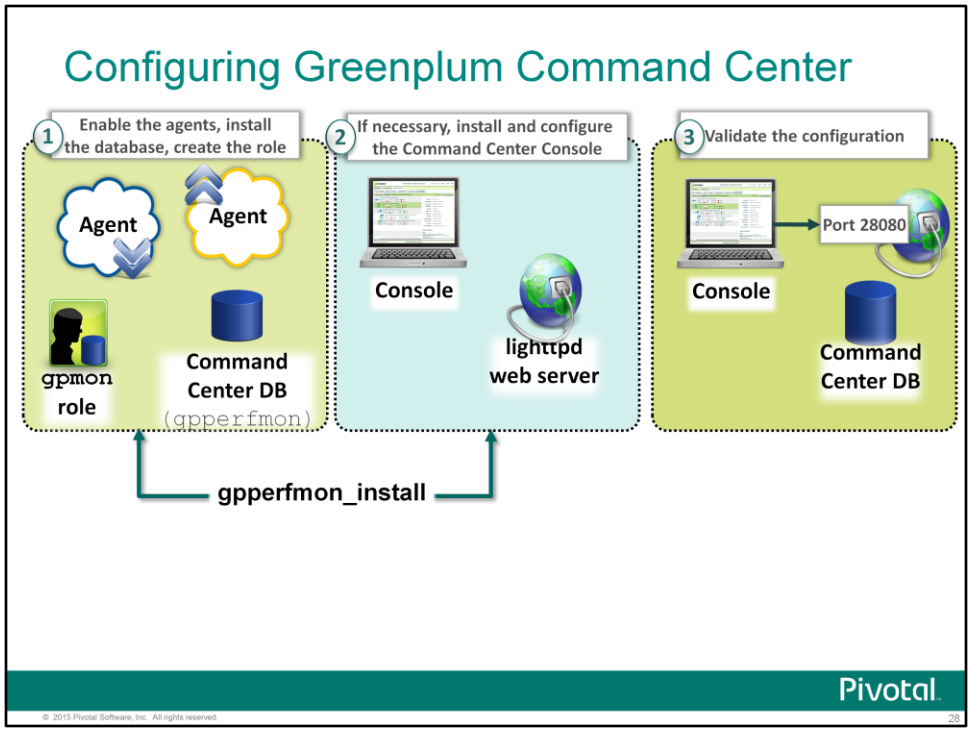
Note that while the gpperfmon database can provide historical data, it should not be used for accounting as it does not trap all events that has happened. What you are seeing is a sampling provided over a period of time. This is to prevent Greenplum Command Center from hogging system resources which could preventing or delaying other queries from running in a timely manner.



A client accesses the console through the port for the instance defined. By default, port 28080 is made available for assignment. Each instance you define must have a unique port. Once the connection has been established, API calls to the Command Center database retrieve current and historical information and make the data available to the graphical console. You can view the same information by using client programs such as psql or through JDBC or ODBC.

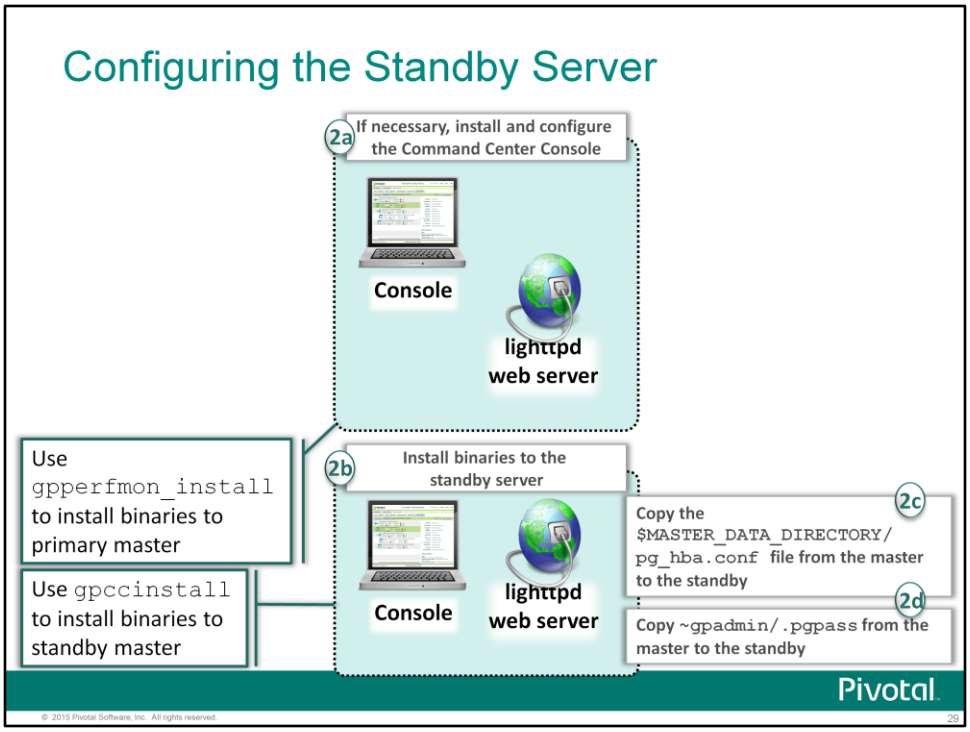
The agent on the master, `gpmmon`, is responsible for gathering information from the agents on the segment servers, `gpsmon`. The agent on the master communicates with these agents over TCP/IP on port 8888 by default. The port number can be changed while installing Greenplum Command Center or after installation by modifying the `gpperfmon_port` parameter in the `postgresql.conf` file on the master, standby, and segment servers.

Information is written to *`now`* files which are available to the external tables. After a period of time, data collected in the *`now`* files are written to the *`tail`* files. This information will eventually be committed to the Command Center database, `gpperfmon`.



You perform three main tasks when enabling Greenplum Command Center:

1. Enable the Greenplum Command Center collection agents, create the Greenplum Command Center database, and create the superuser for the Command Center database. This step is performed by the `gpadmin` user with the `gpperfmon_install` command. During the installation, you specify the password for the Greenplum Command Center superuser role, `gpmon`, that will be created.
2. If you are installing the Greenplum Command Center Console on a remote system, you will use the `gpccinstall` command to do so. Otherwise, the console has already been installed on the master server. The console allows you to configure a separate Command Center instance for each database instance you will be monitoring. Remember, the database instance is not the user database, but is instead the Greenplum Database instance you installed within the environment.
   After the console has been installed, you can create provide users with specific roles to access the console. You will then start the Greenplum Command Center instance you created.
3. Validate the installation was successful. This final step is used to ensure you can log into the environment and validate that information is being collected for the Greenplum environment.



During the setup process, you are given the opportunity to setup Greenplum Command Center on the standby server. If you plan on being able to access Greenplum Command Center from the standby server should the master server become unavailable, you must install the software to the standby server. In the event the master server becomes unavailable, the standby server can service requests for Greenplum Command Center.

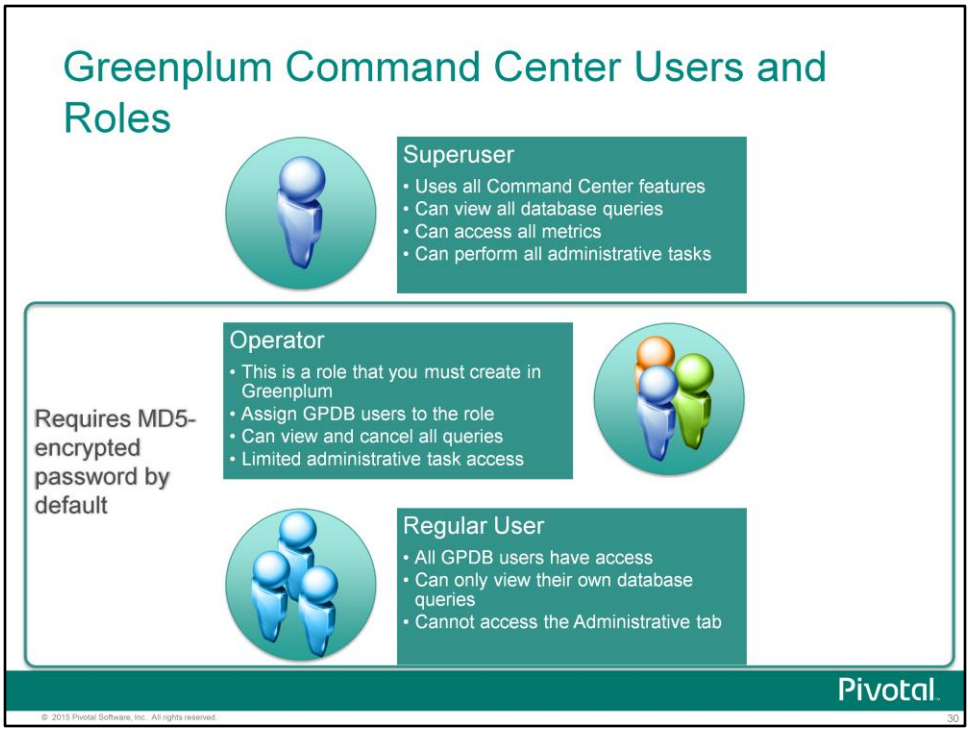
To perform the installation, first, populate a file with the host names of each server for which you would like to install the Greenplum Command Center binaries. Each server hostname must be on a line by itself and must be resolvable.

Assuming the filename is `gpcc_hosts`, the command to perform the installation is:

```
$ gpccinstall -f gpcc_hosts
```

After installing the software, you will also need to synchronize your copy of the `$MASTER_DATA_DIRECTORY/pg_hba.conf` file on the master with the standby server. This ensures that all of the required connections are also available if you need to connect using the standby server.

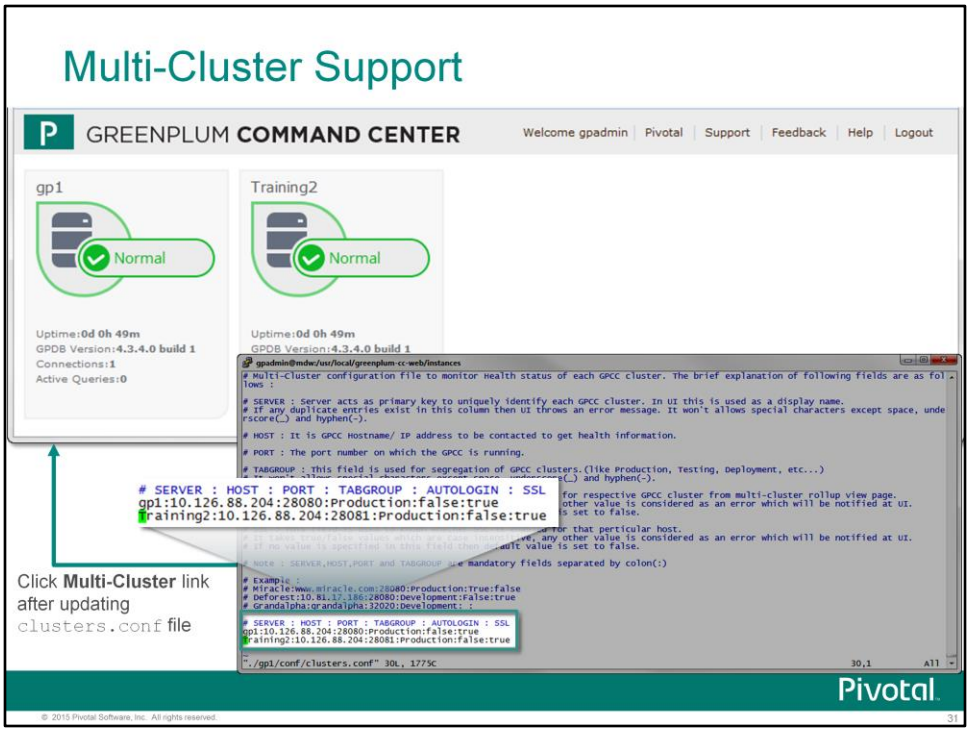
A last step is to copy the `~gpadmin/.pgpass` file from the master to the standby master. This is the authentication file for Greenplum Command Center. The permissions must be set to 0600 (read-only for the owner).



There are three main types of users that can access the Command Center console. These include:

- Superuser – Users assigned the Greenplum Database `SUPERUSER` attribute have full access to all features and tabs within the console. This user can view all database queries and system metrics. By default, the `gpadmin` user is assigned the `SUPERUSER` attribute. The `gpmon` user is granted the `SUPERUSER` attribute and therefore has full access to all features of the console. This database user was created as part of the installation process and is used to manage Greenplum Command Center components and data . The `gpmon` user should not be used for logging into the console. This user is protected with MD5-encrypted password by default.
- Operator – The operator role must be created after installing Greenplum Command Center. The role, `gpcc_operator`, can then be granted to users to provide them with operator permissions. These permissions include viewing and canceling all queries, whether they are owned by the user or not. This user cannot stop or start the database, make changes to workload management parameters, or change resource queues.
- Regular User – All other users which do not have the `SUPERUSER` attribute and are not a part of the operator role have access to the Greenplum Command Center console with the ability to manage their own database queries. They cannot view or access queries associated with other users.

Greenplum Command Center requires MD5-encrypted password authentication by default. Greenplum Database and Command Center both support SHA-256 and so can be used instead.



Greenplum Command Center provides a multi-cluster support, where you are able to obtain a quick view of the status of all the Greenplum Database clusters you monitor with Greenplum Command Center.

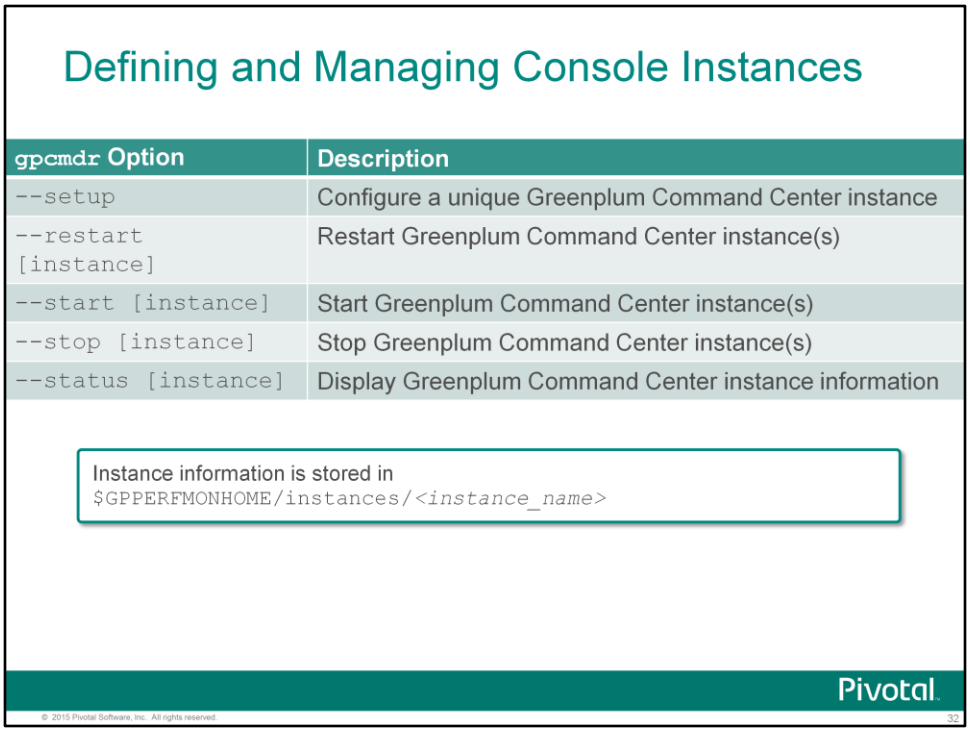
For multi-cluster support, you choose a Greenplum Command Center instance to act as the master. From this master instance, you can view the Multi-Cluster page which provides a status of the instance. You can launch the user interface for each of the individual instances you manage.

To configure multi-cluster support, modify the `$GPPERFMONHOME/instances/<instance_name>/conf/clusters.conf` file and add each cluster you wish to monitor. An example is shown on the screen.

There are a few items to note when modifying the file:

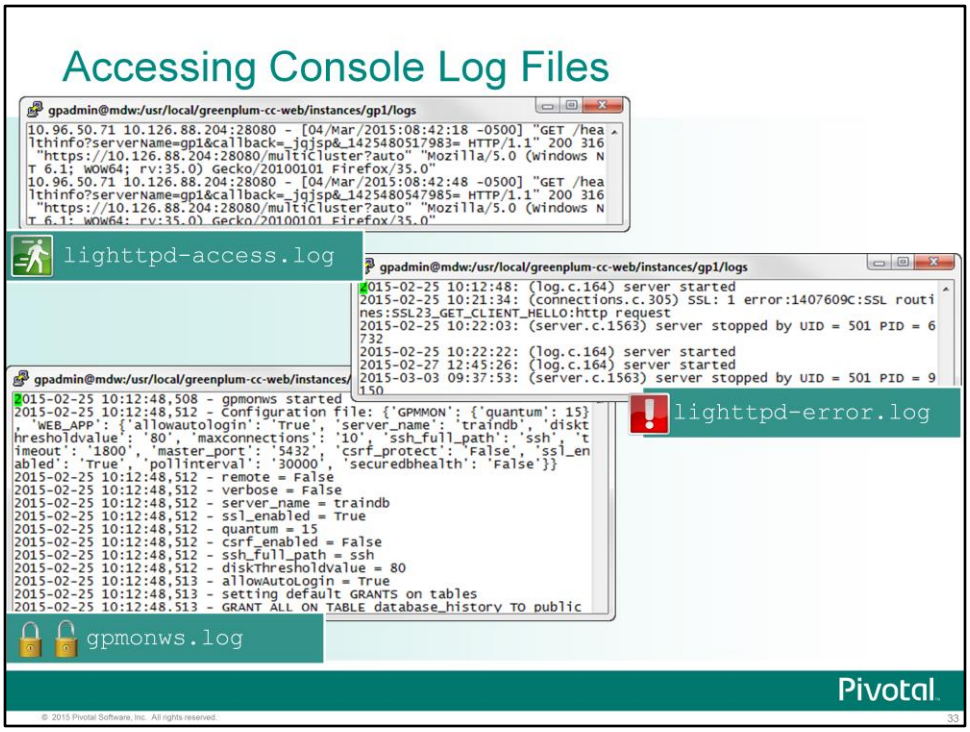
- The first field is the unique ID for the instance. This is how it will be displayed on the Multi-Cluster page.
- The second field takes a resolvable hostname or IP address
- The third field contains the port number for the instance
- The fourth field allows you to categorize the clusters into different groups
- The last two fields are not case sensitive. However, the only valid values are true or false.
- All hosts you are defining must have the same SSL configuration. They must all either have SSL enabled or disabled.
- Autologin support is available and uses the username and password you used to log into the master instance.

Once the file has been updated, you can access the Multi-Cluster page from the Multi-Cluster link when you login to the master instance.



The `gpcmdr` command is used to define and manage Greenplum Command Center instances. Once defined using the `gpcmdr --setup` command, configuration information for the instance is stored in the `$GPPERFMONHOME/instances/<instance_name>` directory.

You can start, stop, and restart instances with the `gpcmdr` command. You can identify a single console instance to manage by specifying the instance name, or manage all services by not specifying an instance name.



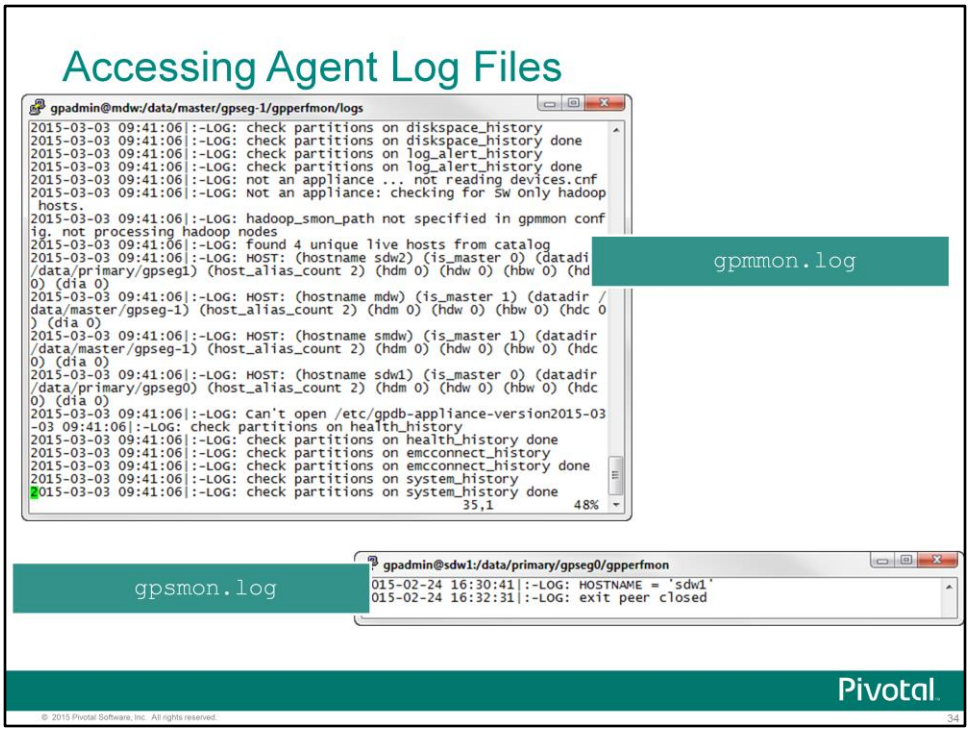
Errors and access information are written to log files within the instance directory. The log files, stored in `$GPPERFMONHOME/instances/<instance_name>/logs`, are:

- `lighttpd-access.log` contains the access information to the instance
- `lighttpd-error.log` contains any errors for the instance

Authentication errors can be found in the log file, `gpmonws.log`, in the same directory. The log contains a trail of the configuration parameter values and permissions for the instance from the time the instance was created and for each start and restart of the instance.

The file locations for the access and error log files are defined in `$GPPERFMONHOME/instances/conf/lighttpd.conf`.

The log files can be maintained with an external log control utility such as `logrotate` or `cronolog`.

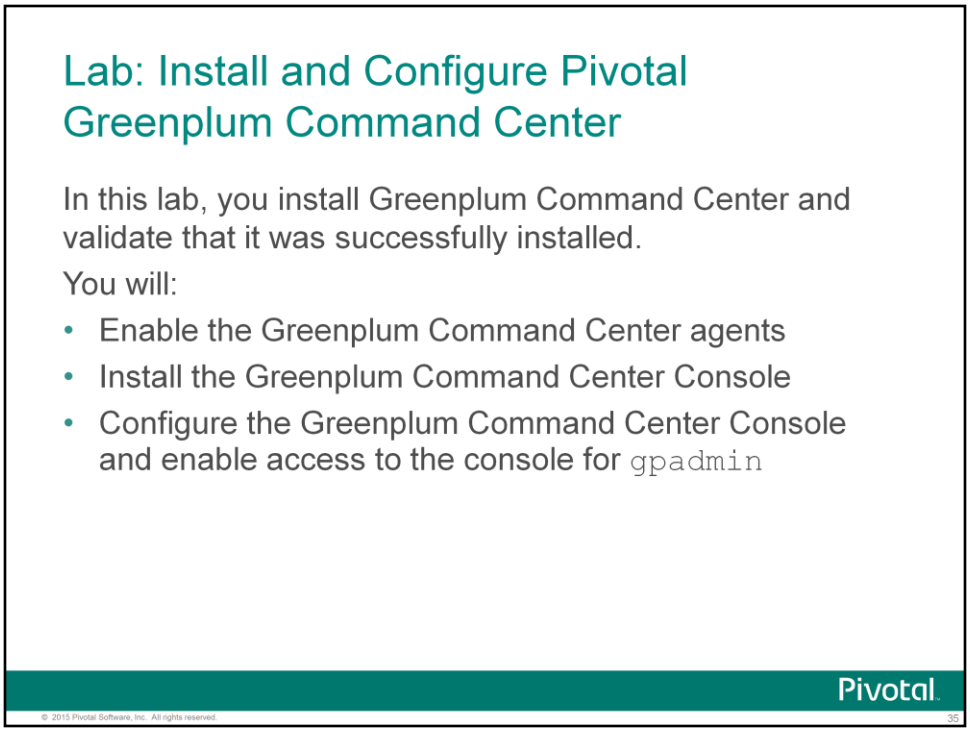


Log files for the master agent are stored as `$MASTER_DATA_DIRECTORY/gpperfmon/logs/gpmmon.<timestamp>.log`. This contains messages highlighting communications to other cluster agents.

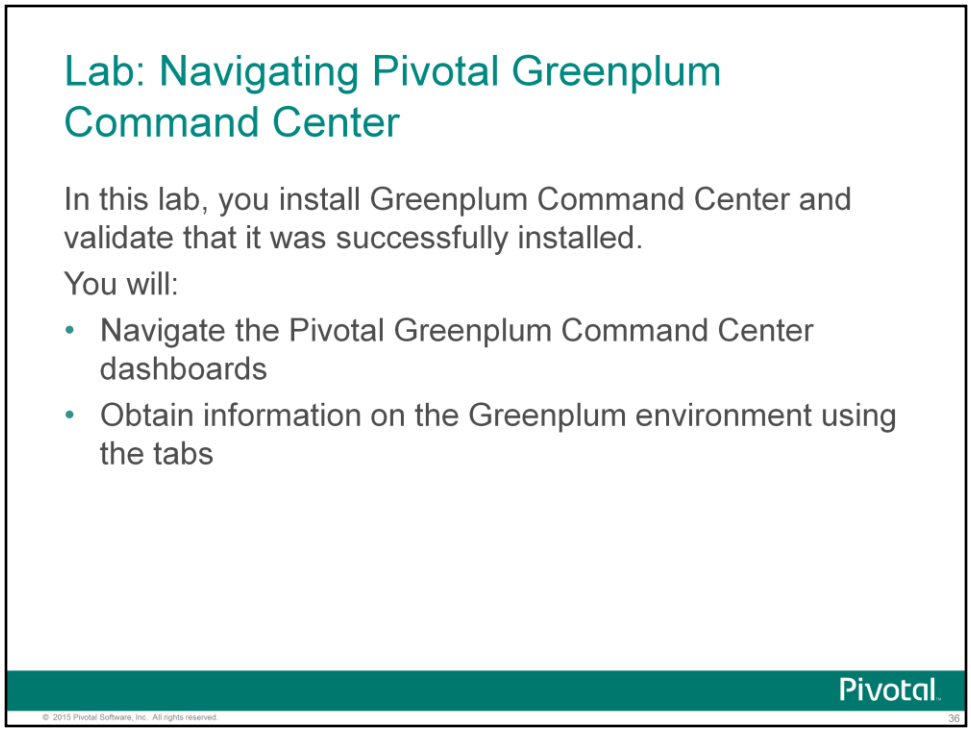
Agent log files for the master and standby master are stored in the same directory as `gpsmon.<timestamp>.log`. On segment hosts, the log files are written to the same filename but are stored in the first segment's data directory. For example, in our class configuration, the files are stored in `/data/primary/gpseg0/gpperfmon`. The agent log contains any specific logging information pertaining to that agent.

The log size is maintained by the `max_log_size` parameter in the `$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf` file. The value of this parameter is measured in bytes.

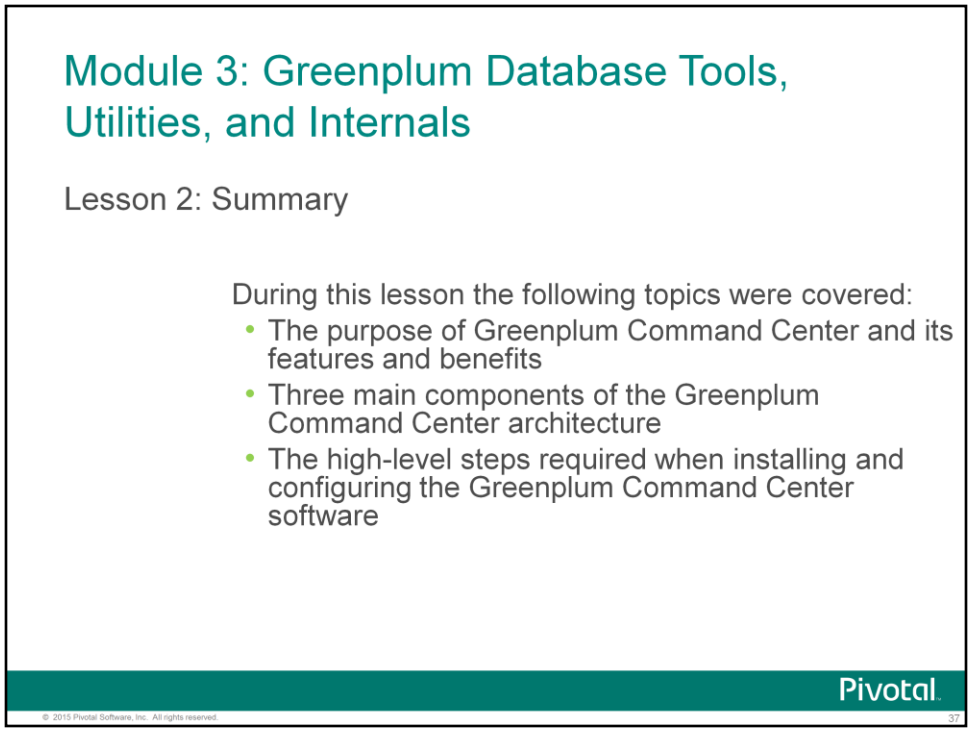
The log alert level can be adjusted by modifying the `gpperfmomn_log_alert_level` parameter which is stored in the `$MASTER_DATA_DIRECTORY/postgreql.conf` file. Use the `gpconfig` command to change the alert level from its default of `warning` to any other valid levels.



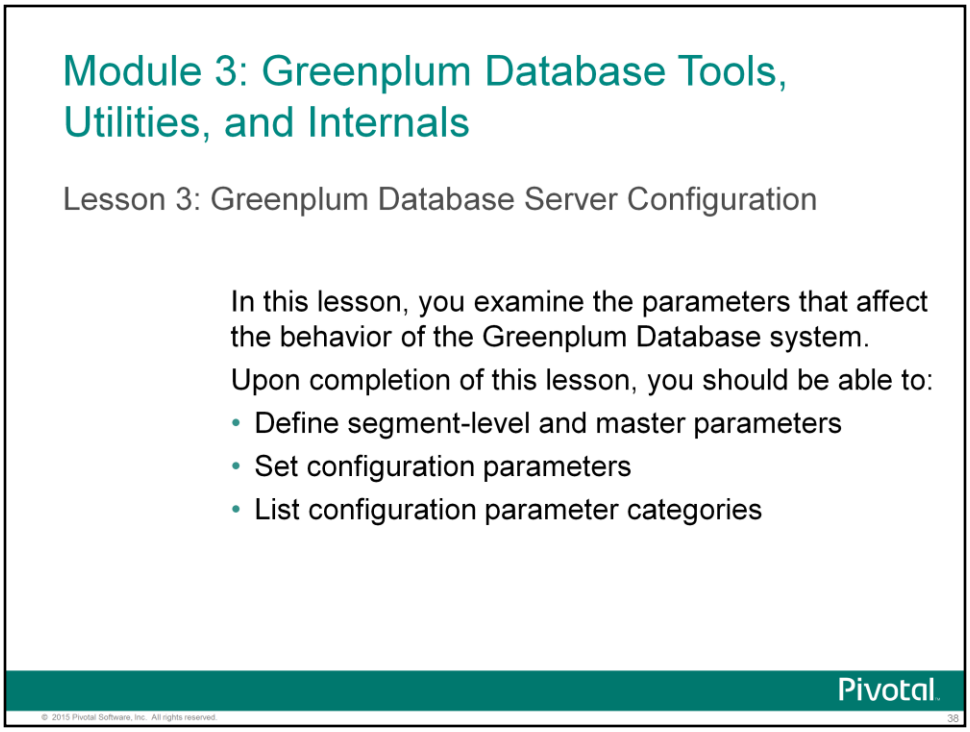
In this lab, you enable the Greenplum Command Center agents and install the Greenplum Command Center Console. After configuring the console, you will explore the environment, issuing commands to verify that the environment has been successfully installed and configured.



In this lab, you use Pivotal Greenplum Command Center to obtain information on the Greenplum environment, monitor specific aspects of the environment, and reset the environment.



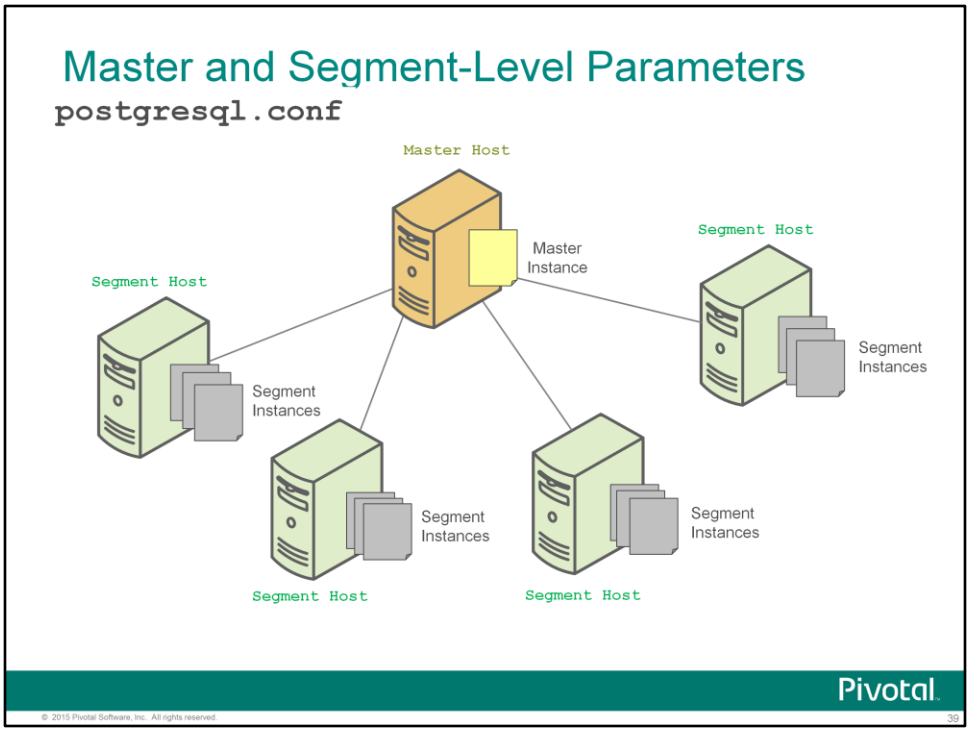
This lesson provided an overview of Greenplum Command Center, its features, and it benefits in monitoring and managing the Greenplum Database environment and providing status of system resources. The lesson also covered the main components found within the architecture and how they communicate with each other. Finally, the lab provided an opportunity to install and configure Greenplum Command Center.



There are many server configuration parameters that affect the behavior of a Greenplum Database system, including the debug level or the behavior of write ahead logs.

In this lesson, you:

- Define and compare segment-level and master parameters
- Examine how to set configuration parameters
- List the configuration parameter categories

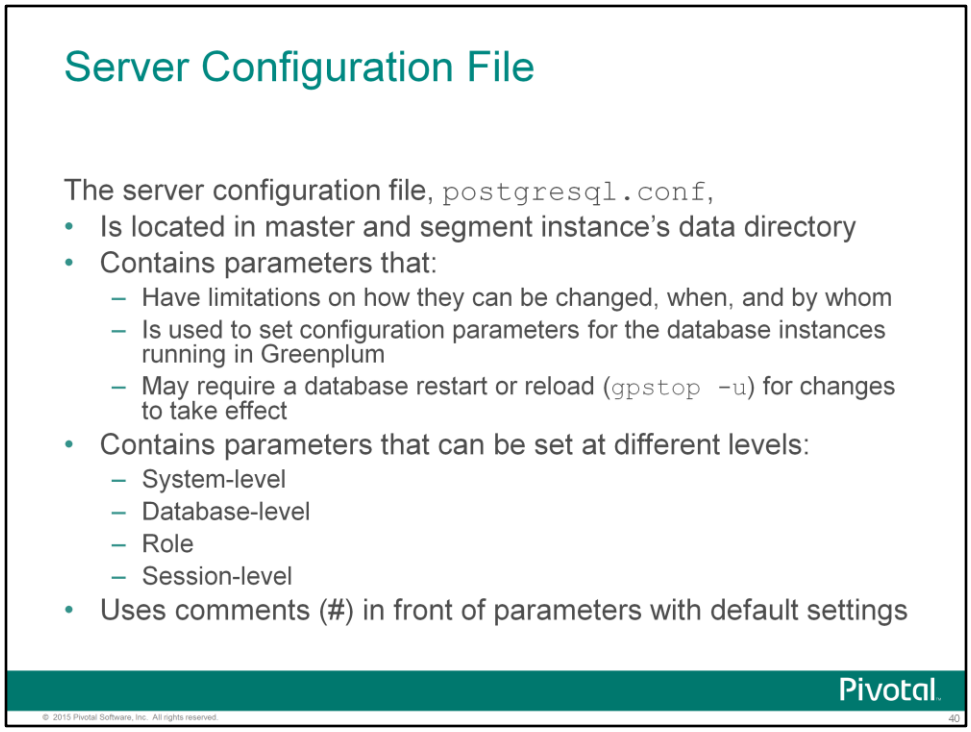


A server configuration file is used to configure various aspects of the DBMS. In Greenplum Database, as in PostgreSQL, this file is called `postgresql.conf`. This configuration file is located in the data directory of the database instance.

The master and each segment instance each have their own copy of the `postgresql.conf` file, located in their respective data directories.

Parameters can be:

- **Segment-level parameters** – Each segment instance evaluates the parameter in its own `postgresql.conf` file. You must set local parameters on each segment instance in the system.
- **Master parameters** – Master-only parameters are relevant only to master processes such as those used for query planning and client authentication. The value of a master is passed down to the segment at query run time. The value may be ignored by the segment.

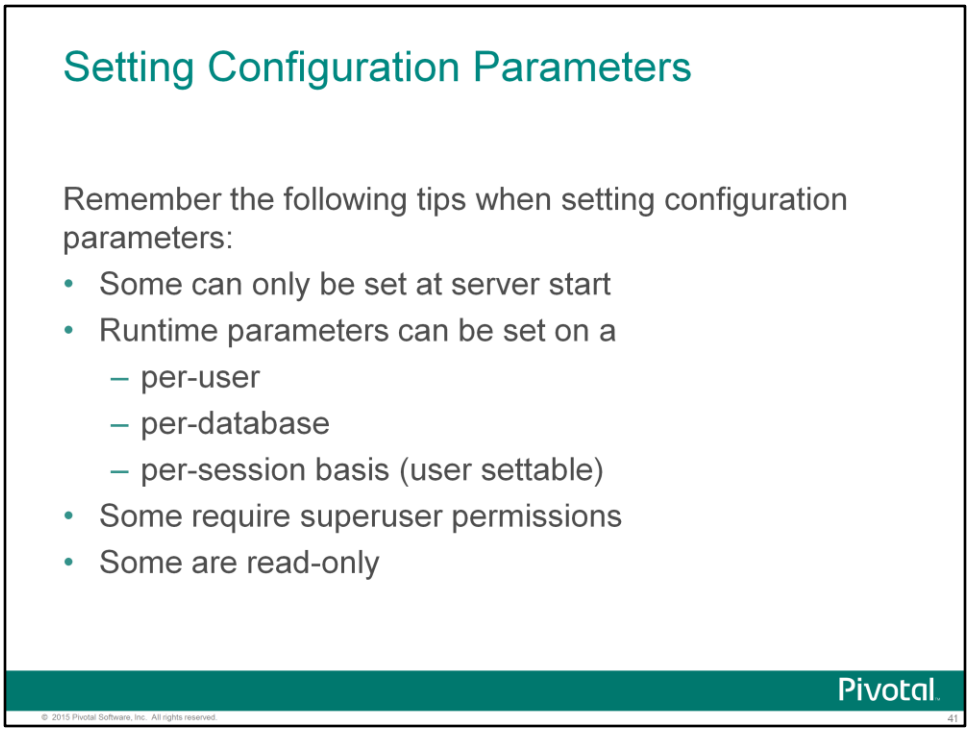


The `postgresql.conf` is the main server configuration file used to dictate the behavior of the DBMS. While the master instance and each segment instance have their own copy of `postgresql.conf`, you will most-likely only edit the server configuration file that resides on the master.

Configuration parameters may have the following restrictions:

- Some 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.
- Some parameters can only be set at the system-level in the `postgresql.conf` file.
- Other parameters may require a restart of the system for the changes to take effect.
- Many configuration parameters are considered *session* parameters. Most session parameters can be changed by any database user within their session, but a few may require superuser permissions. A session parameter can be set at the:
    - System-level
    - Database-level
    - Role-level
    - Session-level

Parameters with default settings are displayed as comments. If you wish to change the value of the parameter, first remove the comment character, #, and change the value of the parameter.



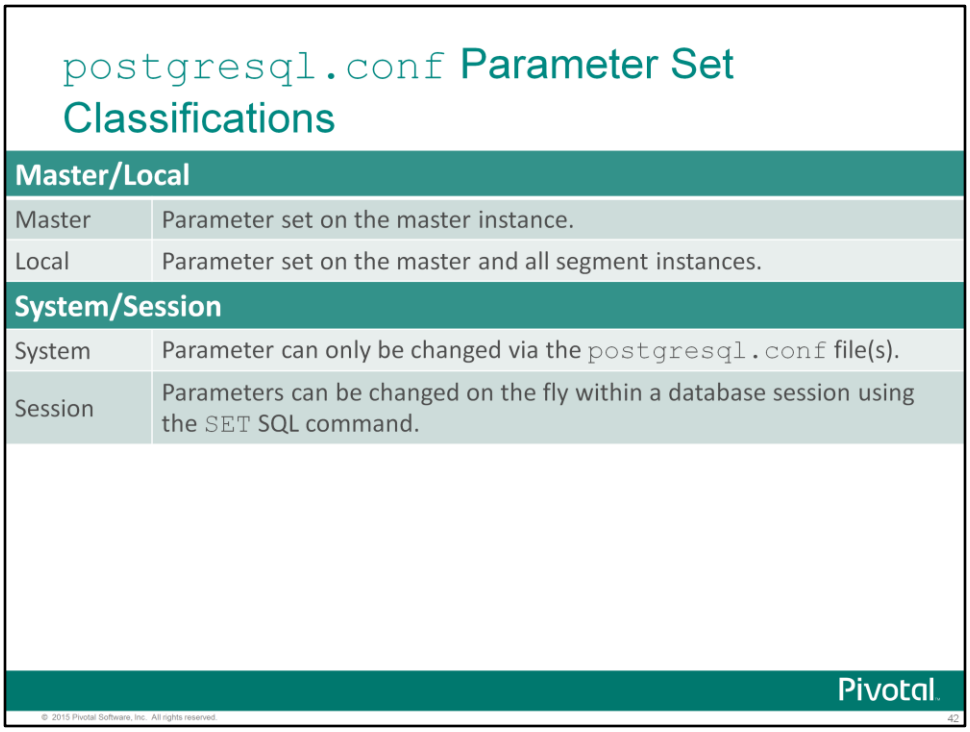
SMany of the configuration parameters can only be changed by using the `postgresql.conf` file and require a system restart for the configuration changes to take effect. For example, the `max_connections` parameter requires a database restart on changing its value.

Others are considered runtime parameters. These parameters can be set while the system is running and their changes will take effect on the next issued query or in some cases the next client session.

Some parameters can only be set by superusers (such as the `log_*` parameters).

Other parameters, such as `server_version`, are read-only. You can view these parameters to obtain information about the system but you cannot change their values.

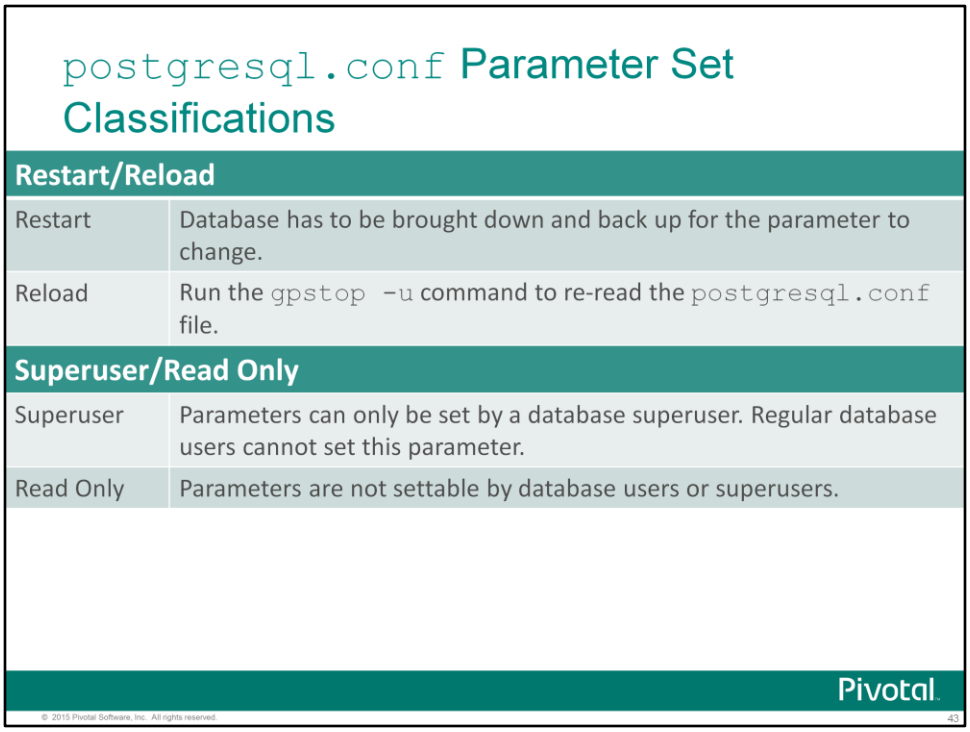
For a detailed list on all of the parameters and whether they are settable or not, reference the *Greenplum Database Administrator Guide.*



All parameters in the `postgresql.conf` files have set classification definitions that determine where and how the parameters are defined and read.

Parameters are defined as:

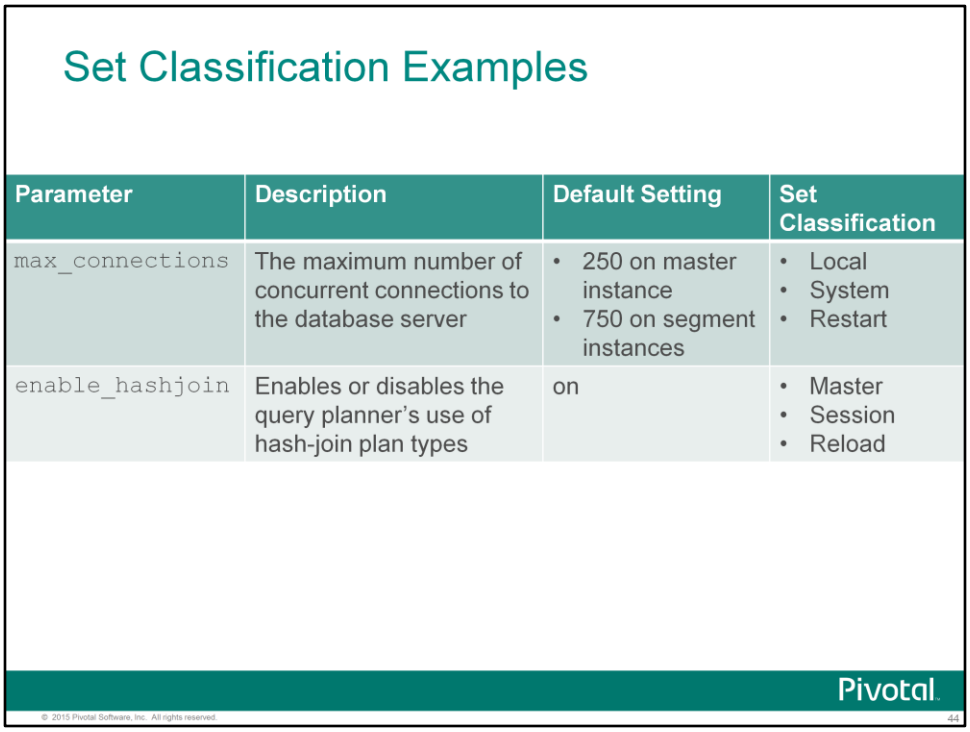
- Master – This parameter is set in the `postgresql.conf` file of the Greenplum master instance. The value for this parameter is then either passed to, or ignored by, the segments at run time.
- Local – This type of parameter must be set in the `postgresql.conf` file of the master and each segment instance. 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.
- Session – Session parameters can be changed within a database session while the database is actively running. If the parameter is set at multiple levels, the most granular setting takes precedence. The hierarchy from least granular to most granular are as follows:
    - System level by defining them in the `postgresql.conf` file
    - Database level by using the `ALTER DATABASE...SET` statement
    - Role level by using the `ALTER ROLE...SET` statement
    - Session level by using the `SET` statement.
- System – System parameters can only be changed within the `postgresql.conf` file, whether at the master or local level.



Some parameters require that the database is restarted when changed in the `postgresql.conf` file. Other parameter can be refreshed by just reloading the server configuration file, using the `gpstop -u` command. Reloading the server configuration file does not require stopping the system.

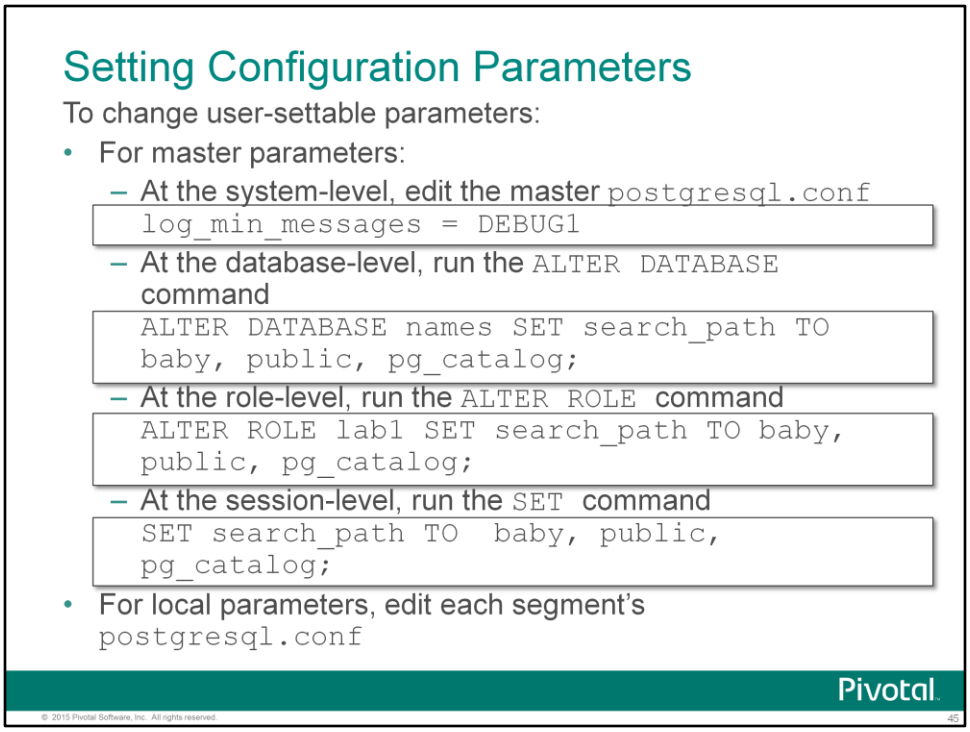
Superuser parameters are session parameters that can only be set by a database superuser. Regular database users cannot set this parameter.

A read only parameter is not settable by database users.



Here are two examples of how to read and understand the set classifications:

- The `max_connections` parameter is set both in the master and segment instances `postgresql.conf` file. This requirement defines a local parameter. A parameter that cannot be set while the database is running is called a system parameter. Parameters that require the database be stopped and restarted are restart parameters.
  Note: This parameter is set to different values on the master and on the segments. In this specific case, the segment instance parameter is set to 3 times the master instance.
- The `enable_hashjoin` parameter is only set on the master instance (Master). It can be set during an active connection (Session). If the parameter is set in the `postgresql.conf` file on the master instance, the configuration file must be re-read by the database for the parameter change to take effect. This requires a reload using the `gpstop -u` command.



User and superuser settable master parameters can be set at different levels – system level, database level, user level or session level. The more granular setting takes precedence, so for example setting a parameter at the session level will override a system-level default.

In the first example for master parameters, the log_min_messages is edited within the `postgresql.conf` file only on the master server. The `log_min_message` parameter controls the message levels of messages written to the server log.

The set classifications for the `log_min_messages` parameter are Master, Session, Reload, Superuser.

In the second example, the SQL command, `alter database` on the `names` database, updates the search path to a specific set of schemas. Anyone connecting to this database will now have this schema search path.

The `search_path` parameter specifies the order in which schemas are searched when an object is referenced by a simple name with no schema component.

The set classifications for the `search_path` parameter are Master, Session, Reload.

Setting Configuration Parameters (continued)

In the third example, the SQL command, `alter role` is applied to the lab1 role. Anyone connecting to a database as the lab1 role will have this schema search path.

In the fourth example, the SQL command, `set`, is applied to the user executing the command during their currently active connection. This alters the parameter during the session. When the active connection is ended, the parameter will revert back to its default setting.

Local parameters can only be set in the `postgresql.conf`. You must edit the `postgresql.conf` for each segment instance and the master instance to change a local setting system-wide. The use of the `gpssh` command is recommended, so that you can make the change to all `postgresql.conf` files at the same time. In most cases, changing local parameters is rarely necessary. All parameters that you would want to change are typically global or master-only parameters.

Managing Parameters with `gpconfig`

Option	Description	Example
`-c | --change <param_name>`	Updates the `postgresql.conf` file by adding a new setting at the bottom of the file	`$ gpconfig -c max_prepared_transactions -v 300`
`-v | --value <value>`	The value to use for the parameter you specified with the `-c` option.	
`-m | --mastervalue <value>`	The value to apply to the master and standby master `postgresql.conf`	`$ gpconfig -c max_connections -v 800 -m 300`
`--masteronly`	Apply the changes to the `postgresql.conf` file on the master only	`$ gpconfig -c max_connections -v 200 --masteronly`
`-r | --remove <param_name>`	Reset the parameter to the default value	`$ gpconfig -r max_connections`
`-l | --list`	List all supported configuration parameters	`$ gpconfig -l | grep max_connections`
`-s | --show <param_name>`	Display the parameter value on all instances as found in the database (and not in the `postgresql.conf` file)	`$ gpconfig -s max_connections`

Pivotal

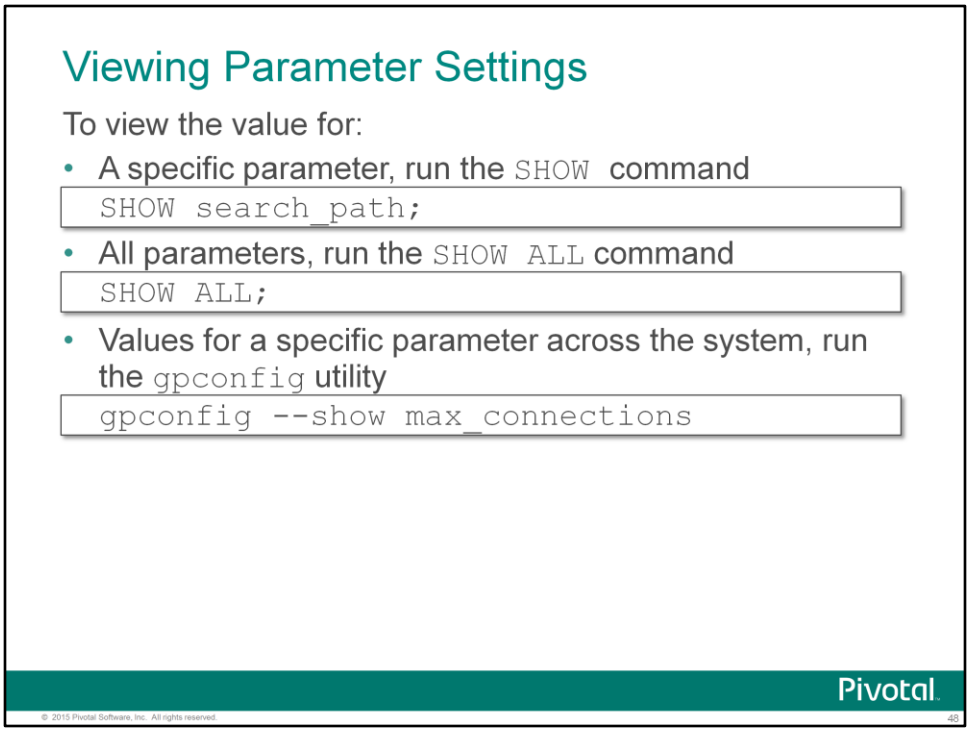
47

The `gpconfig` command is the preferred method of setting system level parameters. The command can also be used to unset or view configuration parameters as found in the postgresql.conf file across all segments, primary and mirror hosts, or master configuration parameters.

The gpconfig command supports a subset of the parameters found within the postgresql.conf file. You can however use it to view the values for all parameters found in the file. Use the `-l` option to obtain a list of supported settable parameters.

When setting a parameter value, the gpconfig command works by commenting out the parameter in the postgresql.conf file and adding an entry to the bottom of the file with the value you specify. When unsetting a value, it comments out the parameter so that the default value is restored.

Depending on the parameter, you must either reread the `postgresql.conf` file or restart the database. In the examples shown on this slide where the `max_connections` and `max_prepared_transactions` parameters are being modified, you will need to restart the database.



The SQL command, `SHOW`, allows you to view the current settings of configuration parameters used by the Greenplum Database system.

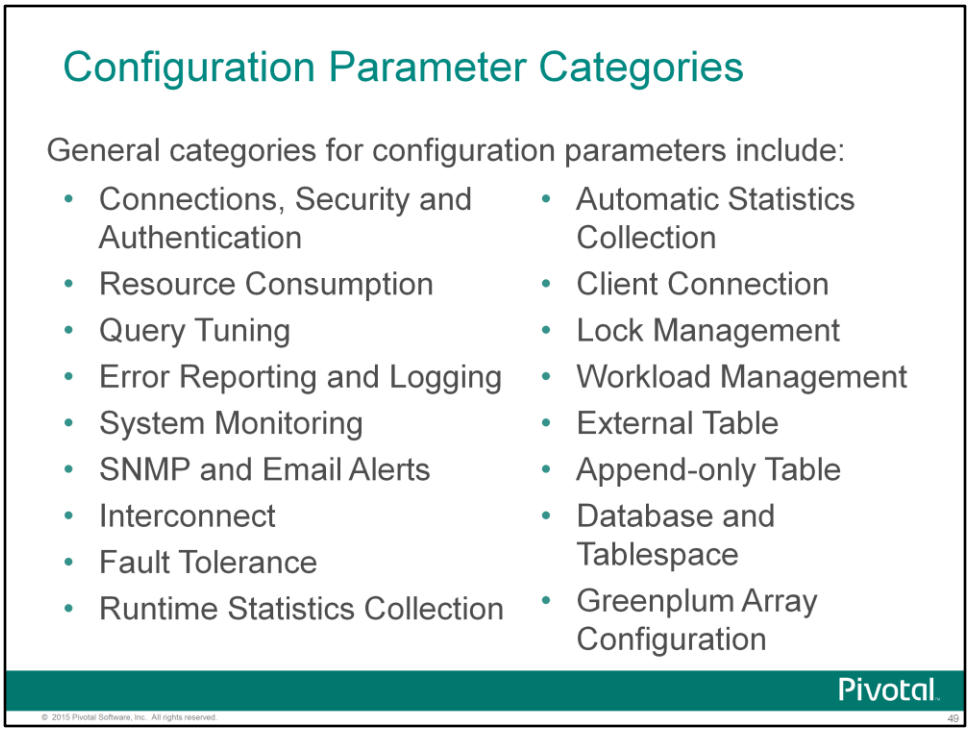
To see the settings for all parameters, execute the SQL command, `SHOW ALL`. For example, to run it non-interactively from PSQL, execute the following command:

```
$ psql -c 'SHOW ALL;'
```

Running SHOW will show the settings for the master instance only. To view the value of a specific parameter across the entire system, for the master and all segments, you can use the `gpconfig` utility. For example:

```
$ gpconfig --show max_connections
```

Note: The $ symbol preceding the command is the shell prompt and should not be executed as part of the command.



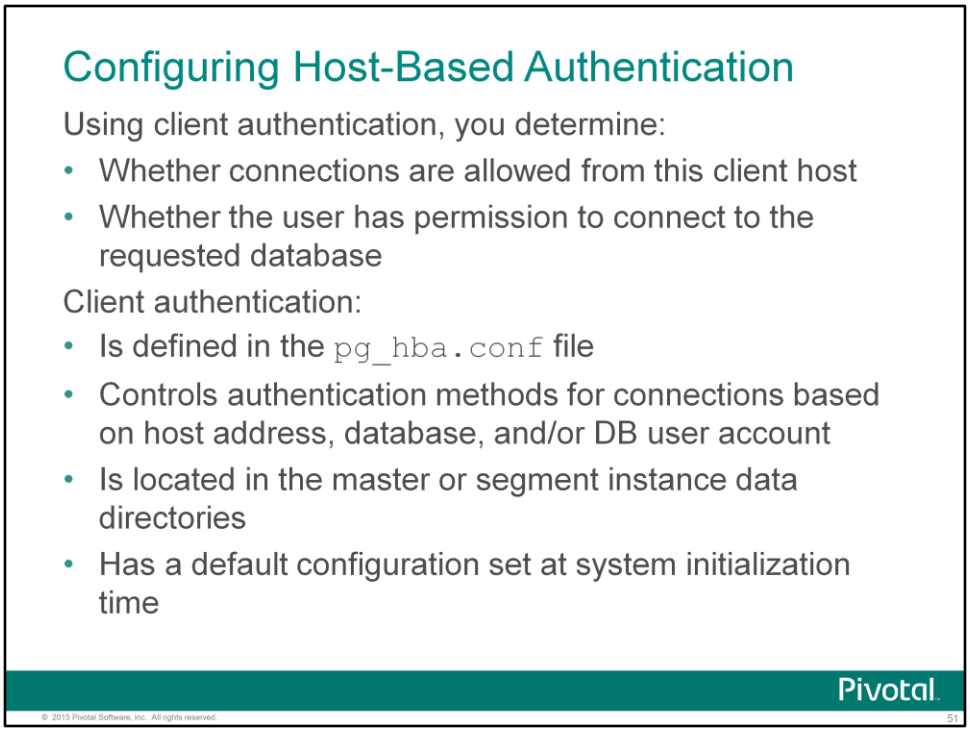
There are too many configuration parameters to cover each one in detail in this class, but here are a few general categories of parameters that you may want to explore:

- **Connections, Security and authentication** – These parameters control the connection, authentication and security such as the password encryption to use and how many concurrent connections are allowed.
- **Resource consumption** – These parameters control the allocation of system resources to database processes, such as memory, free space map, OS resources, and cost-based vacuum delay parameters.
- **Query tuning** – These configuration parameters provide a crude method of influencing the query plans chosen by the query optimizer. If the default plan chosen by the optimizer for a particular query is not optimal, a temporary solution may be found by using one of these configuration parameters to force the optimizer to choose a different plan. Because all query planning in GPDB is done by the master, these parameters are not relevant to segments.
- **Error reporting and logging** – These parameters control what, where and when to log.
- **System monitoring** – These parameters are used to define how alerts are sent to an administrator
- **SNMP and email alerts** – These parameters can be used to control how the SNMP traps work and how email is setup to alert administration staff.

Configuration Parameter Categories (Continued)

- **Interconnect –** These parameters control Interconnect layer behavior
- **Fault Tolerance** – These parameters controls the behavior of the ftsprobe process
- **Runtime statistics collection** – These parameters control the PostgreSQL server statistics collection feature used by the query planner.
- **Automatic statistics collection** – These parameters are used to automate the execution of ANALYZE commands.
- **Client connection** – These parameters are used to determine the client's default settings on connecting to the database.
- **Lock management** – These parameters handle deadlock and the maximum number of locks per transaction.
- **Workload management** – The parameters are used to configure the Greenplum Database workload management feature, query prioritization, memory utilization, and concurrency control.
- **External tables** – The parameters are used to configure the external tables feature for Greenplum.
- **Append-only** – These parameters are used to configure the append-only tables feature of Greenplum Database.
- **Database and tablespace** – The parameters configure the maximum number of tablespaces, databases, and filespaces.
- **Greenplum** – These are Greenplum-specific parameters that control things such as fault behavior and Interconnect defaults.

Note**:** For a complete listing of the parameters that fall into each category, refer to the *Greenplum Database Administrator Guide*.



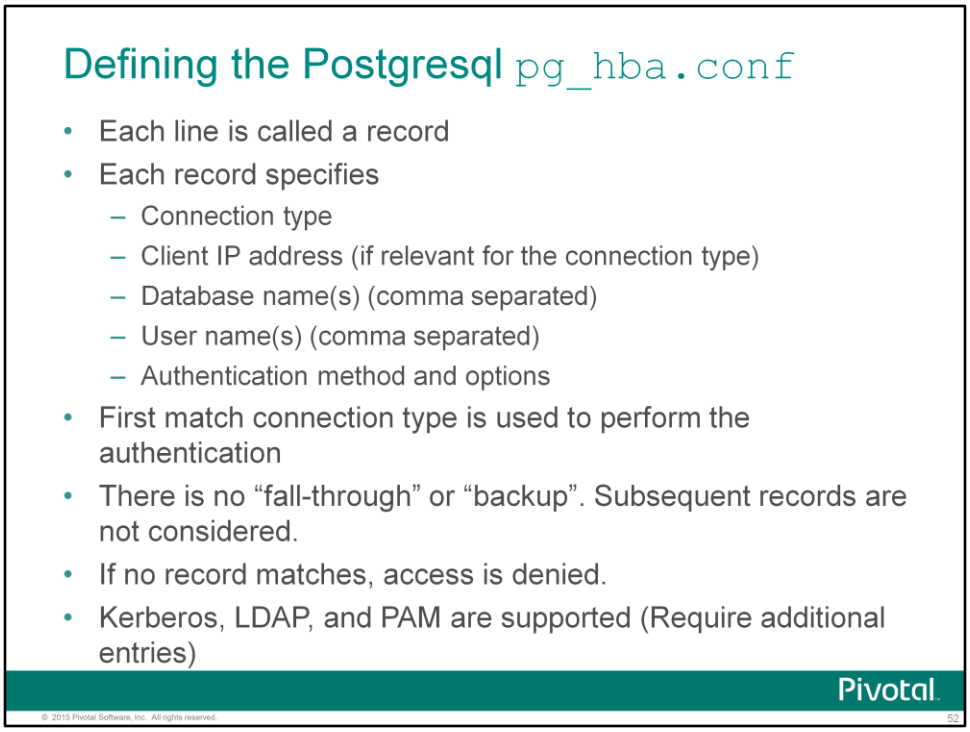
Client authentication is the process by which the database server:

- Establishes the identity of the client.
- Determines whether the client application, or the user who runs the client application, is permitted to connect to the requested database as the database user name given.

In Greenplum Database, as in PostgreSQL, client authentication is controlled with a configuration file called `pg_hba.conf`. This file has entries that tell the database server the method used to authenticate a particular client connection. These client connections can be selected on the basis of:

- Client host address
- Database
- User

A copy of this file is maintained on both the master and segment instance data directories with a default configuration that is set at system initialization.



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 the # 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 record specifies a connection type, a client IP address range (if relevant for the connection type), a database name, a user name, and the authentication method to be used for connections matching these parameters. The first record with a matching connection type, client address, requested database, and user name is used to perform authentication. There is no fall-through or backup. If one record is chosen and the authentication fails, subsequent records are not considered. If no record matches, access is denied.

Refer to the postgresql.org website for more information about each of the fields in the pg_hba.conf, as well required entries for Kerberos, LDAP, and PAM. The website is http://www.postgresql.org/docs/8.2/static/auth-pg-hba-conf.html.

pg_hba.conf Record Examples

TYPE	DATABASE	USER	CIDR-ADDRESS	METHOD
local	all	all		trust
host	all	all	127.0.0.1/32	trust
host	faa	all	192.168.142.0/24	ident md5
host	postgres	all	192.168.12.10/32	md5
host	postgres	all	192.168.93.0/24	md5
host	all	@admin,+support	192.168.0.0/16	md5
local	sameuser	all		trust
host	db1,db2,@listdb	all	192.168.93.0/24	md5

Pivotal

53

Example 1:

Allow any user on the local system to connect to any database under any database user name using

Unix-domain sockets (the default for local connections).

Example 2:

The same as above, but using local loopback TCP/IP connections.

Example 3:

Allow any user from any host with IP address 192.168.142.x to connect to database “faa“ as the same user name

that ident reports for the connection (typically the Unix user name), if the user’s password is correctly supplied.

Example 4:

Allow a user from host 192.168.12.10 to connect to database "postgres" if the user's password is correctly supplied.

`pg_hba.conf` Record Examples (continued)

Example 5:

Allow any user from any host with IP address 192.168.93.x to connect to database "postgres“, if the user’s password is correctly supplied.

Example 6:

Allow administrators and users that belong to the “support” role from 192.168.x.x hosts to connect to any database.

The file `$MASTER_DATA_DIRECTORY/admins` contains a list of names of administrators. Passwords are required in all cases.

Example 7:

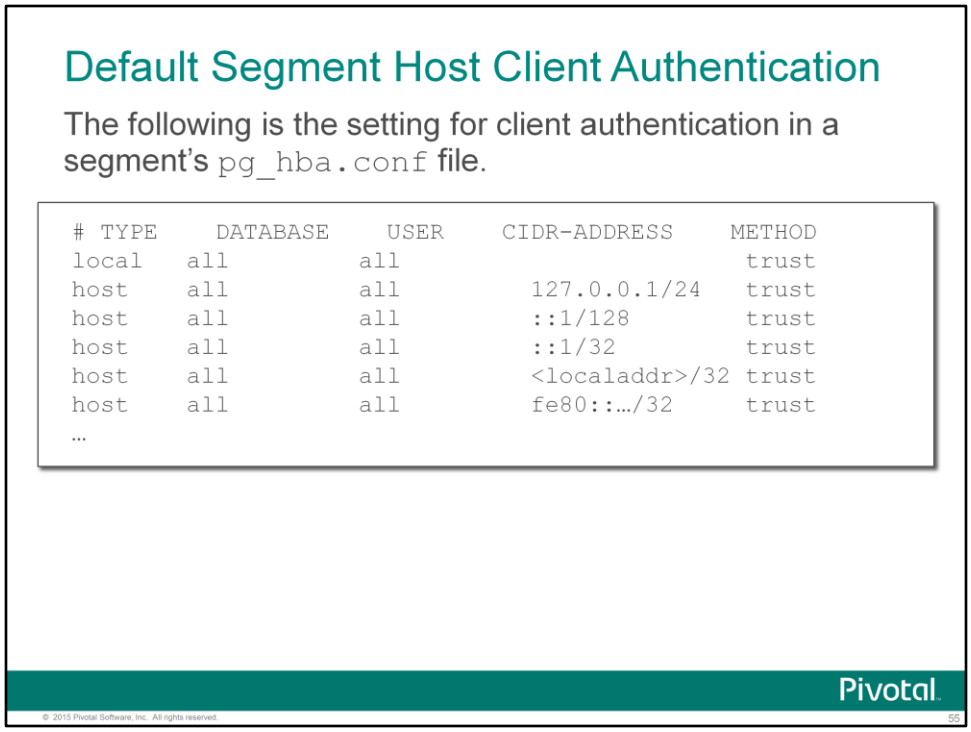
Allow local users to connect only to their own databases (databases with the same name as their database user name).

Passwords are NOT required.

Example 8:

Allow any user from any host with IP address 192.168.93.x to connect to databases “db1“, “db2”, and databases found in the `$MASTER_DATA_DIRECTORY/listdb` file.

Password are required in all cases.

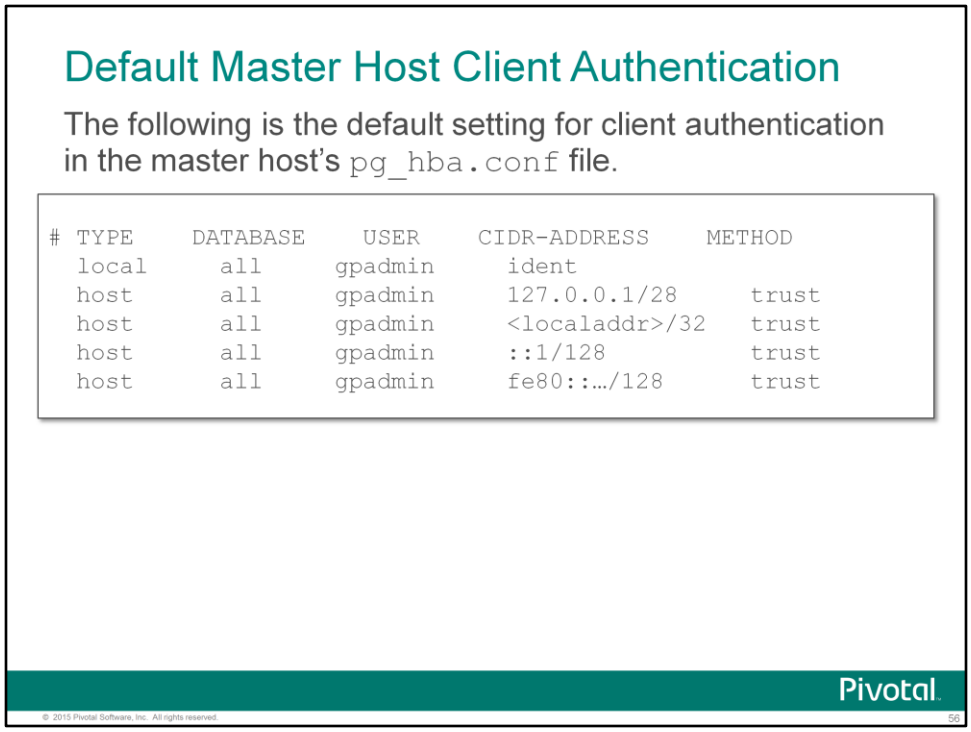


By default, the segment hosts are configured to allow remote client connections from the master host only. If you decide to deploy a warm standby master, you must edit the `pg_hba.conf` file of the segments to allow connections from your standby host as well.

Let us examine each line of the `pg_hba.conf` file for the segment hosts and explain what is happening:

- Line 1 – For local connections, all connections and users are allowed to connect to all databases. This method uses the trust authentication, where no authentication checks or challenges are performed.
- Lines 2 to 6 – Varying degrees of local IP addresses that are trusted. This includes connections from the master and standby servers as well as the localhost.

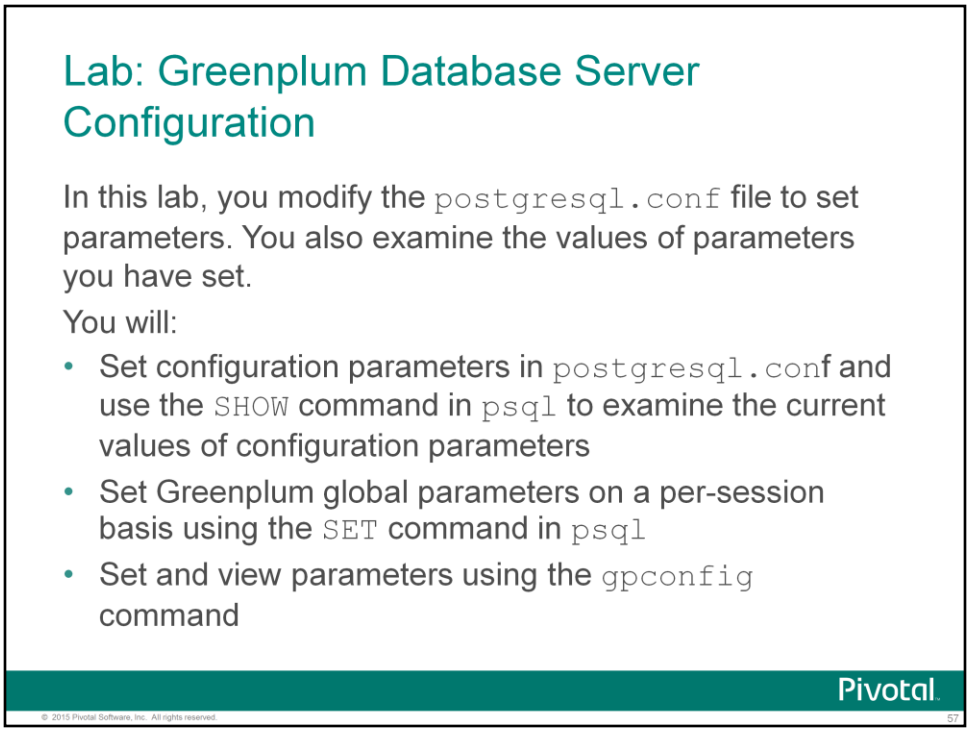
For TCP/IP connections, allow all users to connect to all databases when the connection comes from the local host. Note that the CIDR (classless inter-domain routing) address is in IPv4 or IPv6 format. Again, trust means no authentication is performed.



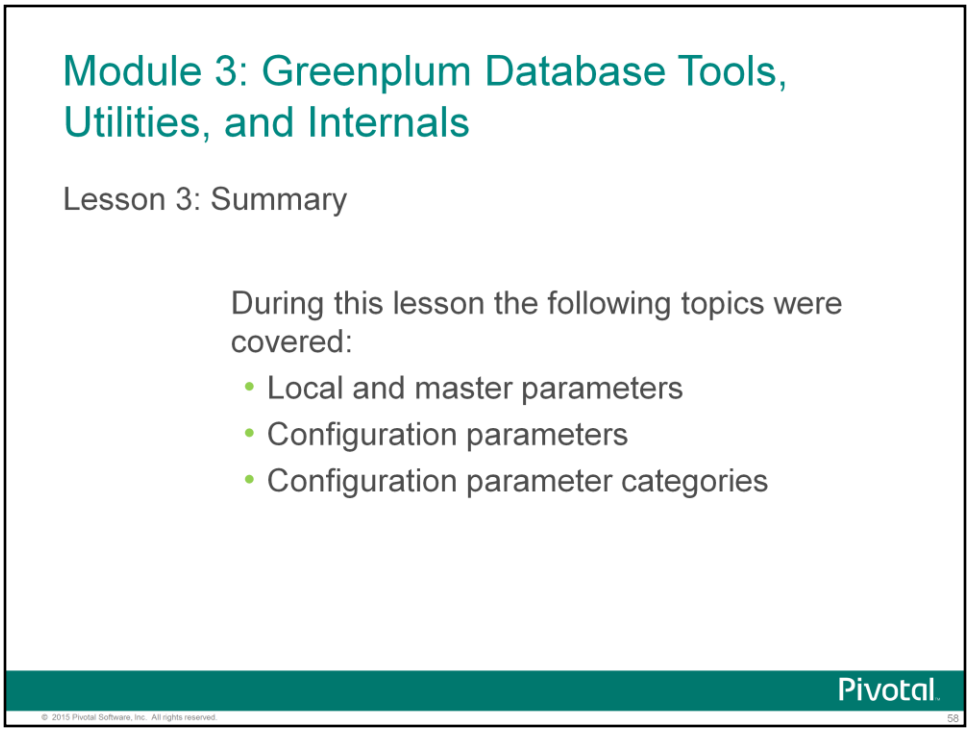
By default, the master host is configured to allow remote client connections from the default superuser account only. This is the user who initialized the Greenplum array. In this case, the OS user name and the Greenplum database user name are the same, but this may not always be the case for other users.

Let us examine each line of the `pg_hba.conf` file for the master host and explain what is happening:

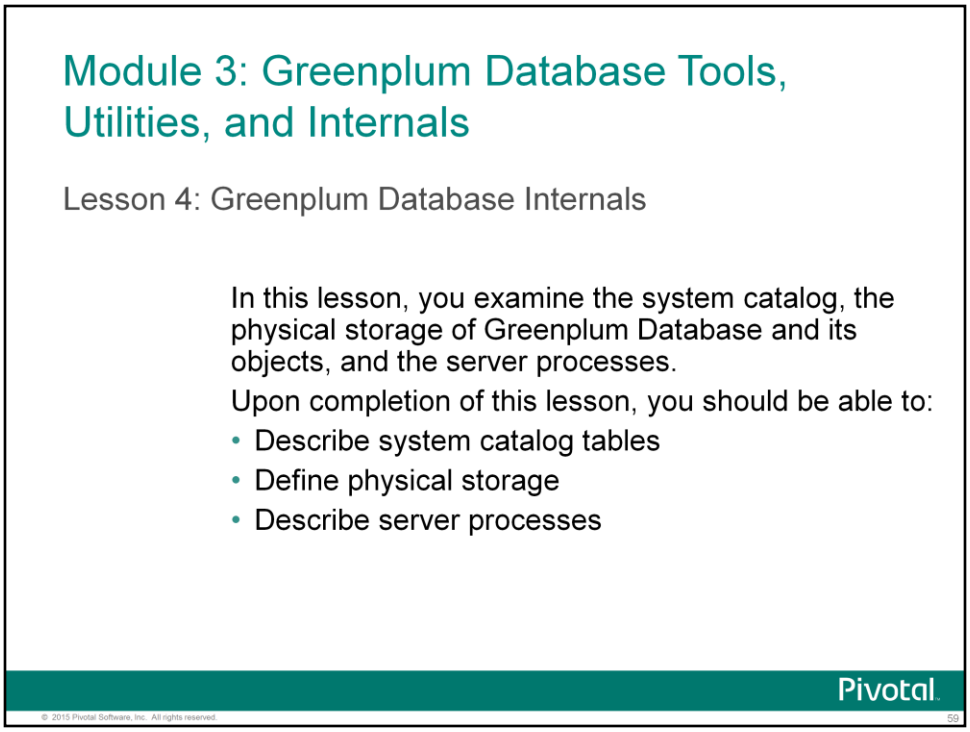
- **Line 1** – For local UNIX domain socket connections, allow the `gpadmin` user and all users to connect to all databases.
  The ident authentication method works by obtaining user name of the client and comparing it against the allowed database user names using a map file. A predefined ident map, `sameuser`, allows any operating system user to connect as the database user of the same name. Any other maps must be created manually using the `pg_ident.conf` file.
- **Line 2 and Line 4** – For TCP/IP connections, allow only the `gpadmin` superuser to connect to all databases when the connection comes from the local host. The CIDR address is in IPv4 format for line 2 and in IPv6 notation in line 4. The `trust` keyword means no authentication is performed.
- **Line 3 and Line 5** – For any remote host TCP/IP connections, allow only the `gpadmin` superuser to connect to all databases. The CIDR address is in IPv4 format for line 3 and in IPv6 notation in line 5. If the keyword, `md5`, is present, it means password authentication is performed.



In this lab, you modify the `postgresql.conf` file to set parameters. You also examine the values of parameters you have set.



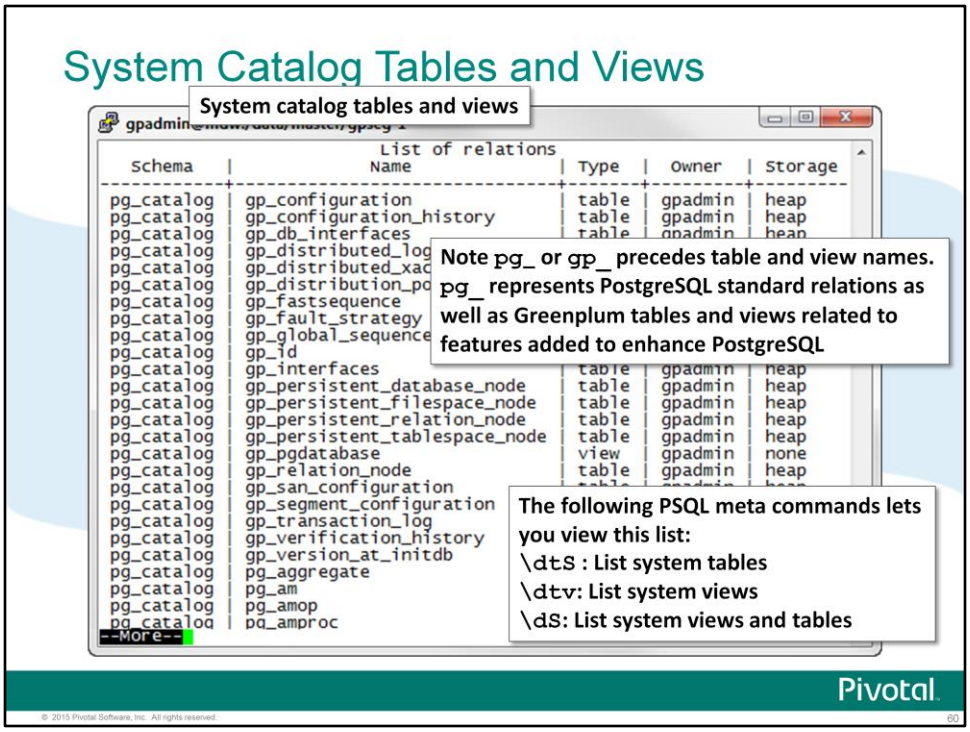
This lesson covered parameters that are set in the `postgresql.conf` file both at the local and master level of the cluster. A look at the classification levels and the categories as well as how to set the parameters rounds out the discussion. Next, we examined the `pg_hba.conf` file and how to define user authentication to one or more databases from the local system or remote systems.



Greenplum Database tracks various metadata information in its system catalogs about the objects stored in a database, such as tables, views, and indexes, and global objects such as roles and tablespaces. As an administrator, you may want to examine information about a table. You should also understand the file structure of the Greenplum Database as well as the processes that it executes.

In this lesson, you will:

- Identify the system catalog tables.
- Examine the physical file storage of the Greenplum Database.
- Identify the server processes for the Greenplum Database.

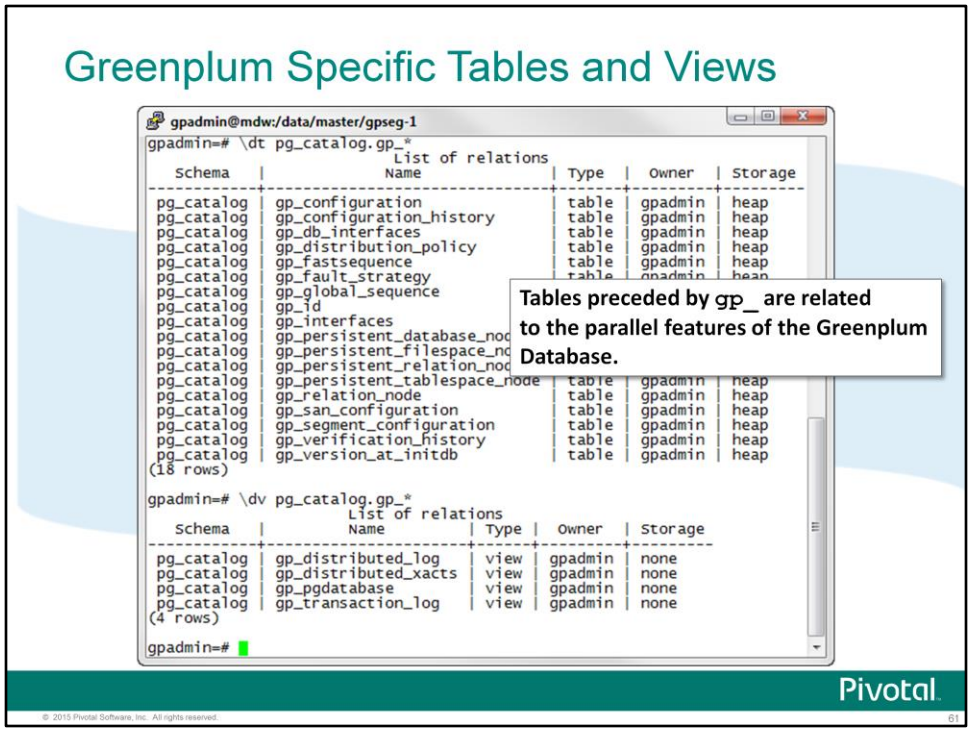


The system catalog is a set of system tables that contain metadata about the Greenplum Database. The system catalog is stored on the master and replicated to the warm standby.

All system catalog tables are stored in the `pg_catalog` schema. A schema is a named logical method of organizing objects and data in a database. Objects are referred to by the schema name, followed by a period, and the object name. This allows you to have objects with the same name defined in the database without creating conflicts, as long as they are in different schemas.

Greenplum uses the standard PostgreSQL system catalog tables, such as `pg_database`, `pg_inherits`, `pg_index`, and `pg_class`.

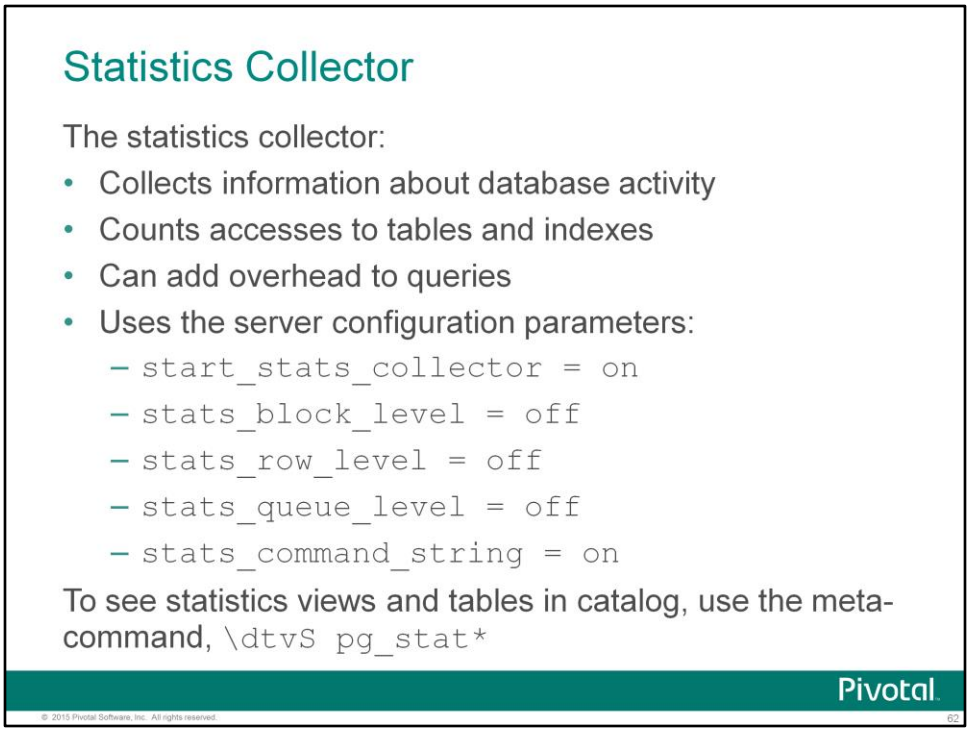
A list of the system catalog tables and views can be obtained from PSQL with the commands `\dtS` and `\dvS` respectively.



In addition to the PostgreSQL system catalog tables, Greenplum defines additional tables used to store information about the array. These are all global tables, shared across all databases on an instance. Some of the tables in the system catalog are:

- `gp_segment_configuration` – This contains the GPDB array master host and segment instances. It provides information on the status, mode, hostname, address, replication port, and whether they are in SAN instead of on the local environment.
- `gp_distribution_policy` – GPDB distribution key columns for a table
- `gp_id` – Each segment has a local copy of this table with its own `dbid` and `contentid`
- `gp_version_at_initdb` – This table is populated on the master and each segment in the Greenplum Database system. It identifies the version of Greenplum Database used when the system was first initialized.
- `pg_resqueue` – Populated only on the master, this table contains information about Greenplum Database resource queues, which are used for the workload management feature.
- pg_exttable – This table is used to track external tables and web tables created by the `CREATE EXTERNAL TABLE` command.

For a complete list of all tables in the system catalog, reference the *Greenplum Database Administration Guide*.

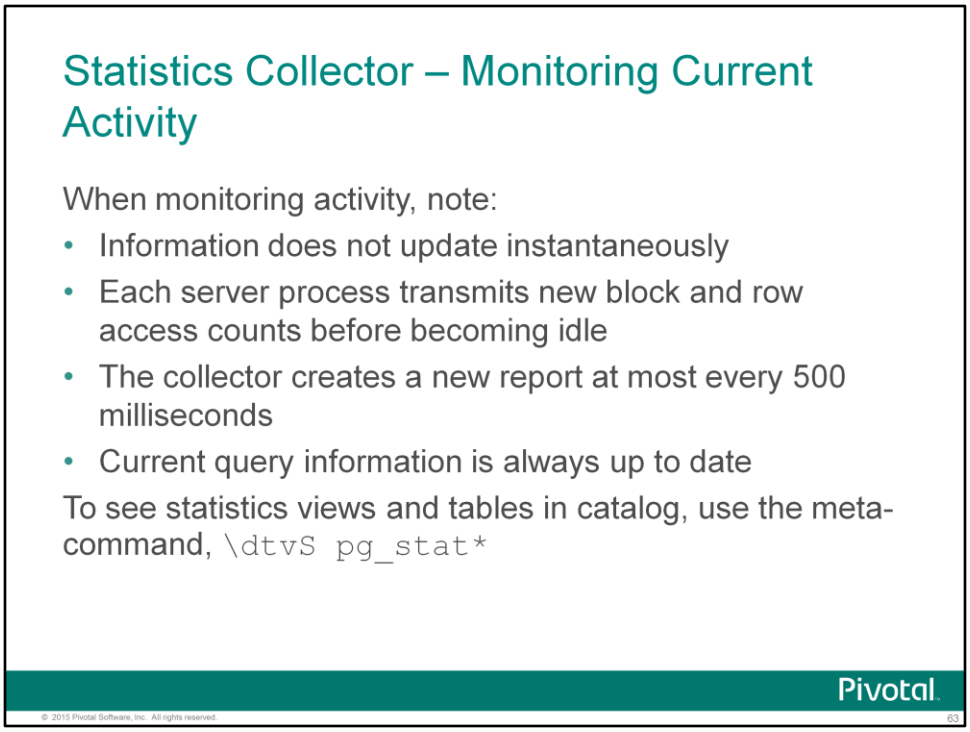


Greenplum makes use of the statistics collector in PostgreSQL. The statistics collector

- Is a subsystem that supports collection and reporting of information about DBMS activity.
- Can count accesses to tables and indexes in both disk-block and individual-row terms.

Statistics collection does add some overhead to queries, so you must enable it on the master in the `postgresql.conf` file with the following parameters:

- The parameters `stats_block_level`, `stats_row_level`, and `stats_queue_level` control how much information is sent to the collector and thus determine how much run-time overhead occurs. These determine whether a server process tracks disk-block-level access statistics and row-level access statistics or queue-level access statistics and sends this information to the collector. Additionally, per-database transaction commit and abort statistics are collected if either of these parameters are set. Since these are set to FALSE by default, very few stats are collected in the default configuration.
  You can enable temporarily to do analysis of activity and then disable to prevent unnecessary overhead.
- The parameter `stats_command_string` enables monitoring of the current command being executed by any server process.

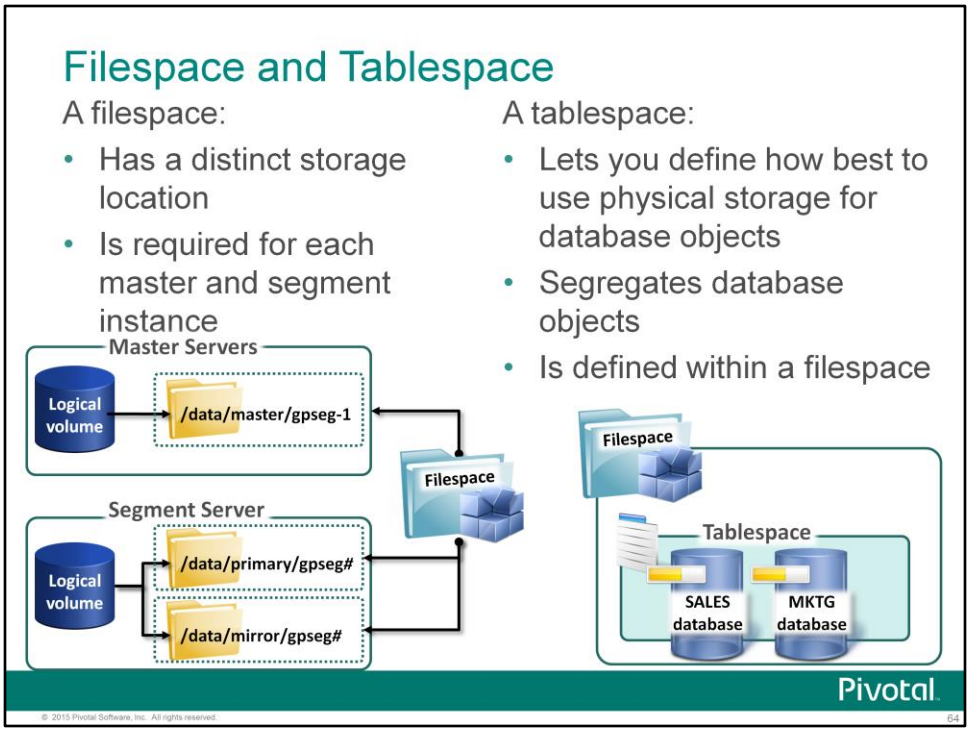


Several tables and predefined views are available to show the results of statistics collection.

When using the statistics to monitor current activity, it is important to realize:

- The information does not update instantaneously.
- Each individual server process transmits new block and row access counts to the collector just before going idle; so a query or transaction still in progress does not affect the displayed totals.
- The collector emits a new report at most once per 500 milliseconds. The displayed information therefore lags behind actual activity.
- Current-query information collected by `stats_command_string` is always up-to-date.

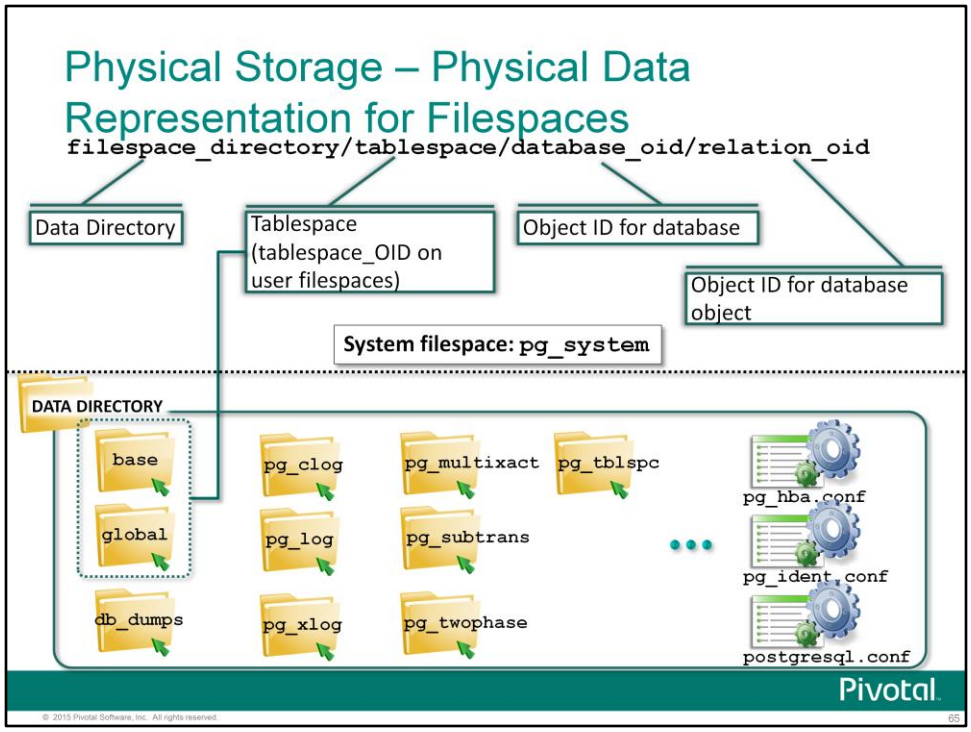
From psql, run the meta-command, `\dtvS pg_stat*`, to see statistics views and tables stored in the system catalog.



When working with the file system, database objects can be segregated by filespaces and tablespaces.

A filespace is a distinct storage location on disk. In Greenplum, each master and segment instance requires its own storage location. For example, the directory represented by `$MASTER_DATA_DIRECTORY` on the master and standby master servers is a filespace. The corresponding filespace on the segment instances are `/data/primary` and `/data/mirror`.

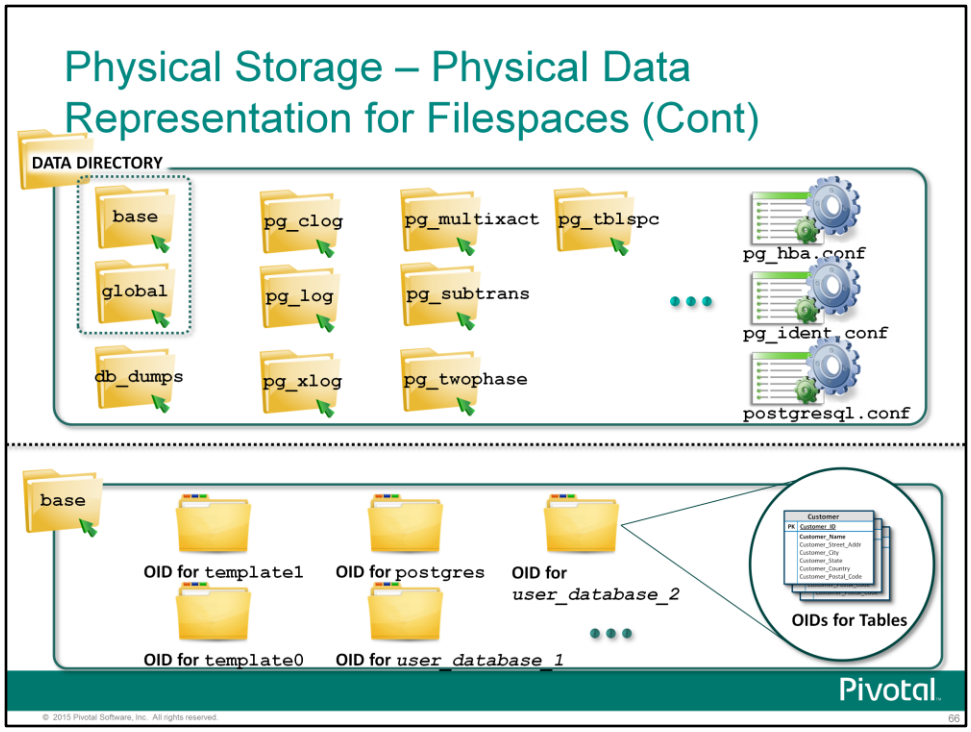
A tablespace lets you control the layout of the physical data on the filesystem. For example, you can define different storage types for different tablespaces, taking advantage of faster disk drives for data which is used more often, and standard disk drives for less frequently accessed data. A tablespace requires a filesystem location to store the database files. The filesystem is represented by the filespace.



Data is represented on the file system with file structures that use object identifiers, OID. Each OID is a unique identifier that describes that object. Data is stored in the file structure, *filespace_directory/tablespace/database_oid/relation_oid*.

There are two main types of filespaces: system and user. The system filespace, `pg_system`, is created automatically when you initialize the Greenplum Database using `gpinitsystem`. No user filespaces exist by default. Therefore, all objects you create initially exist in the system filespace, `pg_system`.

The diagram at the bottom of the slide illustrates some of the directories found in the system filespace. Configuration files, logs, tablespace directories, and database dump directories and files are found in the data directory for the master and segment instances. There are two tablespaces created by default during initialization: `pg_default` and `pg_global`. These two tablespaces are represented not by their OID, but by the directory names, `base` and `global`. This is the exception to the rule. User tablespaces are identified on the file system by their tablespace OID.

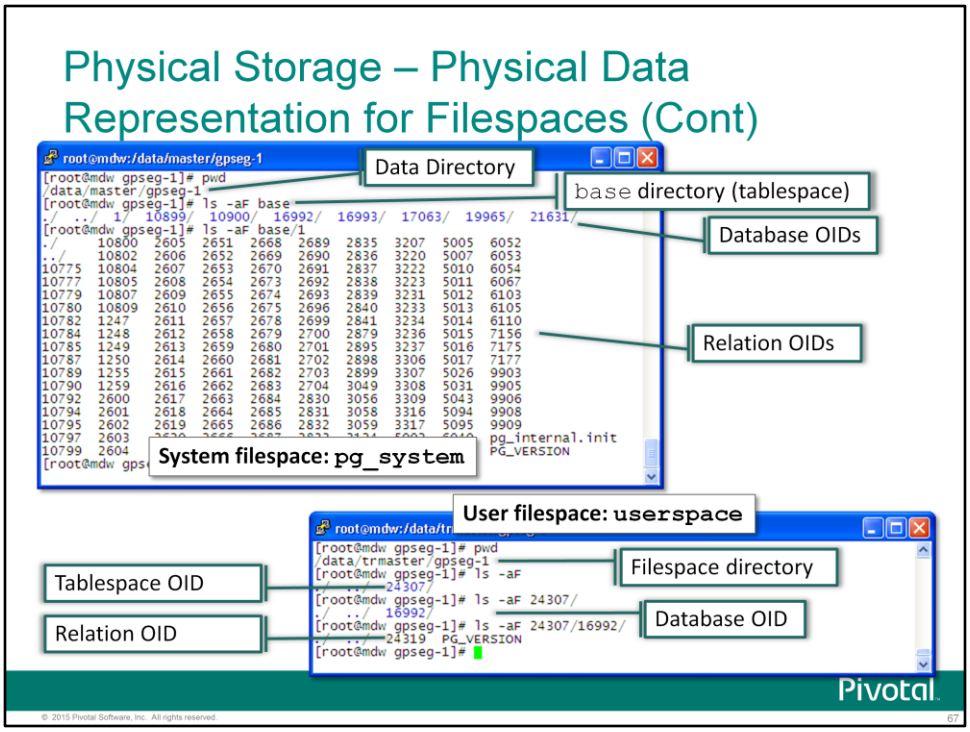


Under each data directory for the system filespace, the file structure for the Greenplum Database is as follows:

- **`base`** – This directory contains subdirectories for each database, identified by their object identifier (OID) in `pg_database`. Each sub-directory contains a list of the tables, stored as files. These files are identified by their OID as defined in `pg_class`.
- **`global`** – The global directory, contains system-wide tables, such as `pg_database` and `gp_configuration`. The database objects are stored directly in the global directory, instead of in subdirectory representing a database OID.
- **`pg_clog`** – This directory contains transaction commit status data.
- **`pg_multixact`** - This directory stores multitransaction status data, used for shared row locks.
- **`pg_subtrans`** – This directory stores subtransaction status data.
- **`pg_tblspc`** – This directory contains symbolic links to tablespaces.
- **`pg_twophase`** – The state files for prepared transactions are stored here.
- **`pg_xlog`** – Write Ahead Log (WAL) files are stored in this directory.

Files stored in this structure include:

- **`postmaster.opts`** – A file recording the command-line options the postmaster was last started with
- **`postmaster.pid`** – A lock file recording the current postmaster process ID and shared memory segment ID (not present after postmaster shutdown)
- Configuration files, including `postgresql.conf`, `pg_ident.conf`, `pg_hba.conf`.

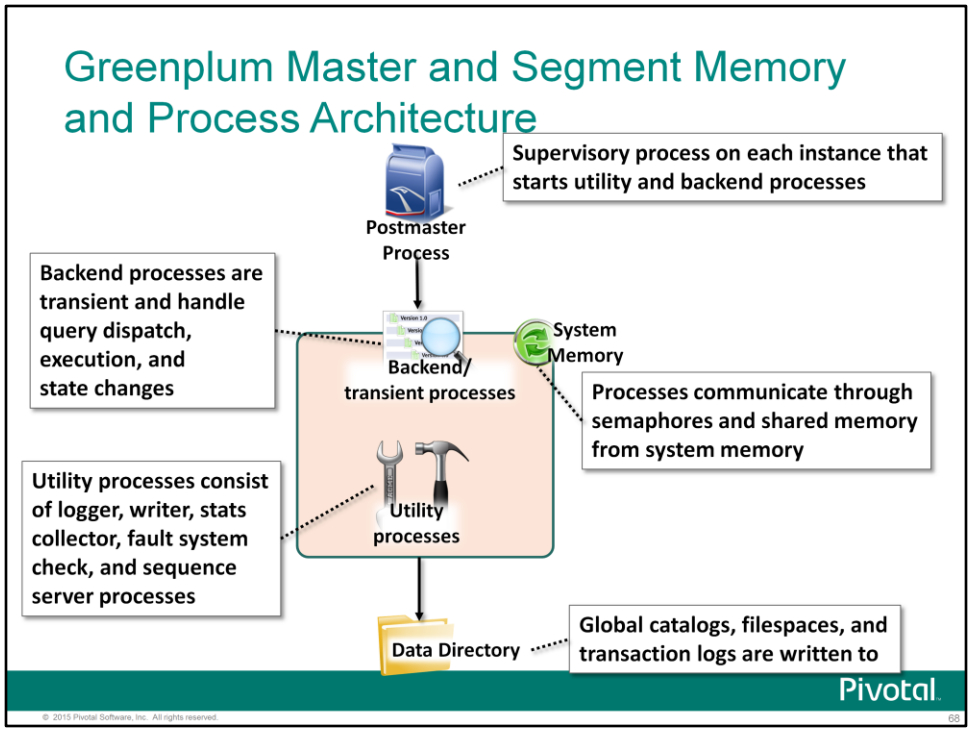


This slide shows how the file structure is represented on a populated system. The first image shows the base directory located inside the data directory on the master and segment instances. Within the base directory are the database OIDs representing different database names that exist on the system, such as `template0`, `template1`, and `postgres`. Next, the relation OIDs are displayed for one of the database directories. These relation OIDs are tables created within the database.

The second example shows a user filespace, called userspace, created on the same system. The directory for that filesystem is `/data/trmaster/gpseg-1`. Within that filesystem directory are sub-directories for any tablespaces created within that filespace. In this example, there is only one tablespace represented by the OID, 24307. Within that tablespace is a database, represented by the OID, 16992. Finally, only one table exists within that database. It is represented by the relation OID, 24319.

Note that the user filespace does not contain any logging or other subdirectory shown for the system filespace.

Now that we have examined the file structure, let us take a look at the processes.



Each Greenplum master and segment instance consists of the following components:

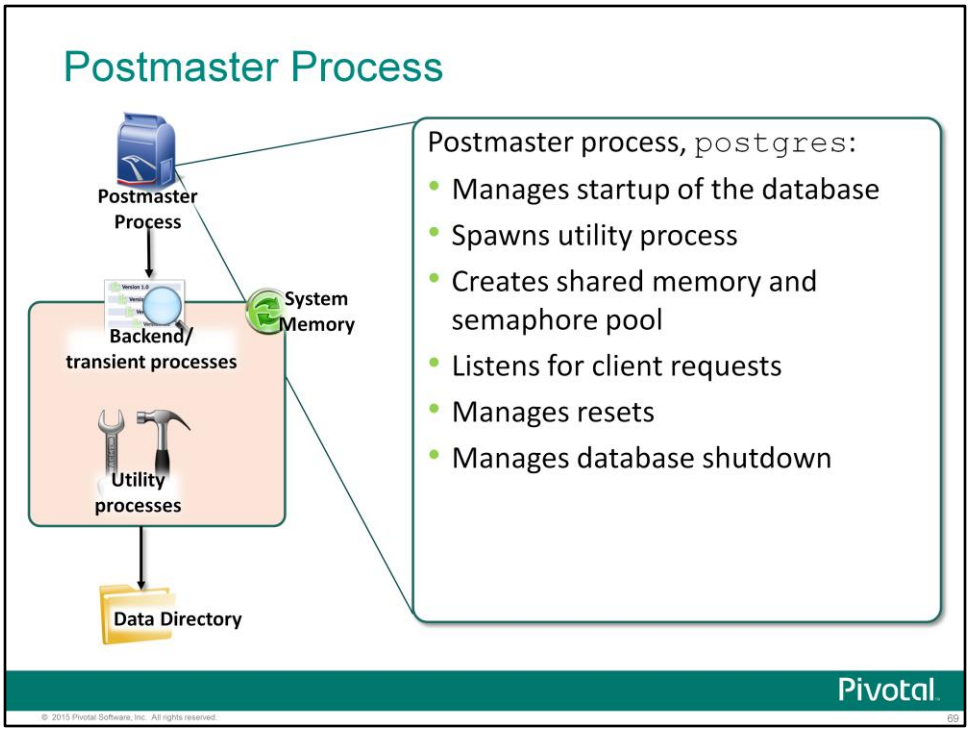
- Processes, which includes the postmaster, backend, and utility processes
- System memory
- File system

Each primary and segment instance has a single supervisory process called the postmaster, which is responsible for spawning and managing processes associated with starting and stopping the database, starting and managing backend and utility processes, and creating shared memory.

Backend processes are processes that perform query dispatch and execution. These processes are responsible for any work directly related to queries. In addition, the `filerep` process is a backend process responsible for managing mirrors.

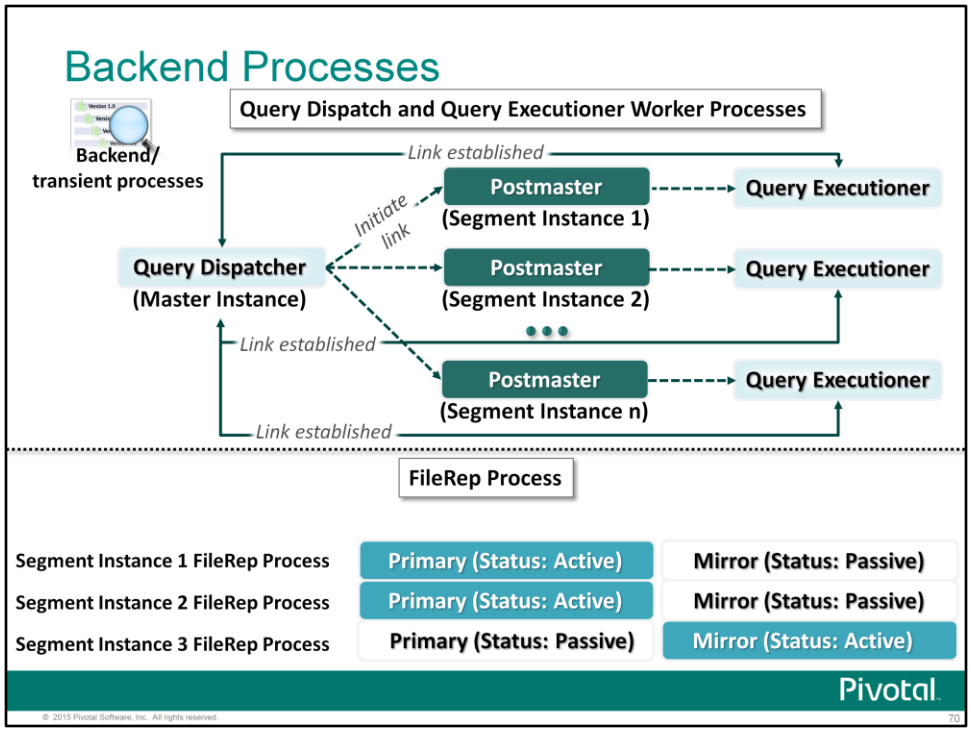
Utility processes are all of the other processes used to manage the environment, such as logger processes that write diagnostic messages to log files, statistics collector process, writer process, fault tolerant system checks process, and sequential server process.

These processes all communicate through shared memory created by the postmaster.



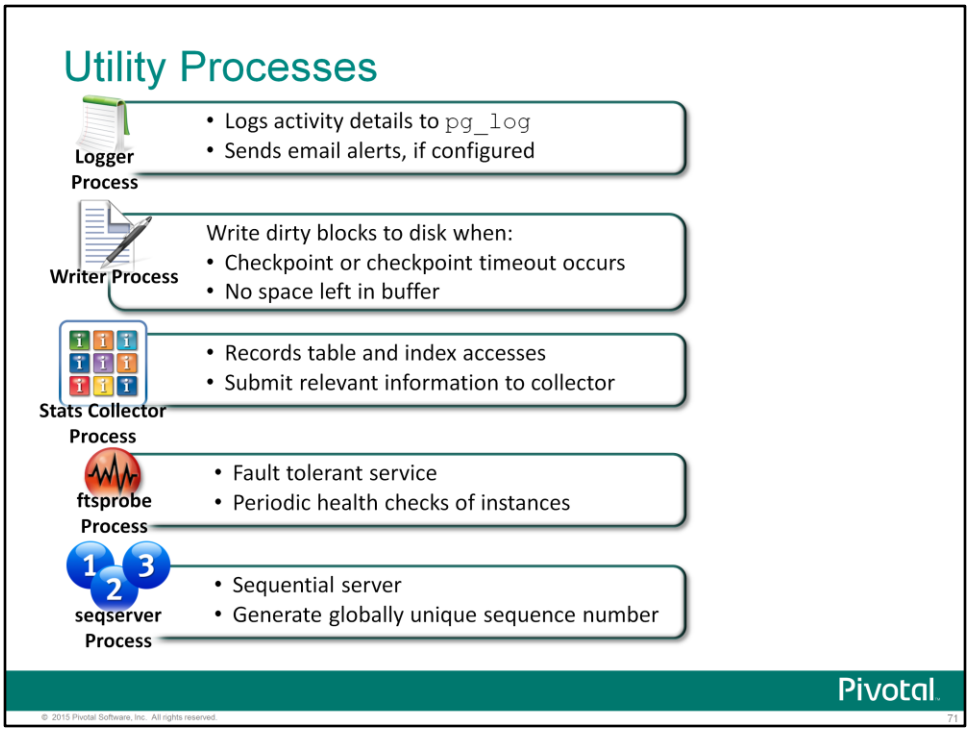
The postmaster process, postgres, is responsible for:

- Managing the startup of all database processes on all instances
- Spawning utility processes
- Creating shared memory and semaphore pool used by processes for both communication and to complete outstanding actions
- Listening for any client requests and connections made to the database, creating a backend process for that connection, and linking the client and backend process so that the postmaster process is no longer in the loop
- Managing resets across segment instances that have experienced a panic situation, due to an extremely inconsistent state
- Managing the parallel shutdown of the database across all instances



Backend processes are processes responsible for working with the data. These processes include:

- Query dispatcher (QD) worker process runs on the master instance and initiates communication with segments when a client request results in a query being issued to the database. This process opens a connection to the postmaster on the segments, which then creates separate query executor (QE) worker processes and links those executor processes to the dispatcher process. Once the link has been established, the query is dispatched to the query executioner worker processes on the segment instances.
  The query dispatcher is responsible for maintaining the gang of processes associated with a task. A gang is defined as a group of one or more QE processes working to complete a task. This gang has identical instructions, so that when the query executor process completes, it returns the result set on that segment instance for that specific task.
- Query executor worker processes run on the active segment after receiving a plan of action from the optimizer. As part of a gang, the QE process extracts data from the data tables on the segment instance it is executing on and completes the assigned task, returning the output to the query dispatcher.
- The FileRep processes execute on every segment instance and is used to manage the segment instance role and state as requested by the script, `gp_primarymirror`. The script is called during database startup, failover, resynchronization, adding mirrors, and database shutdown.



The utility processes responsible for some of the database and operating system management tasks include the following list of processes:

- The `logger process` executes on the master and segment instances, logging details of activity that have occurred on the system to files in the `pg_log` directory. Activities include but are not limited to startup and shutdown of the database, synchronizations, resynchronizations, and transaction logging.
- The `writer process` writes dirty blocks from the buffer pool shared memory to the file system when a checkpoint or checkpoint timeout occurs, or when there is no space left in the buffer pool memory. It works in conjunction with the `checkpoint process`.
- The `stats collector` process records the accesses on tables and indexes as well as user-defined functions. By periodically submitting the information to the stats collector, the information is kept up to date.
- The `ftsprobe` service sends a periodic check to segment instances to coordinate failover between primary and mirror segments. Should mirroring not be configured within the environment, a failed segment will result in the database being stopped.
- The `seqserver` process is a master instance process that generates globally unique sequence numbers that is requested by a worker process.

Listing Server Processes

To obtain a listing of PostgreSQL processes on Linux, run the following command:

```
ps ax | grep postgres
```

The `postgres` database listener process:

- On the Greenplum master instance includes:

  postgress postmaster process

  `postgres: <port port#>, <sub_process_name>`

  `postgres: <port port#> <user> <database> <con#> <host> <cmd#><slice#>`

- On the Greenplum segment instance includes:

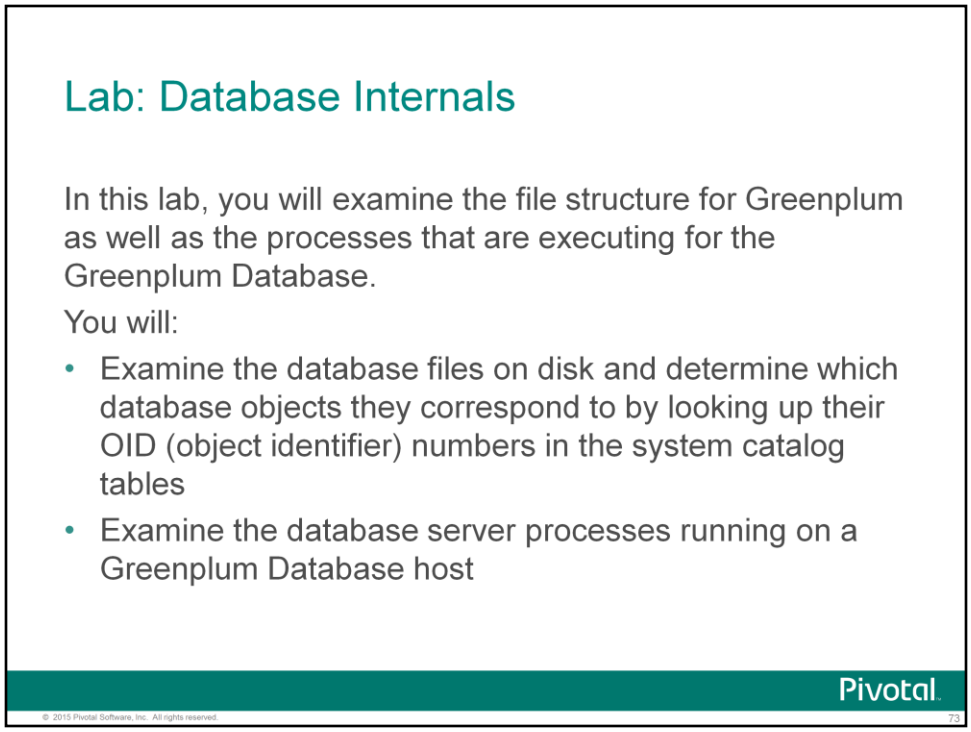
  postgres postmaster process (primary)

  postgres postmaster process (mirror)

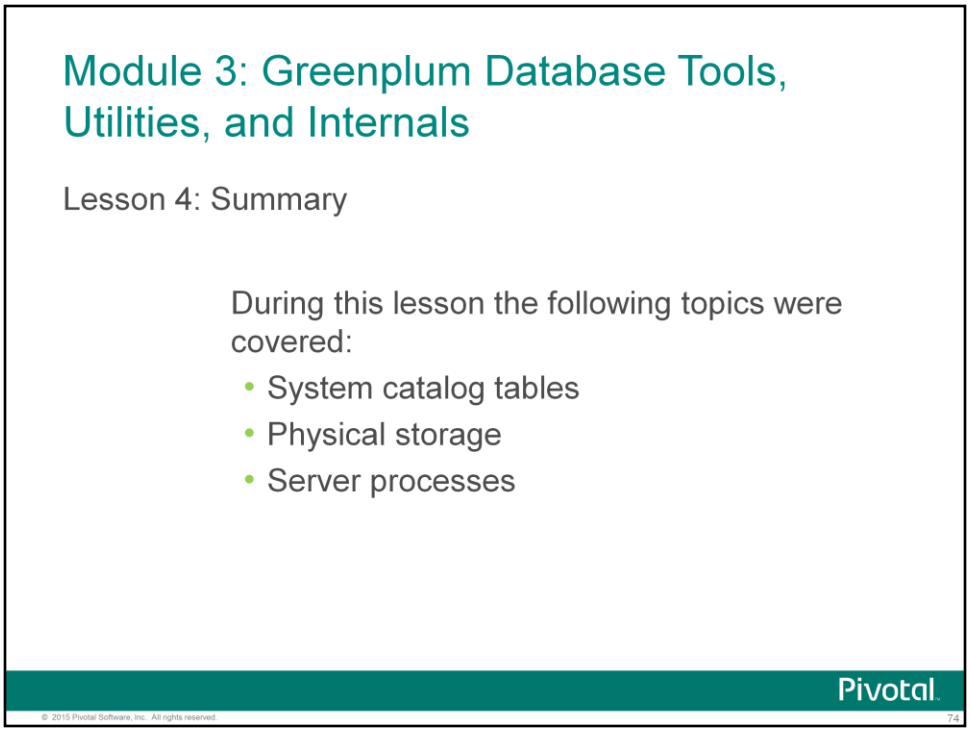
  `process postgres: <sub_process_name>`

Pivotal

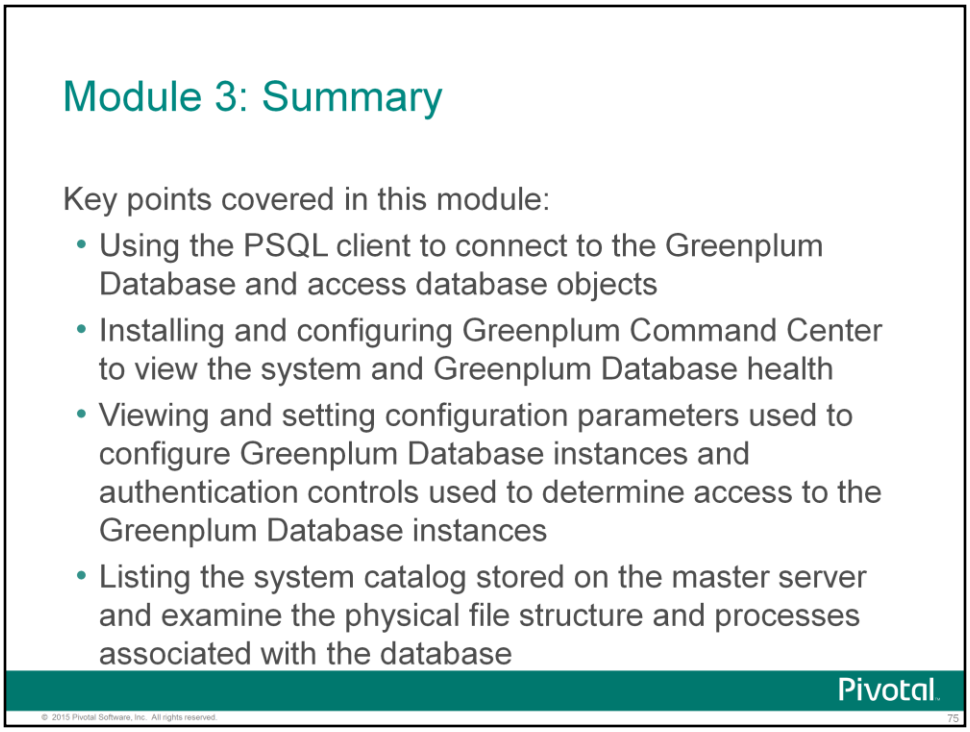
Execute the `ps` command to obtain the listing of Greenplum postmaster and child processes. The postmaster process will be the original postgres listener command with startup options. You cannot see `postgres` subprocess detail, but you do get information on what the subprocess is and the database port to which the process is attached.



In this lab, you will examine the file structure for Greenplum as well as the processes that are executing for the Greenplum Database.



This lesson covered how to access system catalog tables and views and provided an overview of the more commonly used ones. The lesson also highlighted how database objects are stored to the physical storage, with an emphasis on filespaces and tablespaces used by the database. An overview of the server processes that are found on the master, standby, and segment hosts was also provided in the lesson.



Listed are the key points covered in this module. You should have learned to:

- Use the PSQL client to connect to the Greenplum database and access database objects
- Install and configure Greenplum Command Center to view the system and Greenplum Database health
- Set configuration parameters used to configure Greenplum database instances and authentication controls used to determine access to the Greenplum database instances
- List the system catalog stored on the master server and examine the physical file structure and processes associated with the database

This slide is intentionally left blank.

Pivotal