Pivotal

PRODUCT DOCUMENTATION

Pivotal™ Greenplum Database®

Version 4.3

Getting Started

Rev: A01

Notice

Copyright

Revised September, 2015 (4.3.6.0)

Contents

Chapter 1

Preface

This guide describes tasks you complete to install and run your Greenplum Database system, and learn about the main features of the system.

About This Guide

This guide provides information and instructions for installing, initializing, and running a Greenplum Database system.

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

This guide contains the following chapters and appendices:

Introduction to Greenplum Database — Information about the Greenplum Database system architecture and components.

Installing a Single-Node Greenplum Database Instance — Instructions for installing and intializing a single-node Greenplum Database instance.

Getting Started with Greenplum Database — An introduction to Greenplum Database in a hands-on tutorial format.

Introduction to Greenplum In-Database Analytics — An introduction to using the MADlib in-database analytics library in Greenplum Database.

Operating System Configuration — Detailed configuration parameters for supported operating systems.

About the Greenplum Database Documentation Set

The Greenplum Database 4.3 server documentation set consists of the following guides.

Table 1: Greenplum Database server documentation set

Guide Name	Description
Greenplum Database Administrator Guide	Information for administering the Greenplum Database system and managing databases. It covers topics such as Greenplum Database architecture and concepts and everyday system administration tasks such as configuring the server, monitoring system activity, enabling high-availability, backing up and restoring databases, and expanding the system. Database administration topics include configuring access control, creating databases and database objects, loading data into databases, writing queries, managing workloads, and monitoring and troubleshooting performance.
Greenplum Database Reference Guide	Reference information for Greenplum Database systems: SQL commands, system catalogs, environment variables, character set support, datatypes, the Greenplum MapReduce specification, postGIS extension, server parameters, the gp_toolkit administrative schema, and SQL 2008 support.
Greenplum Database Utility Guide	Reference information for command-line utilities, client programs, and Oracle compatibility functions.
Greenplum Database Installation Guide	Information and instructions for installing and initializing a Greenplum Database system.

Document Conventions

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

- *Command Syntax Conventions*

Command Syntax Conventions

Table 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

Pivotal/Greenplum support, product, and licensing information can be obtained as follows.

Product information and Technical Support

For technical support, documentation, release notes, software updates, or for information about Pivotal products, licensing, and services, go to *www.gopivotal.com*.

Additionally, you can still obtain product and support information from the EMC Support Site at: *http://support.emc.com*

Chapter 2

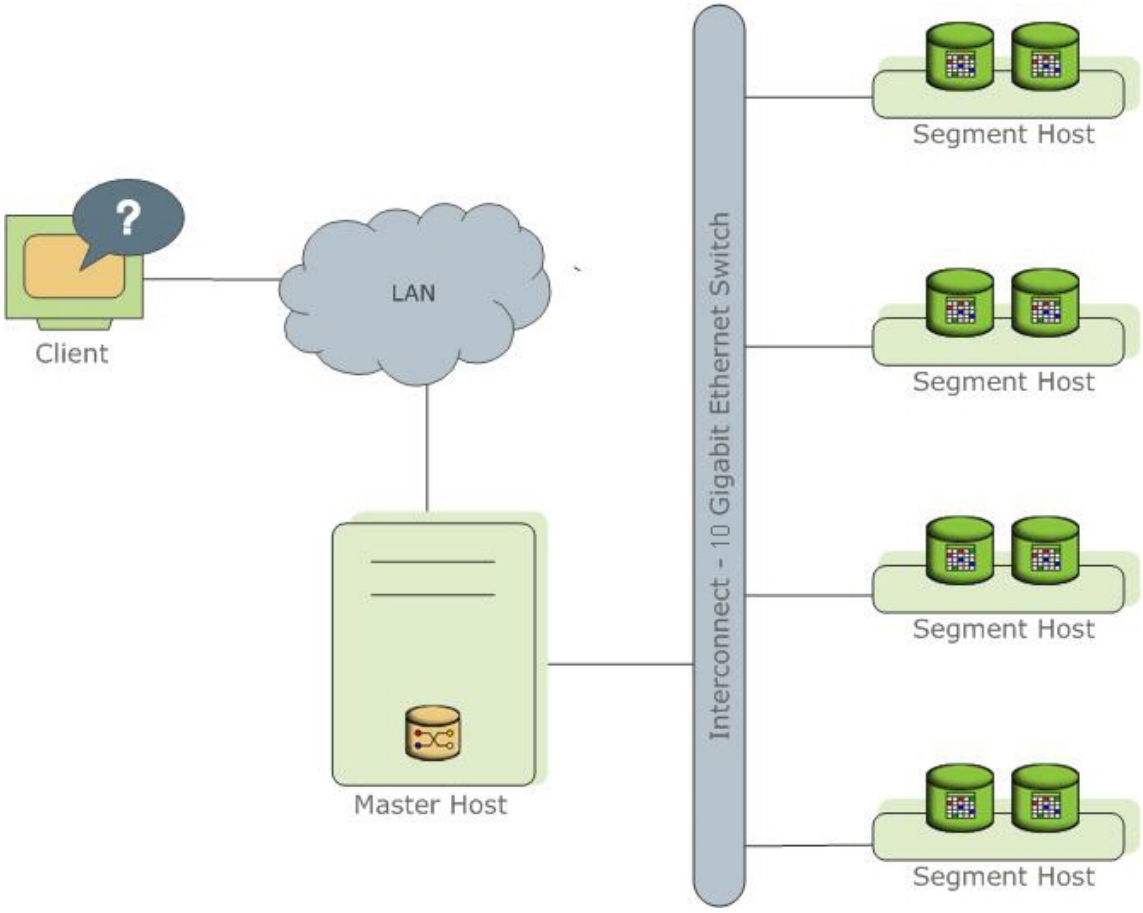
Introduction to Greenplum Database

Greenplum Database is a massively parallel processing (MPP) database server based on PostgreSQL open-source technology. An MPP system is a cluster with two or more PostgreSQL database instances cooperating to accomplish a task, each instance with its own memory and storage. Greenplum Database uses this high-performance system architecture to distribute the load of multi-terabyte data warehouses, and can use all of the system's resources in parallel to process a query.

Greenplum Database Overall Architecture

Greenplum Database is essentially several Pivotal-enhanced PostgreSQL database instances acting together as one cohesive database management system (DBMS).

Each Greenplum Database has a *master* instance and two or more *segment* instances. Database users interact with the master as they would a regular PostgreSQL DBMS. The following illustration shows a Greenplum Database instance with a master instance and eight segment instances.



The master in this illustration is deployed on a dedicated host. A *host* is a computer—either physical or virtual—with an operating system, memory, disk storage, CPU, and one or more network interfaces.

The *master*, or *master instance*, is the Greenplum Database server that client programs interact with. The master listens for client connections on a unique port on the master host, port 5432 by default.

The eight segments shown in the illustration are deployed on four *segment hosts*. Each segment host is a separate computer with its own operating system, memory, CPU, storage, and network interfaces. Like the master host, a segment host may be a standalone computer or a virtual machine.

The segment hosts in this example each host two *segments*, or *segment instances*. A segment is a database server instance managing a portion of the data on disk storage that has been allocated to it. Each segment instance listens on a unique port on the segment host. Deploying multiple segments on a host ensures the host's resources are fully utililized. A multi-core host with multiple segments can be expected to accomplish more work than a multi-core host with a single segment.

The master instance coordinates the entire Greenplum Database instance, distributing requests to the segments and combining the results returned from the segments.

The master and segment hosts communicate over the *interconnect*, a standard gigabit Ethernet using an enhanced User Datagram Protocol (UDP) that is faster than TCP and more reliable than standard UDP.

High availability can be achieved by mirroring the master and each segment instance on separate hosts, and providing a redundant interconnect. This ensures that a standby is available for any component that fails.

Shared Nothing vs. Shared Disk Architectures

The Greenplum Database architecture is called a *shared nothing* architecture because each segment independently manages a portion of the database with its own CPU, memory, and disk. In contrast, a distributed DBMS system with a *shared disk* (or *shared everything)* architecture has multiple database server instances managing a single database instance on a shared collection of disks. The shared disk and shared nothing architectures have different advantages and disadvantages.

In a shared disk system, all data is local to each database server. There is no need to send data over the network to another server to process queries that join tables. However, the network disk storage solution and the software that coordinates disk sharing between servers can limit the amount of data and the number of database servers that can be added to the database cluster. Expensive server and NAS hardware are needed to increase capacity and maintain acceptable query response times.

The primary advantages of the Greenplum shared nothing architecture are greater scalability, lower cost, and faster query execution. Commodity PCs and disks can be added to increase capacity at little expense, so it is possible to add large numbers of CPUs to the database instance at a very low cost-per-CPU. Each segment manages a discrete portion of the data, so there is no need to coordinate shared disk access between servers.

Performance on the shared nothing architecture can suffer when data is unevenly distributed among the segments or is distributed in a way that requires sending large volumes of data between segments to execute queries. Therefore, a shared nothing architecture like Greenplum Database requires up-front planning to ensure that all of the segments participate fully and their resources are used effectively.

Greenplum Database MPP Features

Pivotal has modified and supplemented the internals of PostgreSQL to support the parallel structure of Greenplum Database. For example, the system catalog, query planner, optimizer, query executor, and transaction manager components have been modified and enhanced to execute queries simultaneously across all of the parallel segments. Adding more segments to a Greenplum instance, therefore, can increase data capacity and query response.

Greenplum Database also includes features designed to optimize PostgreSQL for business intelligence (BI) workloads. For example, Greenplum Database has added parallel data loading (external tables), resource management, and query optimizations, which are not found in standard PostgreSQL.

Storage enhancements in Greenplum Database include column-oriented tables, append-optimized tables, and data partitioning. These features allow faster data loading and data management, optimized and compressed disk storage, and fast, targeted disk reads, all keys to agile processing of very large analytics datasets and data warehouses.

Single-Node Greenplum Database Configuration

The smallest multi-segment Greenplum Database configuration is a single-node system, which runs as several distinct processes on a single host computer. There is a *master* instance and one or more *segment* instances, typically one segment per CPU core. The single-node configuration is easily installed and can be used on a development workstation or to evaluate the Greenplum Database software. It lacks high availability features, such as mirroring, but is otherwise a complete Greenplum Database instance running on a single host.

The next section of this guide provides instructions for installing a single-node Greenplum Database instance with a master and two segments. The resulting installation can then be used with the exercises to explore Greenplum Database features.

The following illustration shows a single-node Greenplum Database instance.

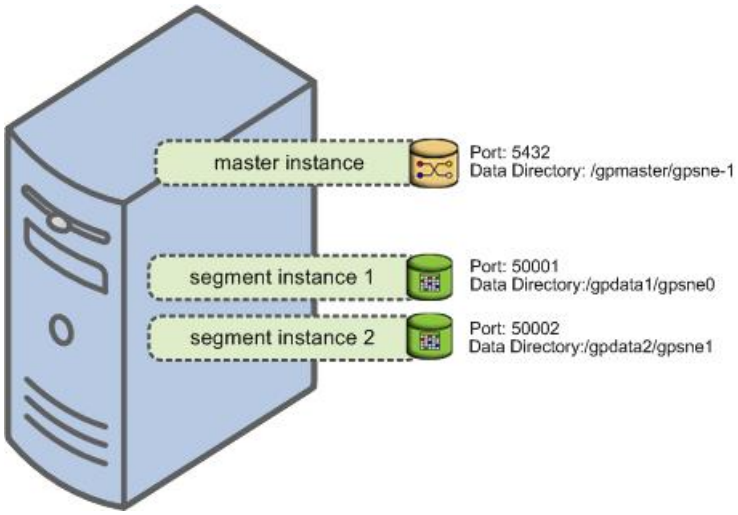


Figure 1: Single-Node Greenplum Database Configuration

The *master* is the entry point to the Greenplum Database system. It is the database instance where clients connect and submit SQL statements. The master coordinates the work of the other database instances in the system, the segments.

The *segments* handle data processing and storage. Each segment contains a portion of the data. When a user issues a query on the master, the master distributes the work to the two segments. A query process is created on each segment to handle the request. When the segments return individual results, the master combines them and presents the final result to the user.

Setting up a production Greenplum Database instance includes planning for capacity, networking, security, and high availability. These topics are beyond the scope of this guide. Please refer to *Greenplum Database Installation Guide* for help installing a production system.

Find Out More

Learn more about Greenplum Database system architecture in *Greenplum Database Administrator Guide*.

Chapter 3

Installing a Single-Node Greenplum Database Instance

This chapter provides instructions for installing a single-node Greenplum Database instance on a Red Hat Enterprise Linux (or CentOS) computer, or on a Mac OS X computer. The instance will have one master and two segments. See *Greenplum Database Installation Guide* for comprehensive instructions on installing a multi-node production Greenplum Database system.

Here is a list of steps to install and configure the Greenplum Database instance:

1. Configure your operating system following instructions in *Operating System Configuration*
2. *Add the Greenplum Database Administrative User Account*
3. *Install the Greenplum Database Software*
4. *Prepare the Data Directory Locations*
5. *Configure Greenplum Database Environment Variables*
6. *Initialize Greenplum Database*
7. *Set the MASTER_DATA_DIRECTORY Environment Variable*

Add the Greenplum Database Administrative User Account

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

For a production system, Pivotal recommends that you:

- designate a user account that will own the Greenplum Database installation
- always start and administer Greenplum Database as this user

> **Note:** Greenplum Database is not supported for production use on Mac OS X. For evaluation or development on Mac OS X, it is acceptable to install a single-node Greenplum Database instance using your own user account. When this documentation references the `gpadmin` user, you can substitute your own Mac OS X user name.

On Linux, you can create a dedicated user account to run Greenplum Database, `gpadmin`, by convention. To add this new user, enter the following commands as `root`:

```
# useradd gpadmin
# passwd gpadmin
New password: password
Retype new password: password
```

The user must have permission to access services and directories that are needed to install and run Greenplum Database. For example, the user needs access to the Greenplum Database installation directory, directories where data is to be stored, and to services such as the Java virtual machine (JVM).

(Optional) If your environment has multiple administrative users or if you will be running multiple Greenplum Database systems on the same machine, you might want to designate a group of users to own your Greenplum Database installation. For the purposes of this documentation, we will use the group name of `gpadmin` as well. To add a new group, for example, run the following commands as `root`:

```
# groupadd gpadmin
# usermod -g gpadmin gp_user1
# usermod -g gpadmin gp_user2
```

Install the Greenplum Database Software

1. Download or copy the Greenplum Database installer file to the system on which you want to install Greenplum Database. Installer files are available at *http://network.pivotal.io*.
2. Unzip the installer file. For example:

   ```
   unzip greenplum-db-4.3.x.x-PLATFORM.zip
   ```

3. As the `root` user, run the installer using `bash`. For example:

   ```
   # /bin/bash greenplum-db-4.3.x.x-PLATFORM.bin
   ```

4. The installer prompts you to accept the Greenplum Database license agreement. Type `yes` to accept the license agreement.
5. The installer prompts you to provide an installation path. Press `ENTER` to accept the default install path (/usr/local/greenplum-db-4.3.*x.x*), or enter an absolute path to an installation location. You must have write permissions to the location you specify.
6. The installer installs the Greenplum Database software and creates a `greenplum-db` symbolic link in the same directory as your version-specific Greenplum Database installation directory. The symbolic link is used to facilitate patch maintenance and upgrades between versions. The installed location is referred to as `$GPHOME`.
7. Change the ownership of your Greenplum Database installation so that it is owned by the `gpadmin` user or group. For example, run the following commands as `root`:

   ```
   # chown -R gpadmin /usr/local/greenplum-db-4.3.x.x
   # chgrp -R gpadmin /usr/local/greenplum-db-4.3.x.x
   ```

Prepare the Data Directory Locations

Each Greenplum Database master and segment has a designated storage area on disk that is called the *data directory* location. This is the file system location where the database data is stored. In a single-node Greenplum Database, you initialize a Greenplum Database master instance and two or more segment instances on the same system, each requiring a data directory location. These directories should have sufficient disk space for your data and be owned by the `gpadmin` user.

Remember that the data directories of the segment instances are where the user data resides, so they must have enough disk space to accommodate your expected data capacity. For the master instance, only system catalog tables and system metadata are stored in the master data directory.

To create the data directory locations:

1. Create or choose a directory that will serve as your master data storage area. For example, run the following commands as `root`:

```
# mkdir /gpmaster
# chown gpadmin /gpmaster
# chgrp gpadmin /gpmaster
```

2. Create or choose the directories that will serve as your segment storage areas. For this two-segment, single-node example system, run the following commands as `root`:

```
$ mkdir /gpdata1
$ chown gpadmin /gpdata1
$ chgrp gpadmin /gpdata1

$ mkdir /gpdata2
$ chown gpadmin /gpdata2
$ chgrp gpadmin /gpdata2
```

Configure Greenplum Database Environment Variables

A `greenplum_path.sh` file is provided in your `$GPHOME` directory with environment variable settings for Greenplum Database. You can source this in the `gpadmin` user's startup shell profile (such as `.bashrc`), or in `/etc/profile` if you want to set the environment variables for all users.

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

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

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

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

or

```
$ source /etc/profile
```

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

Initialize Greenplum Database

The Greenplum `gpinitsystem` utility initializes the Greenplum Database system. For a single-node Greenplum Database system, you are initializing a master and two (or more) segment instances on the same system.

After the Greenplum Database system is initialized and started, you can then create and manage databases by connecting to the Greenplum master database process.

To initialize Greenplum Database:

1. Log in to the system as the `gpadmin` user:

```
su - gpadmin
```

2. Copy the `hostlist_singlenode` and `gpinitsystem_singlenode` example files from your Greenplum Database installation to the current directory:

```
cp $GPHOME/docs/cli_help/gpconfigs/*_singlenode .
```

3. Make the `hostlist_singlenode` and `gpinitsystem_singlenode` files writable so that you can edit them:

```
chmod +w *_singlenode
```

4. Edit the `gpinitsystem_singlenode` file and enter your configuration settings. There are informative comments within the file to help you make your changes. At minimum, you must set the `MASTER_HOSTNAME` parameter to the name of your system, or `127.0.0.1`. Here is an example of the default parameters in this file:

```
ARRAY_NAME="GPDB SINGLENODE"
MACHINE_LIST_FILE=./hostlist_singlenode
SEG_PREFIX=gpsne
PORT_BASE=40000
declare -a DATA_DIRECTORY=(/gpdata1 /gpdata2)
MASTER_HOSTNAME=127.0.0.1
MASTER_DIRECTORY=/gpmaster
MASTER_PORT=5432
```

5. Edit the `hostlist_singlenode` file and replace the first line with the name of your system or `127.0.0.1`.
6. Run the `gpssh-exkeys` utility to exchange ssh keys for the local host:

```
$ gpssh-exkeys -h 127.0.0.1
```

7. Run the following command to initialize Greenplum Database:

```
$ gpinitsystem -c gpinitsystem_singlenode
```

   The utility verifies your setup information and makes sure that the data directories specified in the `gpinitsystem_singlenode` configuration file are accessible. If all of the verification checks are successful, the utility prompts you to confirm your Greenplum Database configuration before creating the system. For example:

```
=> Continue with Greenplum creation? y
```

8. The utility begins setup and the initialization of the master instance and each segment instance in the system. This process can take several minutes (depending on the number of segments to initialize). At the end of a successful setup, the utility starts your Greenplum Database system. You should see:

```
=> Greenplum Database instance successfully created.
```

   Note: For a single-node Greenplum Database instance, you can disregard the warning messages about the use of '127.0.0.1' or 'localhost'. These warnings only apply to a multi-node Greenplum Database system.

Set the MASTER_DATA_DIRECTORY Environment Variable

The Greenplum Database management utilities require that you set the MASTER_DATA_DIRECTORY environment variable. This variable points to the directory created by the gpinitsystem utility in the master data directory location.

For example, add a line similar to the following to the `gpadmin` user's profile file (such as .bashrc):

```
MASTER_DATA_DIRECTORY=/gpmaster/gpsne-1
export MASTER_DATA_DIRECTORY
```

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

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

Find Out More

Learn more about installing and configuring Greenplum Database in *Greenplum Database System Administrators Guide*.

Chapter 4

Getting Started with Greenplum Database

This chapter provides an introduction to basic Greenplum Database operations and introduces the features that make Greenplum Database the best choice for big data analytics.

Sample data

To introduce Greenplum Database, we use a public data set, the Airline On-Time Statistics and Delay Causes data set, published by the United States Department of Transportation at *http://www.transstats.bts.gov/*.

The On-Time Performance dataset records flights by date, airline, originating airport, destination airport, and many other flight details. Data is available for flights since 1987. The exercises in this guide use data for about a million flights in 2009 and 2010.

The FAA uses the data to calculate statistics such as the percent of flights that depart or arrive on time by origin, destination, and airline.

You will create a `tutorial` database, define the schema and tables to hold the data, and load the data from external files. Then you install the MADlib in-database analytics library and run a few analyses to answer questions such as these:

- Are performances differences between an airline's flights originating from different airports significant?
- Are performance differences between different airlines originating from the same airport significant?
- Do delayed flights over longer distances make up time so they can arrive closer to the scheduled arrival time?

You can download a ZIP file containing the sample data and queries from *http://gpdb.docs.pivotal.io/gs/faa-samples-A01.zip* and unzip the file into your home directory. The file extracts to a directory which contains SQL scripts and data files used in the exercises.

The `tutorial` database will contain a fact table and several dimension tables. The fact table holds historical flight records for a few months in 2009 and 2010. You will create two versions of the fact table, a row-oriented version (`otp_r`) and a column-oriented version (`otp_c`). You will load the dimension tables from CSV-formatted text files. For the fact table, you will use an Extract, Load, Transform (ELT) process to load data.

You are encouraged to review the SQL scripts in the `faa` directory as you work through this introduction. You can run most of the exercises by entering the commands yourself or by executing a script in the `faa` directory.

Start and stop Greenplum Database

The Greenplum Database installation includes utility commands to shut down the system gracefully, start it up, and view the current status. The commands are in the bin subdirectory in the Greenplum installation, that is, `$GPHOME/bin`. When your environment has been set up by sourcing `$GPHOME/greenplum_path.sh` in your shell, the commands are on your path.

Execute these commands as the user running Greenplum Database, for example `gpadmin`.

You can get online help for any Greenplum command by running it with the --help option.

To check the current status of the Greenplum Database instance:

1. Run the `gpstate` command:

   ```
   $ gpstate
   ```

   The command displays the status of the master and segment processes. If the Greenplum system is not running, the command displays an error message.

To shut down the Greenplum Database instance:

1. Run the `gpstop` command:

   ```
   $ gpstop
   ```

   Displays parameters for the master and segment processes that are to be stopped.
2. Enter `y` when prompted to stop the Greenplum instance.

To start the Greenplum Database instance:

1. Run the `gpstart` command:

   ```
   $ gpstart
   ```

   The command displays parameters for the master and segment processes that are to be started.
2. Enter `y` when prompted to continue starting up the instance.

Find Out More

Learn more about managing Greenplum Database in the *Greenplum Database Administrator Guide*. See *Greenplum Database Utility Guide* for information about the included utilities.

About Greenplum instances, databases, and templates

A Greenplum Database system is called an *instance* or *cluster*. It is possible to have more than one instance installed; the environment variables set by the `$GPHOME/greenplum_path.sh` file point to the current instance and ensure that the executable programs and scripts are on your path.

An instance can manage multiple *databases*. A database typically contains all of the data and objects needed for one or more related applications. A client can connect to only one database at a time; it is not possible to query across databases.

When newly installed, a Greenplum Database instance has three databases:

- The `template1` database is the default template used to create new databases. If you have objects that should exist in every database managed by your Greenplum Database instance, you can create them in the `template1` database.
- The `postgres` and `template0` databases are used internally and should not be modified or dropped. If you have modified the `template1` database you can use the `template0` database to create a new database without your modifications.

Connect to a database with psql

The `psql` command is an interactive, command-line client used to access a Greenplum database.

By default, `psql` attempts to connect to the Greenplum instance on the current host (`localhost`) at port 5432, with the database user and database name the same as your current login name. For example, if you are logged in as `gpadmin`, entering `psql` at a shell prompt with no options attempts to connect to the `gpadmin` database with the `gpadmin` role. Since there is no `gpadmin` database by default, you must at least specify the database name on the `psql` command line.

The default database name and other connection parameters can be overridden on the command line or by setting environment variables. The essential connection parameters are the name of the host, the port number where the master is listening, the database user name (role), and the database name.

To connect to a database with default connection parameters:

```
$ psql template1
```

To specify connection parameters on the command line:

```
$ psql -h localhost -p 5432 -U gpadmin template1
```

To set connection parameters in the environment:

```
$ export PGPORT=5432
$ export PGHOST=localhost
$ export PGDATABASE=template1
$ psql
```

Entering commands in psql

When you successfully connect to a database, `psql` displays a prompt, the database name followed by `=#`. For example:

```
$ psql template1
psql (8.2.15)
Type "help" for help.
template1=#
```

If you connect with a non-administrative database role, you get a different prompt: `template1=>`.

SQL statements can be quite long, so you can enter them on multiple lines. On continuation lines, the `psql` prompt changes to the database name followed by `-#`, or `->` for a non-administrative role. Terminate a SQL statement with a semicolon. `psql` saves your input in a query buffer until it sees the semicolon, and only then does it execute the statement.

In addition to SQL statements, you can enter `psql` *meta-commands*, which begin with a backslash (`\`). Here are some common `psql` meta-commands:

- Enter `\e` to edit the buffer in an external editor (vi by default)
- Enter `\p` to display the contents of the query buffer
- Enter `\g` instead of a semicolon to terminate a SQL statement
- Enter `\r` to reset the query buffer, abandoning what you have entered
- Enter `\l` to list databases
- Enter `\d` to list tables, views, and sequences
- Enter `\q` to exit `psql`
- Enter `\h` to display help for SQL statements
- Enter `\?` to display help for `psql`

You can get help for an SQL statement even after you have started entering the text of the statement. The `\h` meta-command and everything up to the end of the line is omitted from the query buffer.

There are many more useful `psql` meta-commands. You will encounter some of them in later exercises. See *Greenplum Database Utility Guide* for the `psql` reference documentation.

Set up authentication

Following a fresh installation, Greenplum Database authentication is set up to accept database connections only from the `gpadmin` user on the master host. To allow other users to connect, and to allow connections from other hosts on the network, you must configure Greenplum Database authentication on the master host. Note that the segments are automatically configured to accept connections from the master only.

Greenplum Database supports several authentication schemes, including passwords, LDAP, Kerberos (GSSAPI), Radius, client certificates, PAM, and SSPI. In this exercise, we set up password authentication using MD5-encrypted passwords. The passwords are encrypted and saved in a system table in the database. See *Greenplum Database Administrator Guide* for information about setting up other authentication methods.

You configure authentication in the `$MASTER_DATA_DIRECTORY/pg_hba.conf` file. Lines that begin with a `#` character are comments and describe the file's syntax and the available configuration options.

Each non-comment line is an entry that will be compared to incoming connection requests. The first line that matches a request determines the method of authentication, so the order of the lines is important. The entries allow you to specify different authentication methods depending on the type of connection, the target database, and the database role (user or group) of the requester.

The type of connection can be `local` (for a local Unix socket connection), `host` (for an unencrypted TCP/IP connection), or `hostssl` (for an SSL-encrypted TCP/IP connection). If the connection type is `host` or `hostssl`, the entry contains a CIDR mask that determines which network hosts the entry applies to.

In the following exercise, you set up authentication to allow any user who is a member of the `users` group to connect to the `tutorial` database by supplying a password. SSL is required for connections from other hosts.

1. While logged in as `gpadmin`, edit the $MASTER_DATA_DIRECTORY/pg_hba.conf file:

   ```
   $ vi $MASTER_DATA_DIRECTORY/pg_hba.conf
   ```

2. At the end of the file, add the following entries.

   ```
   local     tutorial   +users                  md5
   local     tutorial   +users   127.0.0.1/28   md5
   hostssl   tutorial   +users   samenet        md5
   ```

   See the *Greenplum Database Administrator Guide* for a complete description of the syntax and permitted values for these entries or read the comments in the `pg_hba.conf` file.

3. Save your changes and reload the `pg_hba.conf` and `postgresql.conf` files with the following command:

   ```
   $ gpstop -u
   ```

In the next section, you create user roles and the `users` group.

Set up roles

Greenplum Database manages database access using roles. Initially, there is one superuser role—the role associated with the OS user who initialized the database instance, usually `gpadmin`. This user owns all of the Greenplum Database files and OS processes, so it is important to reserve the `gpadmin` role for system tasks only.

A role can be a *user* or a *group*. A user role can log in to a database; that is, it has the `LOGIN` *attribute*. A user or group role can become a member of a group.

Permissions can be granted to users or groups. Initially, of course, only the `gpadmin` role is able to create roles. You can add roles with the `createuser` utility command, `CREATE ROLE` SQL command, or the `CREATE USER` SQL command. The `CREATE USER` command is the same as the `CREATE ROLE` command except that it automatically assigns the role the `LOGIN` attribute.

Create a user with the createuser utility command

1. Log in to the master host as the `gpadmin` user.
2. Enter the `createuser` command and reply to the prompts:

```
$ createuser -P user1
Enter password for new role:
Enter it again:
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n) y
Shall the new role be allowed to create more new roles? (y/n) n
```

The next example is the equivalent of this one, but using the `CREATE USER` SQL command in `psql`.

Create a user with the CREATE USER command

1. Connect to the `template1` database as `gpadmin`:

```
$ psql template1
```

2. Create a user with the name `user2`:

```
=# CREATE USER user2 WITH PASSWORD 'changeme' CREATEDB NOSUPERUSER;
```

3. Display a list of roles:

```
# \du
                     List of roles
 Role name |           Attributes              | Member of
-----------+-----------------------------------+-----------
 gpadmin   | Superuser, Create role, Create DB |
 user1     | Create DB                         |
 user2     | Create DB                         |
```

Create a users group and add the users to it

1. While connected to the `template1` database as `gpadmin` enter the following SQL commands:

```
=# CREATE ROLE users;
=# GRANT users TO user1, user2;
```

2. Display the list of roles again:

```
=# \du List of roles
 Role name  | Attributes                          | Member of
```

```
-----------+----------------------------------+-----------
gpadmin    | Superuser, Create role, Create DB |
user1      | Create DB                         | {users}
user2      | Create DB                         | {users}
users      | Cannot login                      |
```

Create a new database

Create a new database with the `CREATE DATABASE` SQL command in `psql` or the `createdb` utility command in a terminal. The new database is a copy of the `template1` database, unless you specify a different template.

To use the `CREATE DATABASE` command, you must be connected to a database. With a newly installed Greenplum Database system, you can connect to the `template1` database to create your first user database. The `createdb` utility, entered at a shell prompt, is a wrapper around the `CREATE DATABASE` command.

In this exercise you will drop the `tutorial` database if it exists and then create it new with the `createdb` utility.

1. Enter these commands to drop the `tutorial` database if it exists:

```
$ dropdb tutorial
```

2. Enter the `createdb` command to create the tutorial database, with the defaults:

```
$ createdb tutorial
```

3. Verify that the database was created using the `psql -l` command:

```
$ psql -l
  List of databases
   Name    |  Owner  | Encoding | Access privileges
-----------+---------+----------+---------------------
 postgres  | gpadmin | UTF8     |
 template0 | gpadmin | UTF8     | =c/gpadmin
                                : gpadmin=CTc/gpadmin
 template1 | gpadmin | UTF8     | =c/gpadmin
                                : gpadmin=CTc/gpadmin
 tutorial  | gpadmin | UTF8     |
 (4 rows)
```

4. Connect to the `tutorial` database as `user1`, entering the password you created for `user1` when prompted:

```
$ psql -U user1 tutorial
Password for user user1:
psql (8.2.15)
Type "help" for help.
=> \q
```

Grant database privileges to users

In a production database, you should grant users the minimum permissions required to do their work. For example, a user may need `SELECT` permissions on a table to view data, but not `UPDATE`, `INSERT`, or `DELETE` to modify the data.

To complete the exercises in this guide, the database users will require permissions to create and manipulate objects in the `tutorial` database.

1. Connect to the `tutorial` database as `gpadmin`.

```
$ psql -U gpadmin tutorial
```

2. Grant `user1` and `user2` all privileges on the `tutorial` database.

```
=# GRANT ALL PRIVILEGES ON DATABASE tutorial
    TO user1, user2;
```

Log out of `psql` and perform the next steps as the `user1` role.

Create a schema and set search path

A database *schema* is a named container for a set of database objects, including tables, data types, and functions. A database can have multiple schemas. Objects within the schema are referenced by prefixing the object name with the schema name, separated with a period. For example, the `person` table in the `employee` schema is written `employee.person`.

The schema provides a namespace for the objects it contains. If the database is used for multiple applications, each with its own schema, the same table name can be used in each schema —`employee.person` is a different table than `customer.person`. Both tables could be accessed in the same query as long as they are qualified with the schema name.

The database contains a *schema search path*, which is a list of schemas to search for objects names that are not qualified with a schema name. The first schema in the search path is also the schema where new objects are created when no schema is specified. The default search path is `"$user",public`, so by default, each object you create belongs to a schema associated with your login name.

In this exercise, you create an `faa` schema and set the search path so that it is the default schema.

1. Change to the directory containing the FAA data and scripts:

   ```
   $ cd ~/faa
   ```

   Like the OS shell, `psql` has a current working directory (CWD), which is initially the same as your OS shell CWD when you invoke `psql`. You can change the `psql` CWD using the `\cd` command. Changing to the `faa` directory allows you to reference the data files and SQL scripts without having to specify the path.
2. Connect to the `tutorial` database with `psql`:

   ```
   $ psql -U user1 tutorial
   ```
3. Create the `faa` schema:

   ```
   =# DROP SCHEMA IF EXISTS faa CASCADE;
   =# CREATE SCHEMA faa;
   ```
4. Add the `faa` schema to the search path:

   ```
   =# SET SEARCH_PATH TO faa, public, pg_catalog, gp_toolkit;
   ```
5. View the search path:

   ```
   =# SHOW search_path;
   ```

The search path you set above is not persistent; you have to set it each time you connect to the database. You can associate a search path with the user role by using the `ALTER ROLE` command:

```
=# ALTER ROLE user1
SET search_path TO faa, public, pg_catalog, gp_toolkit;
```

Then each time you connect to the database with that role, the search path is restored.

Learn more about working with database schemas in *Greenplum Database Administrator Guide*.

Create tables

The `CREATE TABLE` SQL statement creates a table in the database.

About the distribution policy

The definition of a table includes the distribution policy for the data, which has great bearing on system performance. The goals for the distribution policy are to:

- distribute the volume of data and query execution work evenly among the segments, and to
- enable segments to accomplish the most expensive query processing steps locally.

The distribution policy determines how data is distributed among the segments. Defining an effective distribution policy requires an understanding of the data's characteristics, the kinds of queries that will be run once the data is loaded into the database, and what distribution strategies best utilize the parallel execution capacity of the segments.

Use the `DISTRIBUTED` clause of the `CREATE TABLE` statement to define the distribution policy for a table. Ideally, each segment will store an equal volume of data and perform an equal share of work when processing queries. There are two kinds of distribution policies:

`DISTRIBUTED BY (column, ...)` defines a distribution key from one or more columns. A hash function applied to the distribution key determines which segment stores the row. Rows that have the same distribution key are stored on the same segment. If the distribution keys are unique, the hash function ensures the data is distributed evenly. The default distribution policy is a hash on the primary key of the table, or the first column if no primary key is specified.

`DISTRIBUTED RANDOMLY` distributes rows in round-robin fashion among the segments.

When different tables are joined on the same columns that comprise the distribution key, the join can be accomplished at the segments, which is much faster than joining rows across segments. The random distribution policy makes this impossible, so it is best practice to define a distribution key that will optimize joins.

Learn more about table distribution policies in *Greenplum Database Administrator Guide*.

Execute the CREATE TABLE script in psql

The `CREATE TABLE` statements for the `faa` database are in the `faa/create_dim_tables.sql` script. Open the script in a text editor to see the text of the commands that will be executed when you run the script.

The `psql \i` command executes a script.

1. Execute the `faa/create_dim_tables.sql` script:

```
=# \i create_dim_tables.sql
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
```

2. List the tables that were created, using the `psql \dt` command.

```
=# \dt
                    List of relations
 Schema |          Name          | Type  |  Owner   | Storage
--------+------------------------+-------+----------+---------
 faa    | d_airlines             | table | gpadmin | heap
 faa    | d_airports             | table | gpadmin | heap
 faa    | d_cancellation_codes | table | gpadmin | heap
```

```
 faa    | d_delay_groups       | table | gpadmin | heap
 faa    | d_distance_groups    | table | gpadmin | heap
 faa    | d_wac                | table | gpadmin | heap
(6 rows)
```

Load data into tables

Loading external data into Greenplum Database tables can be accomplished in different ways. We will use three methods to load the FAA data:

The simplest data loading method is the SQL `INSERT` statement. You can execute `INSERT` statements directly with `psql` or another interactive client, run a script containing `INSERT` statements, or run a client application with a database connection. This is the least efficient method for loading large volumes of data and should be used only for small amounts of data.

You can use the `COPY` command to load the data into a table when the data is in external text files. The `COPY` command syntax allows you to define the format of the text file so the data can be parsed into rows and columns. This is faster than `INSERT` statements but, like `INSERT` statements, it is not a parallel process and should not be used for large fact tables.

The SQL `COPY` command requires that external files be accessible to the host where the master process is running. On a multi-node Greenplum Database system, the data files may be on a file system that is not accessible to the master process. In this case, you can use the `psql \copy` meta-command, which streams the data to the server over the `psql` connection. The scripts in this tutorial use the `\copy` meta-command.

You can use a pair of Greenplum utilities, `gpfdist` and `gpload`, to load external data into tables at high data transfer rates. This method takes advantage of the Greenplum Database MPP architecture and is recommended for large data loads.

In the following exercises, you load data into the `tutorial` database using each of these methods.

Load data with the INSERT statement

The `faa.d_cancellation_codes` table is a simple two-column look-up table, easily loaded with an `INSERT` statement.

1. Use the `\d psql` meta-command to describe the `faa.d_cancellation_codes` table:

```
=# \d d_cancellation_codes
Table "faa.d_cancellation_codes"
 Column      | Type | Modifiers
-------------+------+-----------
 cancel_code | text |
 cancel_desc | text |
Distributed by: (cancel_code)
```

2. Load the data into the table with a multirow `INSERT` statement:

```
=# INSERT INTO faa.d_cancellation_codes
VALUES ('A', 'Carrier'),
       ('B', 'Weather'),
       ('C', 'NAS'),
       ('D', 'Security'),
       ('', 'none');
```

   Alternatively, use the `\i` meta-command to execute the `insert_into_cancellation_codes.sql` script.

3. Display the data:

```
=# SELECT * FROM faa.d_cancellation_codes
       ORDER BY cancel_code;
 cancel_code | cancel_desc
-------------+-------------
             | none
 A           | Carrier
```

```
 B           | Weather
 C           | NAS
 D           | Security
(5 rows)
```

Load data with the COPY statement

The `COPY` statement moves data between the file system and database tables. Data for five of the FAA tables is in the following CSV-formatted text files:

1. In a text editor, review the `.csv` data files.
   - `L_AIRLINE_ID.csv`
   - `L_AIRPORTS.csv`
   - `L_DISTANCE_GROUP_250.csv`
   - `L_ONTIME_DELAY_GROUPS.csv`
   - `L_WORLD_AREA_CODES.csv`

   Notice that the first line of each file contains the column names and that the last line of the file contains the characters "`\.`", which signals the end of the input data.
2. In a text editor, review the following scripts:
   - `copy_into_airlines.sql`
   - `copy_into_airports.sql`
   - `copy_into_delay_groups.sql`
   - `copy_into_distance_groups.sql`
   - `copy_into_wac.sql`

   The `HEADER` keyword prevents the `\copy` command from interpreting the column names as data.
3. Run the scripts:

```
=# \i copy_into_airlines.sql
=# \i copy_into_airports.sql
=# \i copy_into_delay_groups.sql
=# \i copy_into_distance_groups.sql
=# \i copy_into_wac.sql
```

Load data with gpfdist

For the FAA fact table, we will use an ETL (Extract, Transform, Load) process to load data from the source gzip files into a *loading table*, and then insert the data into a query and reporting table. For the best load speed, use the `gpfdist` Greenplum utility to distribute the rows to the segments.

In a production system, `gpfdist` runs on the servers where the data is located. With a single-node Greenplum Database instance, there is only one host, and you run `gpfdist` on it. Starting `gpfdist` is like starting a file server; there is no data movement until a request is made on the process.

> **Note:** This exercise loads data using the Greenplum Database *external table* feature to move data from external data files into the database. Moving data between the database and external tables is a security consideration, so only superusers are permitted to use the feature. Therefore, you will run this exercise as the `gpadmin` database user.

1. Start a new terminal and execute gpfdist. Use the –d switch to set the “home” directory used to search for files in the faa directory. Use the –p switch to set the port and background the process.

```
$ gpfdist -d $HOME/faa -p 8081 > gpfdist.log 2>&1 &
[1] 5231
```

2. Check that `gpfdist` is running with the `ps` command:

```
$ ps -A | grep gpfdist
```

```
5231 ttys001   0:00.00 gpfdist -d /home/gpadmin/faa -p 8081
5257 ttys001   0:00.00 grep gpfdist
```

3. View the contents of the `gpfdist` log.

```
$ cat gpfdist.log
Serving HTTP on port 8081, directory /home/gpadmin/faa
```

   Disregard any warning about the bind function failing.

4. In another terminal, start a `psql` session as `gpadmin` and execute the `create_load_tables.sql` script.

```
=# \i create_load_tables.sql
```

   This script creates two tables: the `faa_otp_load` table, into which `gpfdist` will load the data, and the `faa_load_errors` table, where load errors will be logged. (The `faa_load_errors` table may already exist. Ignore the error message.) The `faa_otp_load` table is structured to match the format of the input data from the FAA Web site.

5. Create an external table definition with the same structure as the `faa_otp_load` table.

```
=# \i create_ext_table.sql
CREATE EXTERNAL TABLE faa.ext_load_otp
psql:./create_ext_table.sql:5: NOTICE: HEADER means that each one of the data
 files has a header row.
CREATE EXTERNAL TABLE
```

   This is a pure metadata operation. No data has moved from the data files on the host to the database yet. The external table definition references files in the `faa` directory that match the pattern `otp*.gz`. There are two matching files, one containing data for December 2009, the other for January 2010.

6. Move data from the external table to the `faa_otp_load` table.

```
=# INSERT INTO faa.faa_otp_load
   SELECT * FROM faa.ext_load_otp;
NOTICE: Found 26526 data formatting errors (26526 or more input rows). Rejected
 related input data.
INSERT 0 1024552
```

   Greenplum moves data from the gzip files into the load table in the database. In a production environment, you could have many `gpfdist` processes running, one on each host or several on one host, each on a separate port number.

7. Examine the errors briefly. (The `\x on` `psql` meta-command changes the display of the results to one line per column, which is easier to read for some result sets.)

```
=# \x on
Expanded display is on.
=# SELECT DISTINCT relname, errmsg, count(*)
   FROM faa.faa_load_errors
   GROUP BY 1,2;
-[ RECORD 1 ]-------------------------------------------------
relname | ext_load_otp
errmsg  | invalid input syntax for integer: "", column deptime
count   | 55614
```

   The problem is that, for some flights, the CSV file contains an empty string ("") in the `deptime` column where an integer is expected.

Loading Data with gpload

Greenplum provides a wrapper program for `gpfdist` called `gpload` that does much of the work of setting up the external table and the data movement.

In this exercise, you reload the `faa_otp_load` table using the `gpload` utility.

1. Since `gpload` executes `gpfdist`, you must first kill the `gpfdist` process you started in the previous exercise.

```
$ ps -A | grep gpfdist
5012 ttys000   0:00.00 grep gpfdist
2498 ttys001   0:03.38 gpfdist -d /Users/gpadmin/faa -p 8081
```

   Your process id will not be the same, so kill the appropriate one with the `kill` command, or just use the simpler `killall` command:

```
$ killall gpfdist
```

2. Edit and customize the `gpload.yaml` input file. Be sure to set the correct path to the `faa` directory. Notice the `TRUNCATE: true` preload instruction ensures that the data loaded in the previous exercise will be removed before the load in this exercise starts.

```
$ vi gpload.yaml
---
VERSION: 1.0.0.1
# describe the Greenplum database parameters
DATABASE: tutorial
USER: gpadmin
HOST: localhost
PORT: 5432
# describe the location of the source files
# in this example, the database master lives on the same host as the source files
GPLOAD:
   INPUT:
    - SOURCE:
         LOCAL_HOSTNAME:
           - gp-single-host
         PORT: 8081
         FILE:
           - /home/gpadmin/faa/otp*.gz
    - FORMAT: csv
    - QUOTE: '"'
    - ERROR_LIMIT: 50000
    - ERROR_TABLE: faa.faa_load_errors
   OUTPUT:
    - TABLE: faa.faa_otp_load
    - MODE: INSERT
   PRELOAD:
    - TRUNCATE: true
```

3. Execute `gpload` with the `gpload.yaml` input file. (Include the `-v` flag if you want to see details of the loading process.)

```
$ gpload -f gpload.yaml -l gpload.log
2014-01-27 15:05:24|INFO|gpload session started 2014-01-27 15:05:24
2014-01-27 15:05:24|INFO|started gpfdist -p 8081 -P 8082 -f "/home/gpadmin/faa/
otp*.gz" -t 30
2014-01-27 15:05:58|WARN|26528 bad rows
2014-01-27 15:05:58|INFO|running time: 34.55 seconds
2014-01-27 15:05:59|INFO|rows Inserted = 1024552
2014-01-27 15:05:59|INFO|rows Updated = 0
2014-01-27 15:05:59|INFO|data formatting errors = 0
2014-01-27 15:05:59|INFO|gpload succeeded with warnings
```

Create and load fact tables

The final step of the ELT process is to move data from the load table to the fact table. For the FAA example, you create two fact tables. The `faa.otp_r` table is a row-oriented table, which will be loaded with data from the faa.faa_otp_load table. The `faa.otp_c` table has the same structure as the `faa.otp_r` table, but is column-oriented and partitioned. You will load it with data from the `faa.otp_r` table. The two

tables will contain identical data and allow you to experiment with a column-oriented and partioned table in addition to a traditional row-oriented table.

1. Create the `faa.otp_r` and `faa.otp_c` tables by executing the `create_fact_tables.sql` script.

   ```
   =# \i create_fact_tables.sql
   ```

   Review the `create_fact_tables.sql` script and note that some columns are excluded from the fact table and the data types of some columns are cast to a different datatype. The MADlib routines usually require `float8` values, so the `numeric` columns are cast to `float8` as part of the transform step.
2. Load the data from the `faa_otp_load` table into the `faa.otp_r` table using the SQL `INSERT FROM` statement. Load the `faa.otp_c` table from the `faa.otp_r` table. Both of these loads can be accomplished by running the `load_into_fact_table.sql` script.

   ```
   =# \i load_into_fact_table.sql
   ```

Data loading summary

sele

The ability to load billions of rows quickly into the Greenplum database is one of its key features. Using “Extract, Load and Transform” (ELT) allows load processes to make use of the massive parallelism of the Greenplum system by staging the data (perhaps just the use of external tables) and then applying data transformations within Greenplum Database. Set-based operations can be done in parallel, maximizing performance.

With other loading mechanisms such as `COPY`, data is loaded through the master in a single process. This does not take advantage of the parallel processing power of the Greenplum segments. External tables provide a means of leveraging the parallel processing power of the segments for data loading. Also, unlike other loading mechanisms, you can access multiple data sources with one `SELECT` of an external table.

External tables make static data available inside the database. External tables can be defined with `file://` or `gpfdist://` protocols. `gpfdist` is a file server program that loads files in parallel. Since the data is static, external tables can be rescanned during a query execution.

External Web tables allow `http://` protocol or an `EXECUTE` clause to execute an operating system command or script. That data is assumed to be dynamic—query plans involving Web tables do not allow rescanning because the data could change during query execution. Execution plans may be slower, as data must be materialized (I/O) if it cannot fit in memory.

The script or process to populate a table with external Web tables may be executed on every segment host. It is possible, therefore, to have duplication of data. This is something to be aware of and check for when using Web tables, particularly with SQL extract calls to another database.

Queries and performance tuning

This section introduces some of the basic principles of query and performance tuning in a Greenplum database.

Some items to consider in performance tuning:

- `VACUUM` and `ANALYZE`
- Explain plans
- Indexing
- Column or row orientation
- Set based vs. row based
- Distribution and partitioning

VACUUM and ANALYZE

Greenplum uses Multiversion Concurrency Control (MVCC) to guarantee isolation, one of the ACID properties of relational databases. MVCC enables multiple users of the database to obtain consistent results for a query, even if the data is changing as the query is being executed. There can be multiple versions of rows in the database, but a query sees a snapshot of the database at a single point in time, containing only the versions of rows that were valid at that point in time. When a row is updated or deleted and no active transactions continue to reference it, it can be removed. The `VACUUM` command removes older versions that are no longer needed, leaving free space that can be reused.

In a Greenplum database, normal OLTP operations do not create the need for vacuuming out old rows, but loading data while tables are in use may. It is a best practice to `VACUUM` a table after a load. If the table is partitioned, and only a single partition is being altered, then a `VACUUM` on that partition may suffice.

The `VACUUM FULL` command behaves much differently than `VACUUM`, and its use is not recommended in Greenplum databases. It can be expensive in CPU and I/O, cause bloat in indexes, and lock data for long periods of time.

The `ANALYZE` command generates statistics about the distribution of data in a table. In particular it stores histograms about the values in each of the columns. The query optimizer depends on these statistics to select the best plan for executing a query. For example, the optimizer can use distribution data to decide on join orders. One of the optimizer's goals in a join is to minimize the volume of data that must be analyzed and potententially moved between segments by using the statistics to choose the smallest result set to work with first.

Analyze the tables

After a load you should analyze the tables, using the `ANALYZE` command. This gathers statistics on the data distribution, which the optimizer uses to choose an explain plan. Unanalyzed tables lead to less than optimal explain plans and less than optimal query performance.

Run the `ANALYZE` command on each of the tables:

```
=> ANALYZE faa.d_airports;
ANALYZE
=> ANALYZE faa.d_airlines;
ANALYZE
=> ANALYZE faa.d_wac;
ANALYZE
=> ANALYZE faa.d_cancellation_codes;
ANALYZE
=> ANALYZE faa.faa_otp_load;
ANALYZE
=> ANALYZE faa.otp_r;
ANALYZE
```

```
=> ANALYZE faa.otp_c;
ANALYZE
```

View explain plans

An explain plan explains the method the optimizer has chosen to produce a result set. Depending on the query, there can be a variety of methods to produce a result set. The optimizer calculates the cost for each method and chooses the one with the lowest cost. In large queries, cost is generally measured by the amount of I/O to be performed.

An explain plan does not do any actual query processing work. Explain plans use statistics generated by the `ANALYZE` command, so plans generated before and after running `ANALYZE` can be quite different. This is especially true for queries with multiple joins, because the order of the joins can have a tremendous impact on performance.

In the following exercise, you will generate some small tables that you can query and view some explain plans.

> **Note:** In the results in this section, long lines are reformatted for easier reading.

1. Enable timing so that you can see the effects of different performance tuning measures.

   ```
   => \timing on
   ```

2. View the `create_sample_table.sql` script, and then run it.

   ```
   => \i create_sample_table.sql
   DROP TABLE
   Time: 18.777 ms
   SET
   Time: 0.800 ms
   psql:create_sample_table.sql:3: NOTICE: CREATE TABLE will create implicit sequence
    "sample_id_seq" for serial column
   "sample.id"
   CREATE TABLE
   Time: 12.718 ms
   INSERT 0 1000000
   Time: 32776.929 ms
   UPDATE 1000000
   Time: 1411.764 ms
   UPDATE 50000
   Time: 680.276 ms
   UPDATE 1000000
   Time: 1748.169 ms
   ```

   The script has created the `sample` table, inserted one million rows, and some data.

3. Request the explain plan for the `COUNT()` aggregate.

   ```
   => EXPLAIN SELECT COUNT(*) FROM sample WHERE id > 100;
                              QUERY PLAN
   ------------------------------------------------------------------
    Aggregate  (cost=16367.17..16367.18 rows=1 width=8)
      ->  Gather Motion 2:1  (slice1; segments: 2)  (cost=16367.11..16367.15 rows=1
    width=8)
            ->  Aggregate  (cost=16367.11..16367.12 rows=1 width=8)
                  ->  Seq Scan on sample  (cost=0.00..13863.80 rows=500662 width=0)
                        Filter: id > 100
   (5 rows)

   Time: 4.071 ms
   ```

   Query plans are read from bottom to top. In this example, there are four steps. First there is a sequential scan on each segment server to access the rows. Then there is an aggregation on each segment server to produce a count of the number of rows from that segment. Then there is a gathering

of the count value to a single location. Finally, the counts from each segment are aggregated to produce the final result.

The cost number on each step has a start and stop value. For the sequential scan, this begins at time zero and goes until 13863.80. This is a fictional number created by the optimizer—it is not a number of seconds or I/O operations.

The cost numbers are cumulative, so the cost for the second operation includes the cost for the first operation. Notice that nearly all the time to process this query is in the sequential scan.

4. The `EXPLAIN ANALYZE` command actually runs the query (without returning the result set). The cost numbers reflect the actual timings. It also produces some memory and I/O statistics.

```
=> EXPLAIN ANALYZE SELECT COUNT(*)
FROM sample
WHERE id > 100;
                     QUERY PLAN
------------------------------------------------------------
 Aggregate  (cost=16367.17..16367.18 rows=1 width=8)
   Rows out:  1 rows with 427 ms to end, start offset by 0.649 ms.
   ->  Gather Motion 2:1  (slice1; segments: 2)  (cost=16367.11..16367.15 rows=1
 width=8)
         Rows out:  2 rows at destination with 425 ms to first row, 427 ms to end,
 start offset b
y 0.650 ms.
         ->  Aggregate  (cost=16367.11..16367.12 rows=1 width=8)
               Rows out:  Avg 1.0 rows x 2 workers.  Max 1 rows (seg0) with 424 ms
 to end, start
offset by 1.772 ms.
               ->  Seq Scan on sample  (cost=0.00..13863.80 rows=500662 width=0)
                     Filter: id > 100
                     Rows out:  Avg 499950.0 rows x 2 workers.  Max 499951 rows
 (seg0) with 90 ms
 to first row, 171 ms to end, start offset by 1.773 ms.
 Slice statistics:
   (slice0)    Executor memory: 159K bytes.
   (slice1)    Executor memory: 163K bytes avg x 2 workers, 163K bytes max (seg0).
 Statement statistics:
   Memory used: 128000K bytes
 Total runtime: 428.059 ms
(15 rows)
```

Indexes and performance

Greenplum Database does not depend upon indexes to the same degree as conventional data warehouse systems. Because the segments execute table scans in parallel, each segment scanning a small segment of the table, the traditional performance advantage from indexes is diminished. Indexes consume large amounts of space and require considerable CPU time to compute during data loads.

There are, however, times when indexes are useful, especially for highly selective queries. When a query looks up a single row, an index can dramatically improve performance.

In this exercise, you first run a single row lookup on the `sample` table without an index, then rerun the query after creating an index.

1. Select a single row and note the time to execute the query.

```
=> SELECT * from sample WHERE big = 12345;
  id   |  big  | wee | stuff
-------+-------+-----+-------
 12345 | 12345 |   0 |
(1 row)

Time: 148.671 ms
```

2. View the explain plan for the previous query:

```
=> EXPLAIN SELECT * FROM sample WHERE big = 12345;
                        QUERY PLAN
------------------------------------------------------------
 Gather Motion 2:1  (slice1; segments: 2)  (cost=0.00..13863.80 rows=1 width=46)
   ->  Seq Scan on sample  (cost=0.00..13863.80 rows=1 width=46)
         Filter: big = 12345
(3 rows)

Time: 0.931 ms
```

3. Create an index on the sample table.

```
=# CREATE INDEX sample_big_index ON sample(big);
CREATE INDEX
Time: 1196.005 ms
```

4. View the explain plan for the single-row select query with the new index in place:

```
=# EXPLAIN SELECT * FROM sample WHERE big = 12345;
                     QUERY PLAN
------------------------------------------------------------
 Gather Motion 2:1  (slice1; segments: 2)  (cost=0.00..200.61 rows=1 width=46)
   ->  Index Scan using sample_big_index on sample  (cost=0.00..200.61 rows=1
 width=46)
         Index Cond: big = 12345
(3 rows)

Time: 5.341 ms
```

5. Run the single-row SELECT query with the index in place and note the time.

```
=# SELECT * FROM sample WHERE big = 12345;
   id   |  big  | wee | stuff
-------+-------+-----+-------
 12345 | 12345 |   0 |
(1 row)

Time: 5.810 ms
```

Notice the difference in timing between the single-row SELECT with and without the index. The difference would have been much greater for a larger table. Note that even when there is a index, the optimizer can choose not to use it if it calculates a more efficient plan.

6. View the following explain plans to compare plans for some other common types of queries.

```
=# EXPLAIN SELECT * FROM sample WHERE big = 12345;

=# EXPLAIN SELECT * FROM sample WHERE big > 12345;

=# EXPLAIN SELECT * FROM sample
   WHERE big = 12345 OR big = 12355;

=# DROP INDEX sample_big_index;

=# EXPLAIN SELECT * FROM sample
   WHERE big = 12345 OR big = 12355;
```

Row vs. column orientation

Greenplum Database offers the ability to store a table in either row or column orientation. Both storage options have advantages, depending upon data compression characteristics, the kinds of queries executed, the row length, and the complexity and number of join columns.

As a general rule, very wide tables are better stored in row orientation, especially if there are joins on many columns. Column orientation works well to save space with compression and to reduce I/O when there is much duplicated data in columns.

In this exercise, you will create a column-oriented version of the fact table and compare it with the row-oriented version.

1. Create a column-oriented version of the FAA On Time Performance fact table and insert the data from the row-oriented version.

```
=#
DROP TABLE IF EXISTS faa.otp_c;
CREATE TABLE faa.otp_c (LIKE faa.otp_r)
WITH (appendonly=true, orientation=column)
DISTRIBUTED BY (UniqueCarrier, FlightNum)
PARTITION BY RANGE(FlightDate)
( PARTITION mth
   START('2009-06-01'::date)
   END ('2010-10-31'::date)
   EVERY ('1 mon'::interval)
);
=# INSERT INTO faa.otp_c
SELECT * FROM faa.otp_r;
```

2. Compare the definitions of the row and column versions of the table.

```
=# \d faa.otp_r
                Table "faa.otp_r"
         Column         |       Type       |  Modifiers
 -----------------------+------------------+-----------
  flt_year              | smallint         |
  flt_quarter           | smallint         |
  flt_month             | smallint         |
  flt_dayofmonth        | smallint         |
  flt_dayofweek         | smallint         |
  flightdate            | date             |
  uniquecarrier         | text             |
  airlineid             | integer          |
  carrier               | text             |
  flightnum             | text             |
  origin                | text             |
  origincityname        | text             |
  originstate           | text             |
  originstatename       | text             |
  dest                  | text             |
  destcityname          | text             |
  deststate             | text             |
  deststatename         | text             |
  crsdeptime            | text             |
  deptime               | integer          |
  depdelay              | double precision |
  depdelayminutes       | double precision |
  departuredelaygroups  | smallint         |
  taxiout               | smallint         |
  wheelsoff             | text             |
  wheelson              | text             |
  taxiin                | smallint         |
```

```
  crsarrtime           | text             |
  arrtime              | text             |
  arrdelay             | double precision |
  arrdelayminutes      | double precision |
  arrivaldelaygroups   | smallint         |
  cancelled            | smallint         |
  cancellationcode     | text             |
  diverted             | smallint         |
  crselapsedtime       | integer          |
  actualelapsedtime    | double precision |
  airtime              | double precision |
  flights              | smallint         |
  distance             | double precision |
  distancegroup        | smallint         |
  carrierdelay         | smallint         |
  weatherdelay         | smallint         |
  nasdelay             | smallint         |
  securitydelay        | smallint         |
  lateaircraftdelay    | smallint         |
 Distributed by: (uniquecarrier, flightnum)
```

Notice that the column-oriented version is append-only and partitioned. It has seventeen child files for the partitions, one for each month from June 2009 through October 2010.

```
=# \d faa.otp_c
Append-Only Columnar Table "faa.otp_c"
        Column         |   Type           | Modifiers
-----------------------+------------------+-----------
 flt_year              | smallint         |
 flt_quarter           | smallint         |
 flt_month             | smallint         |
 flt_dayofmonth        | smallint         |
 flt_dayofweek         | smallint         |
 flightdate            | date             |
 uniquecarrier         | text             |
 airlineid             | integer          |
 carrier               | text             |
 flightnum             | text             |
 origin                | text             |
 origincityname        | text             |
 originstate           | text             |
 originstatename       | text             |
 dest                  | text             |
 destcityname          | text             |
 deststate             | text             |
 deststatename         | text             |
 crsdeptime            | text             |
 deptime               | integer          |
 depdelay              | double precision |
 depdelayminutes       | double precision |
 departuredelaygroups  | smallint         |
 taxiout               | smallint         |
 wheelsoff             | text             |
 wheelson              | text             |
 taxiin                | smallint         |
 crsarrtime            | text             |
 arrtime               | text             |
 arrdelay              | double precision |
 arrdelayminutes       | double precision |
 arrivaldelaygroups    | smallint         |
 cancelled             | smallint         |
 cancellationcode      | text             |
 diverted              | smallint         |
 crselapsedtime        | integer          |
 actualelapsedtime     | double precision |
 airtime               | double precision |
 flights               | smallint         |
 distance              | double precision |
```

```
 distancegroup        | smallint         |
 carrierdelay         | smallint         |
 eatherdelay          | smallint         |
 nasdelay             | smallint         |
 securitydelay        | smallint         |
 lateaircraftdelay    | smallint         |
Checksum: f
Number of child tables: 17 (Use \d+ to list them.)
Distributed by: (uniquecarrier, flightnum)
```

3. Compare the sizes of the tables using the `pg_relation_size()` and `pg_total_relation_size()` functions. The `pg_size_pretty()` function converts the size in bytes to human-readable units.

```
=# SELECT pg_size_pretty(pg_relation_size('faa.otp_r'));
 pg_size_pretty
----------------
 256 MB
(1 row)
```

```
=# SELECT pg_size_pretty(pg_relation_size('faa.otp_c'));
 pg_size_pretty
----------------
 0 bytes
(1 row)
```

```
=# SELECT pg_size_pretty(pg_total_relation_size('faa.otp_c'));
 pg_size_pretty
----------------
 288kB bytes
(1 row)
```

The `pg_relation_size()` function for the column-oriented table returns 0. This is because the `pg_relation_size()` function excludes the sizes of partitions. The `pg_total_relation_size()` function is the sum of the sizes of all the partitions. The column-oriented table is much smaller than the row-oriented version, although no compression was specified. Column values, which are all the same type, can be stored more compactly than rows with mixed data types.

Check for even data distribution on segments

The `faa.otp_r` and `faa.otp_c` tables are distributed with a hash function on `UniqueCarrier` and `FlightNum`. These two columns were selected because they produce an even distribution of the data onto the segments. Also, with frequent joins expected on the fact table and dimension tables on these two columns, less data moves between segments, reducing query execution time.

When there is no advantage to co-locating data from different tables on the segments, a distribution based on a unique column ensures even distribution. Distributing on a column with low cardinality, such as `Diverted`, which has only two values, will yield a poor distribution.

1. One of the goals of distribution is to ensure that there is approximately the same amount of data in each segment. The query below shows one way of determining this. Since the column-oriented and row-oriented tables are distributed by the same columns, the counts should be the same for each.

```
=# SELECT gp_segment_id, COUNT(*)
FROM faa.otp_c
GROUP BY gp_segment_id
ORDER BY gp_segment_id;
 gp_segment_id |  count
---------------+---------
             0 | 1028144
             1 | 1020960
(2 rows)
```

About partitioning

Partitioning a table can improve query performance and simplify data administration. The table is divided into smaller child files using a range or a list value, such as a date range or a country code.

Partitions can improve query performance dramatically. When a query predicate filters on the same criteria used to define partitions, the optimizer can avoid searching partitions that do not contain relevant data.

A common application for partitioning is to maintain a rolling window of data based on date, for example, a fact table containing the most recent 12 months of data. Using the `ALTER TABLE` statement, an existing partition can be dropped by removing its child file. This is much more efficient than scanning the entire table and removing rows with a `DELETE` statement.

Partitions may also be subpartitioned. For example, a table could be partitioned by month, and the month partitions could be subpartitioned by week. Greenplum Database creates child files for the months and weeks. The actual data, however, is stored in the child files created for the week subpartitions—only child files at the leaf level hold data.

When a new partition is added, you can run `ANALYZE` on just the data in that partition. `ANALYZE` can run on the root partition (the name of the table in the CREATE TABLE statement) or on a child file created for a leaf partition. If `ANALYZE` has already run on the other partitions and the data is static, it is not necessary to run it again on those partitions.

The following exercise compares `SELECT` statements with `WHERE` clauses that do and do not use a partitioned column.

1. The column-oriented version of the fact table you created is partitioned by date. First, execute a query that filters on a non-partitioned column and note the execution time.

```
=# \timing on
=# SELECT MAX(depdelay) FROM faa.otp_c
   WHERE UniqueCarrier = 'UA';
 max
------
 1360
(1 row)

Time: 148.333 ms
```

2. Execute a query that filters on `flightdate`, the partitioned column.

```
=# SELECT MAX(depdelay) FROM faa.otp_c
    WHERE flightdate = '2009-12-03';
 max
-----
 982
(1 row)

Time: 35.013 ms
```

The query on the partitioned column takes much less time to execute because only one partition must be scanned.

If you compare the explain plans for the queries in this exercise, you will see that the first query scans each of the seventeen child files, while the second scans just one child file. The reduction in I/O and CPU time explains the improved execution time.

Chapter 5

Introduction to Greenplum In-Database Analytics

Running analytics directly in Greenplum Database, rather than exporting data to a separate analytics engine, allows greater agility when exploring large data sets and much better performance due to parallelizing the analytic processes across all the segments.

A variety of power analytic tools is available for use with Greenplum Database:

- MADlib, an open-source, MPP implementation of many analytic algorithms, available at *http://madlib.net*
- R statistical language
- SAS, in many forms, but especially with the SAS Accelerator for Greenplum
- PMML, Predictive Modeling Markup Language

The exercises in this chapter introduce using MADlib with Greenplum Database, using the FAA on-time data example dataset. You will examine scenarios comparing airlines and airports to learn whether there are significant relationships to be found.

Run PostgreSQL built-in aggregates

PostgreSQL has built-in aggregate functions to get standard statistics on database columns—minimum, maximum, average, and standard deviation, for example. The functions take advantage of the Greenplum Database MPP architecture, aggregating data on the segments and then assembling results on the master.

First, gather simple descriptive statistics on some of the data you will analyze with MADlib. The commands in this exercise are in the `stats.sql` script in the sample data directory.

1. Get average delay, standard deviation, and number of flights for USAir and Delta airlines.

```
=# SELECT carrier, AVG(arrdelayminutes),
STDDEV(arrdelayminutes), COUNT(*)
FROM faa.otp_c
WHERE carrier = 'US' OR carrier = 'DL'
GROUP BY carrier;
 carrier |       avg        |      stddev      | count
---------+------------------+------------------+-------
 US      | 10.9447897117189 | 28.0561845925158 | 65268
 DL      | 11.2861567151753 | 34.7066907557035 | 90339
(2 rows)
```

2. Get average delay, standard deviation, and number of flights originating from Chicago O'Hare or Atlanta Hartsfield airports.

```
=# SELECT origin, AVG(arrdelayminutes),
STDDEV(arrdelayminutes), COUNT(*)
FROM faa.otp_c
WHERE origin = 'ORD' OR origin = 'ATL'
GROUP BY origin;
 origin |       avg        |      stddev      | count
--------+------------------+------------------+-------
 ATL    | 12.3588880294827 | 29.8344526603625 | 66217
 ORD    |  18.018366261709 | 39.0627644508841 | 49135
(2 rows)
```

3. Get average delay, standard deviation, and number of flights originating from any of three Delta hubs.

```
=# SELECT origin, AVG(arrdelayminutes),
STDDEV(arrdelayminutes), COUNT(*)
FROM faa.otp_c
WHERE carrier = 'DL' AND origin IN ('ATL', 'MSP', 'DTW')
GROUP BY origin;
 origin |       avg        |      stddev      | count
--------+------------------+------------------+-------
 DTW    | 12.4719246398227 | 35.5776257603982 |  5430
 ATL    |  10.118265281469 | 27.3159945593657 | 24899
 MSP    | 11.6270039187745 | 31.7119612288857 |  5635
(3 rows)
```

4. Get average delay, standard deviation, and number of flights for Delta and USAir flights originating from Atlanta Harsfield.

```
=# SELECT carrier, AVG(arrdelayminutes),
 STDDEV(arrdelayminutes), COUNT(*)
FROM faa.otp_c
WHERE carrier IN ('DL', 'UA') AND origin = 'ATL'
GROUP BY carrier;
 carrier |       avg        |      stddev      | count
---------+------------------+------------------+-------
 DL      |  10.118265281469 | 27.3159945593657 | 24899
 UA      | 10.5479452054795 | 30.9478052682434 |   146
(2 rows)
```

Install MADlib

MADlib is a library of user-defined database functions and types that run in Greenplum Database and PostgreSQL databases. There are two parts to installing MADlib. First, you download and install the MADlib distribution. This step installs MADlib on your computer's file system. Second, use the `madpack` utility to install the MADlib library into the databases you want to use it with.

Download and install MADlib

MADlib distributions are available for Linux (Red Hat and CentOS), Mac OS X, and Pivotal HAWQ. You can download a package for your OS from *http://madlib.net* and install it using the package manager on your OS, or obtain a Greenplum installation package (`gppkg`) from your Pivotal representative.

- For Mac OS X, click the `.dpg` file and follow the installer's instructions.
- For Linux, use the `yum` command to install the package you downloaded:

```
$ sudo yum install madlib-x.x-Linux.rpm --nogpgcheck
```

If you are installing MADlib on a multi-node Greenplum Database instance, you must install MADlib on each segment host also. The `gpscp` and `gpssh` utility programs make it easy to copy the installation package to the segments and execute the installer.

If you are on a Linux distribution that does not use RPM, you can download and compile the source. The instructions for building from source are on the MADlib Web site.

Install the MADlib library into a database

With the MADlib distribution installed on your system, you use the `madpack` utility to install the MADlib database objects into the database. The MADlib distribution is by default installed in the `/usr/local/madlib` directory.

The following steps install MADlib on a single-node Greenplum Database installation.

1. Make sure the Greenplum Database environment variables are set in your shell by sourcing the `/usr/local/greenplum-db/greenplum_path.sh` file.

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

2. Register MADlib in the `tutorial` database using the `madpack` utility:

```
$ /usr/local/madlib/bin/madpack -p greenplum \
-c gpadmin@localhost/tutorial install
Password for user gpadmin:
madpack.py : INFO : Detected Greenplum DB version 4.3ORCA.
madpack.py : INFO : *** Installing MADlib ***
madpack.py : INFO : MADlib tools version    = 1.8 (/usr/local/madlib/Versions/1.8/
bin/../madpack/madpack.py)
madpack.py : INFO : MADlib database version = None (host=localhost:5432,
 db=tutorial, schema=madlib)
madpack.py : INFO : Testing PL/Python environment...
madpack.py : INFO : > Creating language PL/Python...
madpack.py : INFO : > PL/Python environment OK (version: 2.6.2)
madpack.py : INFO : Installing MADlib into MADLIB schema...
madpack.py : INFO : > Creating MADLIB schema
madpack.py : INFO : > Creating MADLIB.MigrationHistory table
madpack.py : INFO : > Writing version info in MigrationHistory table
madpack.py : INFO : > Creating objects for modules:
madpack.py : INFO : > - array_ops
madpack.py : INFO : > - bayes
madpack.py : INFO : > - crf
madpack.py : INFO : > - elastic_net
```

```
madpack.py : INFO : > - linalg
madpack.py : INFO : > - pmml
madpack.py : INFO : > - prob
madpack.py : INFO : > - quantile
madpack.py : INFO : > - sketch
madpack.py : INFO : > - svd_mf
madpack.py : INFO : > - svec
madpack.py : INFO : > - tsa
madpack.py : INFO : > - conjugate_gradient
madpack.py : INFO : > - data_profile
madpack.py : INFO : > - lda
madpack.py : INFO : > - stats
madpack.py : INFO : > - svec_util
madpack.py : INFO : > - utilities
madpack.py : INFO : > - assoc_rules
madpack.py : INFO : > - cart
madpack.py : INFO : > - convex
madpack.py : INFO : > - glm
madpack.py : INFO : > - kernel_machines
madpack.py : INFO : > - linear_systems
madpack.py : INFO : > - recursive_partitioning
madpack.py : INFO : > - regress
madpack.py : INFO : > - sample
madpack.py : INFO : > - summary
madpack.py : INFO : > - kmeans
madpack.py : INFO : > - pca
madpack.py : INFO : > - validation
madpack.py : INFO : MADlib 1.8 installed successfully in MADLIB schema.
```

The `madpack` utility lists the MADlib modules as they are installed into the database.

The `madpack` utility can be used to install, uninstall, upgrade, and test the MADlib objects in the database. See the MADlib Web site for complete documentation.

By default, the MADlib objects are installed in the `madlib` schema. You can add the `madlib` schema to your search path (see *Create a schema and set search path*) so you can call MADlib functions without the `madlib` prefix.

Run MADlib ANOVA

ANOVA (Analysis of Variance) shows whether groups of samples are significantly different from each other. The MADlib ANOVA function uses an integer value to distinguish between the groups to compare and a column for the data. The groups we want to analyze in the FAA fact table are text in the data, so we use a PostgreSQL `CASE` statement to assign the samples to integer values based on the text values. The ANOVA module then divides the rows into groups and performs the test.

ANOVA is a general linear model. To determine whether statistical data samples are significantly different from one another, you compare the total variability of the group by the variability between the groups. This is tempered by the number of observations, which is summarized by the degrees of freedom within the groups. The relevant statistic that measures the degree to which the difference between groups is significant is the ratio of the variance between groups divided by the variance within groups, called the F statistic. If it is close to zero, the groups do not differ by much. If it is far from zero, they do.

From statistical theory you can determine the probability distribution of the F statistic if the groups were identical given sampling error. This is given by the p-value. A p-value close to zero indicates it is very likely that the groups are different. A p-value close to one indicates that it is very likely the groups are the same.

The `\x psql` command displays each row of output in a column. This format is easier to read when there are many columns.

The commands in this section are in the `one_way_anova.sql` script in the sample data directory.

1. Run an ANOVA analysis on the average delay minutes between USAir and Delta airlines. The `CASE` clause assigns USAir flights to group 1 and Delta flights to group 2.

```
=# \x Expanded display is on.
=# SELECT (MADlib.one_way_anova (
   CASE WHEN carrier = 'US' THEN 1
        WHEN carrier = 'DL' THEN 2
        ELSE NULL
   END,
   arrdelayminutes
)).* FROM faa.otp_r;
-[ RECORD 1 ]--------+-------------------
sum_squares_between  | 4400.13955936305
sum_squares_within   | 159682139.016315
df_between           | 1
df_within            | 155091
mean_squares_between | 4400.13955936305
mean_squares_within  | 1029.60287196752
statistic            | 4.27362789980821
p_value              | 0.0387098585959039
```

2. Run an ANOVA analysis to determine if the average delays for flights from Chicago and Atlanta are statistically different.

```
=# SELECT (MADlib.one_way_anova (
    CASE WHEN origin = 'ORD' THEN 1
         WHEN origin = 'ATL' THEN 2
         ELSE NULL
    END,
     arrdelayminutes
)).* FROM faa.otp_r;
-[ RECORD 1 ]--------+----------------------
sum_squares_between  | 899245.747556926
sum_squares_within   | 133295082.780767
df_between           | 1
df_within            | 114829
mean_squares_between | 899245.747556926
mean_squares_within  | 1160.81375593941
statistic            | 774.668410807376
```

```
p_value              | 6.39815750308269e-170
```

3. Run an ANOVA analysis to determine if the differences in average delay minutes from three Delta hubs are significant.

```
=# SELECT (MADlib.one_way_anova (
   CASE WHEN carrier = 'DL' AND origin = 'ATL' THEN 1
        WHEN carrier = 'DL' AND origin = 'MSP' THEN 2
        WHEN carrier = 'DL' AND origin = 'DTW' THEN 3
        ELSE NULL
   END,
  arrdelayminutes
)).* FROM faa.otp_r;
-[ RECORD 1 ]--------+--------------------
sum_squares_between  | 30223.5187869544
sum_squares_within   | 31025783.8336195
df_between           | 2
df_within            | 35859
mean_squares_between | 15111.7593934772
mean_squares_within  | 865.216091737625
statistic            | 17.4658787992813
p_value              | 2.62033860219297e-08
```

4. Run an ANOVA analysis to determine if the differences in average delay minutes between Delta and USAir flights from Atlanta are significant.

```
=# SELECT (MADlib.one_way_anova (
   CASE WHEN carrier = 'DL' AND origin = 'ATL' THEN 1
        WHEN carrier = 'UA' AND origin = 'ATL' THEN 2
        ELSE NULL
   END,
  arrdelayminutes
)).* FROM faa.otp_r;
sum_squares_between  | 26.7976816595111
sum_squares_within   | 18668355.8192519
df_between           | 1
df_within            | 24978
mean_squares_between | 26.7976816595111
mean_squares_within  | 747.391937675229
statistic            | 0.0358549247170977
p_value              | 0.849816939486254
```

From these ANOVA analyses we have learned the following:

- There is a fairly certain difference between delays for USAir and Delta, but the difference is not great
- Delays from O'Hare seem to be significantly different than from Atlanta
- There is a large difference between delays at the three Delta hubs
- There is no significant difference in delays from Atlanta between United and Delta.

Perform linear regression

Linear regression shows the relationship between variables. A classic example is the linear relationship between height and weight of adult males in a particular country or ethnic group. MADlib includes modules to perform linear regression with one or multiple independent variables.

The *r2* statistic measures the proportion of the total variability in the dependent variable that can be explained by the independent variable.

1. Perform a linear regression to see if there is any relationship between distance and arrival delay. This tests the hypothesis that longer flights are more likely to be on time because the flight crew can make up delays by flying faster over longer periods of time. Test this by running a regression on arrival time as the dependent variable and distance as the independent variable. This command is in the `linear1.sql` script in the sample data directory.

```
=# SELECT ( madlib.linregr(
            arrdelayminutes,
             ARRAY[1,distance])
          ).* FROM faa.otp_c;
-[ RECORD 1 ]-
+-----------------------------------------------------------------------------------------
coef          | {13.3842874825104,6.92849635192299e-05}
r2            | 1.33269205069481e-06
std_err       | {0.05491177998136,5.94079738957443e-05}
t_stats       | {243.741643178454,1.16625696814402}
p_values      | {0,0.243510565833601}
condition_no  | 1514.53275290688
num_processed | 1020608
vcov          | {{0.00301530358072129,-2.58421962379574e-06},
{-2.58421962379574e-06,3.52930736239744e-09}}
```

   The regression shows that *r2* is close to zero, which means that distance is not a good predictor for arrival delay time.
2. Run a regression with departure delay time as the independent variable and arrival delay time as the dependent variable. This tests the hypothesis that if a flight departs late, it is unlikely that the crew can make up the time. This command is in the `linear2.sql` script in the sample data directory.

```
=# SELECT ( madlib.linregr(
            arrdelayminutes,
             ARRAY[1,depdelayminutes])
).* FROM faa.otp_c;
-[ RECORD 1 ]-
+-----------------------------------------------------------------------------------------
coef          | {1.25027293128434,0.963608047925262}
r2            | 0.899877141891865
std_err       | {0.0113416459093601,0.000318160579519938}
t_stats       | {110.237344850672,3028.68460127655}
p_values      | {0,0}
condition_no  | 38.130629020182
num_processed | 1020608
vcov          | {{0.000128632931933304,-1.27999657307789e-06},
{-1.27999657307789e-06,1.01226154360463e-07}}
```

The r2 statistic is very high, especially with 1.5 million samples. The linear relationship can be written as

*Arrival_delay = 1.25027293128434 + 0.963608047925262 * departure_delay*

The `condition_no` result is a measure of the mathematical stability of the solution. In computer arithmetic, numbers do not have infinite precision, and round-off error in calculations can be significant, especially if there are a large number of independent variables and they are highly correlated. This is very common in econometric data and techniques have evolved to deal with it.

Learn more about MADlib

MADlib is an open source project on *GitHub*. You can find source code for the latest release and information about participating in the project in the GitHub repository. Access the MADlib user documentation on the MADlib Web site at *http://doc.madlib.net*.

Appendix A

Operating System Configuration

This appendix provides information about platform support, prerequisites, and operating system configuration for the tutorial in this guide. See the release notes for your Greenplum Database release for supported platforms and prerequisites. Before you install Greenplum Database, check the prerequistes and adjust your OS configuration parameters.

Linux

Supported Operating System Versions

- Red Hat Enterprise Linux 64-bit 6.x
- Red Hat Enterprise Linux 64-bit 5.x
- CentOS 64-bit 6.x
- CentOS 64-bit 5.x

OS Configuration

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

```
kernel.shmmax = 500000000
kernel.shmmni = 4096
kernel.shmall = 4000000000
kernel.sem = 250 64000 100 512
net.ipv4.tcp_tw_recycle=1
net.ipv4.tcp_max_syn_backlog=4096
net.core.netdev_max_backlog=10000
vm.overcommit_memory=2
```

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

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

Mac OS X

Supported Operating System Versions

- Mac OS X 10.5 and higher

Add the following to `/etc/sysctl.conf`:

```
kern.sysv.shmmax=2147483648
kern.sysv.shmmin=1
kern.sysv.shmmni=64
kern.sysv.shmseg=16
kern.sysv.shmall=524288
kern.maxfiles=65535
kern.maxfilesperproc=65535
net.inet.tcp.msl=60
```

If using DHCP, add the following line to `/etc/hostconfig`:

```
HOSTNAME="your_hostname"
```

You would then use this host name to initialize your Greenplum Database system (instead of `localhost`).

> **Note:** You must disable IPv6 addressing on Mac OS X before using Greenplum Database. Greenplum management scripts use IPv4 addressing by default. Although most platforms match IPv4 address formats to IPv6 addresses, this matching does not occur on Mac OS X systems.